Oracle® Fusion Middleware

API Gateway User Guide

11g Release 2 (11.1.2.1.0)

January 2013

ORACLE®

Oracle API Gateway User Guide, 11g Release 2 (11.1.2.1.0)

Contents

1. Getting Started ....
1. Oracle API Gateway Overview 31
Overview 31
Integration 31
Performance 31
Governance 32
Security 32
2. Oracle API Gateway Architecture 34
Overview 34
Basic Architecture 34
Policy Development 35
API Gateway Administration 36
API Service Administration 36
System Administration 36
Managed Domain Architecture 36
3. Oracle API Gateway Concepts 40
Overview 40
Product Concepts 40
4. Starting the API Gateway Tools 46
Overview 46
Before you Begin 46
Launching API Gateway Manager 46
Starting Policy Studio 47
Getting Started Tutorial 47
5. Virtualizing a Service 48
Overview 48
Accessing the Example Service 48
Creating a Workspace in API Service Manager 49
Step 1—Basic Information 49
Step 2—Service Exposure 49
Step 3—Request Processing 49
Step 4—Routing 49
Step 5—Response Processing 49
Step 6—Monitoring 49
Step 7—Tags 50
Deploying to a Group 50
Accessing the Virtualized Service 50
Monitoring a Service 51
6. Monitoring Services 52
Overview 52
Enabling Monitoring 52
Viewing Real-time Monitoring 52
Viewing Message Traffic 53
Viewing Message Content 54
Viewing Performance Statistics 55
Detecting Malformed Messages 55
Monitoring System Data 56
Configuring Trace and Log Settings 57
Using Oracle API Gateway Analytics 57
7. Troubleshooting 58
Overview 58
Viewing API Gateway Trace Files 58

Setting API Gateway Trace Levels 58
Configuring API Gateway Trace Files 59
Running Trace at DEBUG level 60
Running Trace at DATA level 61
Integrating Trace Output with Apache log4J 63
Configuring Logging Output 63
Configuring Log Level and Message 63
Getting Help 64
8. License Acknowledgments 65
Overview 65
Acknowledgments 65
9. Oracle Contact Details 66
Contact Details 66
2. Sample Policies ....
1. Configuring the Sample Policies 67
Overview 67
Enabling the Sample Services Interface 67
Configuring a Different Sample Services Interface 68
StockQuote Demo Service 68
Remote Host Settings 69
2. Conversion Sample Policy 71
Overview 71
REST to SOAP Policy 71
Running the Conversion Sample 72
3. Security Sample Policies 73
Overview 73
Signature Verification 73
Encryption and Decryption 74
4. Throttling Sample Policy 77
Overview 77
Throttling Policy 77
Running the Throttling Sample 77
5. Virtualized Service Sample Policy 79
Overview 79
Virtualized Service policies 79
Running the Virtualized Service Sample 83
6. Stress Testing with Send Request (SR) 84
Overview 84
Basic SR Examples 84
Advanced SR Examples 85
SR Arguments 85
7. Sending a Request with API Gateway Explorer 87
Overview 87
Creating a Request in API Gateway Explorer 87
Further Information 88
3. Managing API Services ....
1. Introduction to API Service Manager 89
Overview 89
Accessing API Service Manager 89
Deploying to a Group 89
Resetting your Configuration 90
2. Managing API Services 91
Overview 91
Virtualizing a Service in API Service Manager 91
Step 1—Basic Information 91
Step 2—Service Exposure 92
Step 3—Request Processing 92

Step 4—Routing ... 92
Step 5—Response Processing ... 92
Step 6—Monitoring ... 93
Step 7—Tags ... 93
Deploying to a Group ... 94

4. Governance ...

1. Configuring Security Policies from WSDL Files ... 95
Overview ... 95
Importing a WSDL File ... 95
Configuring Policy Settings ... 96
Configuring Policy Filters ... 96
Editing a Policy ... 99
Removing Security Tokens ... 99
Further Information ... 101

2. Securing a Virtual Service using Policies ... 102
Overview ... 102
Importing a WSDL File ... 102
Configuring a Security Policy ... 103
Configuring Policy Settings ... 103
Configuring Policy Filters ... 103
Editing a Security Policy ... 105
Using WCF WS-Policies ... 105
Removing Security Tokens ... 107
Further Information ... 108

3. Configuring Policies Manually ... 109
Overview ... 109
Configuration ... 109

4. Configuring Global Policies ... 111
Overview ... 111
Global Policy Roles ... 111
Selecting a Global Policy ... 112
Configuring Global Policies in a Policy Shortcut Chain ... 113
Configuring Global Policies for a Service ... 115
Showing Global Policies ... 115

5. Configuring Policy Packages ... 117
Overview ... 117
Configuring a Policy Package ... 117
Applying a Policy Package ... 118
Applying a Policy Package to a Service ... 118

5. Managing Deployments ...

1. Getting Started with Managing Deployments ... 119
Overview ... 119
Connecting to a Server in the Policy Studio ... 119
Editing Server Configuration in the Policy Studio ... 119
Managing Deployments in the API Gateway Manager ... 120
Managing Admin Users in the API Gateway Manager ... 120
Configuring Policies in the Policy Studio ... 120

2. Deploying Configuration ... 121
Overview ... 121
Creating a Deployment Archive in the Policy Studio ... 121
Deploying a Deployment Archive in the Policy Studio ... 121
Deploying a Factory Configuration in the Policy Studio ... 122
Deploying a Currently Loaded Configuration in the Policy Studio ... 122
Deployment Summary in the Policy Studio ... 122
Deploying an Archive in API Gateway Manager ... 123
Deploying on the Command Line ... 123

3. Deploying the API Gateway in Multiple Environments ... 124

Overview 124
Configuring Environment Variables 124
Configuring Certificates as Environment Variables 125
4. Managing Admin Users 128
Overview 128
Admin User Privileges 128
Admin User Roles 128
Adding a New Admin User 129
Removing an Admin User 130
Resetting an Admin User Password 130
Managing Admin User Roles 130
5. Configuring Role-Based Access Control (RBAC) 131
Overview 131
Local Admin User Store 133
Access Control List 134
Configuring Users and Roles 135
Management Service Roles and Permissions 136
6. Using Active Directory for Authentication and RBAC of Management Services 139
Overview 139
Step 1: Create an Active Directory Group 139
Step 2: Create an Active Directory User 141
Step 3: Create an LDAP Connection 145
Step 4: Create an LDAP Repository 146
Step 5: Create a Test Policy for LDAP Authentication and RBAC 148
Step 6: Use the LDAP Policy to Protect Management Services 154
Adding an LDAP User with Limited Access to Management Services 154
7. Using OpenLDAP for Authentication and RBAC of Management Services 156
Overview 156
Step 1: Create an OpenLDAP Group for RBAC Roles 156
Step 2: Add RBAC Roles to the OpenLDAP RBAC Group 157
Step 3: Add Users to the OpenLDAP RBAC Group 160
Step 4: Create an LDAP Connection 161
Step 5: Create an OpenLDAP Repository 161
Step 6: Create a Test Policy for LDAP Authentication and RBAC 163
Step 7: Use the OpenLDAP Policy to Protect Management Services 169
6. General Configuration ....
1. Startup Instructions 170
Overview 170
Setting Passphrases 170
Starting the Node Manager 170
Starting the API Gateway 170
Connecting to the API Gateway 171
2. Connection Details 172
Overview 172
Connecting to a URL 172
Connecting to a File 173
Unlocking a Server Connection 173
3. Global Configuration 174
Overview 174
Server Configuration 174
API Gateway Settings 174
Web Services Repository 175
Processes 175
Policies 175
Certificates and Keys 176
API Gateway User Store 176
System Alerts 176

External Connections ... 176
Caches ... 177
Black list and White list ... 177
Schema Cache ... 178
Scripts ... 178
Stylesheets ... 178
References ... 178
4. Server Configuration ... 180
Overview ... 180
Deploy ... 180
5. API Gateway Settings ... 181
Overview ... 181
Default Settings ... 181
Audit Log ... 181
Namespace ... 181
MIME/DIME ... 181
Traffic Monitor ... 181
Metrics ... 181
Session Settings ... 181
Cache ... 181
Access Log ... 182
Security Service Module ... 182
Kerberos ... 182
Tivoli ... 182
6. Policy Studio Preferences ... 183
Overview ... 183
Management Services ... 183
Policy Colors ... 183
Proxy Settings ... 183
Runtime Dependencies ... 184
Server Connection ... 184
SSL Settings ... 184
Status Bar ... 185
Trace Level ... 185
Web and XML ... 185
WS-I Settings ... 185
7. Policy Studio Viewing Options ... 187
Overview ... 187
Filtering the Tree ... 187
Configuring Viewing Options ... 187
Configuring the Policy Filter Palette ... 187
8. Web Service Repository ... 188
Overview ... 188
Testing WS-I Compliance ... 188
Registering the WSDL File ... 188
Loading the WSDL File ... 189
Selecting WSDL Operations ... 189
WS-Policy Options ... 189
Deploy Policy ... 190
Secure Virtual Service ... 190
WSDL Import Summary ... 190
What is Created? ... 191
Publishing the WSDL ... 192
9. Setting the Encryption Passphrase ... 193
Encryption Passphrase Overview ... 193
Setting the Group Passphrase in the Policy Studio ... 193
Entering the Group Passphrase in the Policy Studio ... 193
Specifying the Passphrase in a File or on Startup ... 194

10. Default Settings 195
Overview 195
Settings 195
11. Namespace Settings 198
Overview 198
SOAP Namespace 198
Signature ID Attribute 198
WSSE Namespace 199
12. MIME/DIME Settings 200
Overview 200
Configuration 200
13. Session Settings 201
Overview 201
Configuration 201
14. Exporting API Gateway Configuration 202
Overview 202
What is Exported 202
Exporting Configuration Items 202
Exporting All API Gateway Configuration 203
15. Importing API Gateway Configuration 204
Overview 204
Importing Configuration 204
Viewing Differences 204
What is Imported 205
7. Reporting .....
1. Configuring the API Gateway for API Gateway Analytics 206
Overview 206
Connecting to the API Gateway 206
Configuring the Database Connection 206
Configuring the Database Logging 206
Configuring Monitoring Settings 207
Deploying to the API Gateway 207
2. Using Oracle API Gateway Analytics 208
Overview 208
Launching API Gateway Analytics 208
System 208
API Services 209
Remote Hosts 210
Clients 210
Audit Trail 211
Reports 211
Custom Reporting 211
3. Scheduled Reports 214
Overview 214
Database Configuration 214
Scheduled Reports Configuration 214
SMTP Configuration 214
4. Real-Time Monitoring Settings 216
Overview 216
Configuring Metrics Settings 216
Configuring Reports Settings 216
5. Configuring Traffic Monitoring 217
Overview 217
Configuration 217
6. Purging the Reports Database 219
Overview 219
Running the dbpurger Command 219

Example Commands ... 219

8. API Gateway Instances ...

1. Configuring API Gateway Instances ... 221
Overview ... 221
Add Remote Host ... 221
Add HTTP Services ... 221
Add SMTP Services ... 221
Add File Transfer Services ... 221
Add Policy Execution Scheduler ... 221
Messaging System ... 221
FTP Poller ... 222
Directory Scanner ... 222
POP Client ... 222
TIBCO ... 222
API Gateway Settings ... 222
API Gateway Logging ... 222
Cryptographic Acceleration ... 222

2. Configuring HTTP Services ... 223
Overview ... 223
HTTP Services Groups ... 223
HTTP and HTTPS Interfaces ... 224
HTTPS Interfaces Only ... 226
Relative Paths ... 229
Web Service Resolvers ... 231
Static Content Provider ... 233
Servlet Applications ... 234
Management Services ... 235
Changing the Management Services Port ... 236

3. Configuring SMTP Services ... 237
Overview ... 237
Adding an SMTP Service ... 237
Adding an SMTP Interface ... 238
Configuring Policy Handlers for SMTP Commands ... 238
Adding an HELO/EHLO Policy Handler ... 239
Adding an AUTH Policy Handler ... 239
Adding a MAIL Policy Handler ... 240
Adding a RCPT Policy Handler ... 241
Adding a DATA Policy Handler ... 242
SMTP Authentication ... 242
SMTP Content-Transfer-Encoding ... 243
Deployment Example ... 244

4. File Transfer Service ... 248
Overview ... 248
General Configuration ... 248
File Upload ... 249
Secure Services ... 250
Commands ... 250
Access Control ... 251
Messages ... 251
Directory ... 251

5. Policy Execution Scheduling ... 253
Overview ... 253
Cron Expressions ... 253
Adding a Schedule ... 255
Adding a Policy Execution Scheduler ... 255

6. FTP Poller ... 256
Overview ... 256

General Settings 256
Scan Details 256
Connection Type 257
FTP and FTPS Connections 257
FTPS Connections 257
SFTP Connections 258
7. Directory Scanner 259
Overview 259
Directory to Scan 259
Directory for Output 259
Completed Directory 260
Working Directory 260
Policy to Use 260
8. Packet Sniffers 261
Overview 261
Configuration 261
9. Messaging System 263
Overview 263
Configuring a JMS Service 263
Configuring a JMS Session 264
Configuring a JMS Consumer 264
Configuring the JMS Wizard 265
10. Remote Host Settings 266
Overview 266
General Settings 266
Address and Load Balancing Settings 267
Advanced Settings 267
11. Configuring an HTTP Watchdog 270
Overview 270
Configuration 270
12. Configuring Conditions for HTTP Interfaces 271
Overview 271
Requires Endpoint Condition 271
Requires Link Condition 272
13. POP Client 273
Overview 273
Configuration 273
14. TIBCO Integration 274
Overview 274
TIBCO Rendezvous Integration 274
TIBCO Enterprise Messaging Service Integration 274
15. Cryptographic Acceleration 275
Overview 275
General Configuration 275
Conversations for Crypto Engines 276
16. Cryptographic Acceleration Conversation: Request-Response 277
Conversations for Crypto Engines 277
17. TIBCO Rendezvous Daemon 278
Overview 278
Configuration 278
18. TIBCO Rendezvous Listener 280
Overview 280
Configuration 280
19. TIBCO Enterprise Messaging Service Consumer 281
Overview 281
Configuration 281
20. Oracle Security Service Module Settings (10g) 283
Overview 283

Prerequisites 283
Settings 284
Name Authority Definition 285
Further Information 285

9. Resources ....

1. Certificates and Keys 286
Overview 286
Viewing Certificates and Private Keys 286
Configuring an X.509 Certificate 286
Configuring a Private Key 287
Global Options 287
Managing Certificates and Keystores 288
Configuring Key Pairs 288
Configuring PGP Key Pairs 289
2. API Gateway Users 291
Overview 291
Users 291
Adding Users 291
Attributes 291
Groups 292
Adding Groups 292
Updating Users or Groups 292
3. Global Schema Cache 293
Overview 293
Adding Schemas to the Cache 293
Testing WSDL Files for WSI Compliance 293
Organizing Schemas with Schema Containers 294
Schema Validation 295
4. External Connections 296
Overview 296
Authentication Repository Profiles 296
Connection Sets 296
Database Connections 297
ICAP Servers 297
JMS Services 297
Kerberos Connections 297
LDAP Connections 298
OCSP Connections 298
Proxy Servers 298
RADIUS Clients 298
SiteMinder 298
SMTP Servers 299
SOA Security Manager 299
Syslog Servers 299
TIBCO 299
Tivoli 299
URL Connection Sets 300
XKMS Connections 300
5. Global Caches 301
Overview 301
Local Caches 301
Distributed Caches 302
Distributed Cache Settings 303
Example of Caching Response Messages 304

10. Attributes ....

1. Compare Attribute 307
Overview 307

Configuration 307
2. Extract REST Request Attributes 308
Overview 308
Configuration 308
3. Extract WSS Timestamp 309
Overview 309
Configuration 309
4. Extract WSS UsernameToken 310
Overview 310
Configuration 310
5. Extract WSS Header 311
Overview 311
Configuration 311
6. Get Cookie 312
Overview 312
Configuration 312
Attribute Storage 312
7. Retrieve Attribute from Database 314
Overview 314
General Configuration 314
Database 314
Advanced 314
8. Retrieve Attributes from Directory Server 316
Overview 316
General Configuration 316
Database 316
Advanced 317
9. Retrieve Attribute from HTTP Header 318
Overview 318
Configuration 318
10. Insert SAML Attribute Assertion 319
Overview 319
General Configuration 319
Assertion Details 320
Assertion Location 320
Subject Confirmation Method 321
Advanced 323
11. Retrieve Attributes with JSON Path 325
Overview 325
Configuration 325
JSON Path Examples 325
12. Retrieve Attribute from Message 329
Overview 329
Configuration 329
13. Retrieve Attribute from SAML Attribute Assertion 330
Overview 330
Details 330
Trusted Issuers 331
Subject Configuration 331
Lookup Attributes 331
14. SAML PDP Attributes 333
Overview 333
Request Configuration 333
Response Configuration 335
15. Retrieve Attribute from User Store 336
Overview 336
General Configuration 336
Database 336

Advanced 336

11. Authentication ....

1. Attribute Authentication 337
Overview 337
Configuration 337
2. Authenticate API Key 338
Overview 338
General Settings 338
API Key Settings 338
Advanced 339
3. CA SOA Security Manager Authentication 341
Overview 341
Agent Configuration 341
Message Details Configuration 342
XmlToolkit.properties File 342
4. HTML Form-based Authentication 344
Overview 344
General Settings 344
Session Settings 344
5. HTTP Basic Authentication 346
Overview 346
Configuration 346
6. HTTP Digest Authentication 348
Overview 348
Configuration 348
7. HTTP Header Authentication 349
Overview 349
Configuration 349
8. IP Address 350
Overview 350
Configuration 350
Configuring Subnet Masks 350
9. SAML Authentication 353
Overview 353
General Settings 354
Details 354
Trusted Issuers 354
10. SAML PDP Authentication 355
Overview 355
Request Configuration 355
Response Configuration 357
11. Insert SAML Authentication Assertion 358
Overview 358
General Configuration 358
Assertion Details 359
Assertion Location 359
Subject Confirmation Method 360
Advanced 362
12. Insert Timestamp 364
Overview 364
Configuration 364
13. Insert WS-Security Username Token 365
Overview 365
General Configuration 365
Credential Details 365
Advanced 366
14. Kerberos Client Authentication 367

Overview 367
Kerberos Client 367
Kerberos Token Profile 368
15. Kerberos Service Authentication 369
Overview 369
Kerberos Service 369
Kerberos Standard 369
Message Level 370
Transport Level 370
Advanced SPNEGO 370
16. Kerberos Configuration 371
Overview 371
Kerberos Configuration File - krb5.conf 371
Advanced Settings 371
Native GSS Library 372
17. Kerberos Clients 373
Overview 373
Ticket Granting Ticket Source 373
Kerberos Principal 374
Secret Key 374
Advanced Tab 375
18. Kerberos Services 377
Overview 377
Kerberos Endpoint Tab 377
Advanced Tab 378
19. Kerberos Principals 379
Overview 379
Configuration 379
20. Kerberos Keytab 381
Overview 381
Configuration 381
21. SAML Authentication XML-Signature Verification 383
Overview 383
Configuration 383
22. XML Signature Authentication 386
Overview 386
Configuration 386
23. SSL Authentication 388
Overview 388
Configuration 388
24. Security Token Service Client 389
Overview 389
Example Request 389
General Settings 390
Request Settings 390
Issue: POP Key 390
Issue: On Behalf Of Token 392
Issue: Token Scope and Lifetime 393
Validate: Target 394
Policies Settings 394
Routing 395
Response Settings 395
Advanced Settings 395
25. WS-Security Username Authentication 398
Overview 398
General Configuration 398
Token Validation 399
Token Verification via Repository 399

12. Authorization ....
1. Attributes 401
Overview 401
Configuration 401
2. Certificate Attributes 402
Overview 402
Configuration 402
3. RSA Access Manager Authorization 404
Overview 404
General Details 404
Connection Details 404
Authorization Details 405
4. Entrust GetAccess Authorization 406
Overview 406
GetAccess WS-Trust STS 406
GetAccess SAML PDP 406
5. Insert SAML Authorization Assertion 408
Overview 408
General Configuration 408
Assertion Details 408
Assertion Location 409
Subject Confirmation Method 410
Advanced 412
6. RBAC Filter 414
Overview 414
Configuration 414
7. SAML Authorization Assertion 415
Overview 415
General Settings 415
Details 415
Trusted Issuers 416
Optional Settings 416
8. SAML PDP Authorization 417
Overview 417
Request Configuration 417
Response 419
9. Tivoli Integration 420
Overview 420
Integration Architecture 420
Prerequisites 421
Global Tivoli Configuration 423
Tivoli Authorization 425
Tivoli Authentication 426
Tivoli Attribute Retrieval 427
10. Tivoli Authorization 429
Overview 429
Adding a Tivoli Client 429
Adding Users and Web Services to Tivoli 429
Configuring Tivoli Authorization 430
Tivoli Authentication Refresh 431
11. Retrieve Attributes from Tivoli 432
Overview 432
Configuration 432
12. CA SOA Security Manager Authorization 433
Overview 433
Configuration 433
13. SAML Authorization XML-Signature Verification 434

Overview 434
Configuration 435
14. XACML Policy Enforcement Point 437
Overview 437
Example XACML Request 438
General Settings 438
XACML Settings 438
Routing Settings 441
Advanced Settings 442

13. CA SiteMinder ....

1. SiteMinder Certificate Authentication 443
Overview 443
Configuration 443
2. SiteMinder Session Validation 445
Overview 445
Configuration 445
3. SiteMinder Logout 447
Overview 447
Configuration 447
4. SiteMinder Authorization 448
Overview 448
Configuration 448
5. SiteMinder/SOA Security Manager Connection 449
Overview 449
SiteMinder and SOA Security Manager Connection Details 449
SOA Security Manager Connection Details Only 450

14. Certificates ....

1. Static CRL Certificate Validation 451
Overview 451
Configuration 452
2. Dynamic CRL Certificate Validation 453
Overview 453
Configuration 453
3. CRL LDAP Validation 454
Overview 454
Configuration 454
4. CRL Responder 455
Overview 455
Configuration 455
5. Create Thumbprint from Certificate 456
Overview 456
Configuration 456
6. Certificate Validity 457
Overview 457
Configuration 457
7. Find Certificate 458
Overview 458
Configuration 458
8. Extract Certificate Attributes 459
Overview 459
Generated Message Attributes 459
Configuration 461
9. Certificate Chain Check 462
Overview 462
Configuration 462
10. OCSP Certificate Validation 463
Overview 463

Configuration 463
11. OCSP Certificate Validation Connection 464
Overview 464
Configuration 464
12. Validate Server's Certificate Store 465
Overview 465
Configuration 465
Deployment Example 465
13. XKMS Certificate Validation 468
Overview 468
Configuration 468
14. XKMS Certificate Validation Connection 469
Overview 469
Configuration 469
15. Cache .....
1. Cache Attribute 470
Overview 470
Configuration 470
2. Create Key 471
Overview 471
Configuration 471
3. Is Cached? 472
Overview 472
Configuration 472
4. Removed Cached Attribute 473
Overview 473
Configuration 473
16. Content Filtering .....
1. ClamAV Anti-Virus 474
Overview 474
Configuration 474
2. Content Type Filtering 475
Overview 475
Allow or Deny Types 475
Configuring MIME/DIME Types 475
3. Content Validation 476
Overview 476
Manual XPath Configuration 476
XPath Wizard 477
4. HTTP Header Validation 478
Overview 478
Configuring HTTP Header Regular Expressions 478
Configuring Threatening Content Regular Expressions 479
5. ICAP Filter 481
Overview 481
Configuration 481
Example Policies 481
Further Information 482
6. McAfee Anti-Virus 483
Overview 483
Configuring a McAfee Anti-Virus Filter 483
Configuring Custom Options 484
Reporting Message Status 485
Loading McAfee Updates 485
7. Message Size 487
Overview 487
Configuration 487

8. Query String Validation 488
Overview 488
Request Query String 488
Configuring Query String Attribute Regular Expressions 488
Configuring Threatening Content Regular Expressions 490
9. Schema Validation 491
Overview 491
Schema to Use 491
Part of Message to Match 491
Advanced 492
Reporting Schema Validation Errors 494
10. JSON Schema Validation 496
Overview 496
Configuration 496
Generating a JSON Schema Using Jython 497
11. Sophos Anti-Virus 499
Overview 499
General Settings 499
Sophos Configuration Settings 499
12. Threatening Content 501
Overview 501
Scanning Details 501
MIME Types 501
13. Throttling 502
Overview 502
General Settings 502
Cache Settings 503
Using Multiple Throttling Filters 503
14. Validate Message Attributes 505
Overview 505
Configuring Message Attribute Regular Expressions 505
Threatening Content Regular Expressions 506
15. Validate REST Request 507
Overview 507
General Configuration 507
REST Request Parameter Restrictions 507
16. Validate Timestamp 510
Overview 510
Configuration 510
17. WS-SecurityPolicy Layout 512
Overview 512
Configuration 512
18. XML Complexity 513
Overview 513
Configuration 513
17. Conversion .....
1. Add HTTP Header 514
Overview 514
Configuration 514
2. JSON Add Node 515
Overview 515
Configuration 515
Examples 516
3. Add XML Node 522
Overview 522
General Configuration 522
Configure where to Insert the New Nodes 522

- Node Source ... 522
- Configure New Node Details ... 523
- Attribute Node Details ... 523
- Examples ... 523

4. Contivo Transformation ... 525
   - Overview ... 525
   - Configuration ... 525
5. Multipart Bodypart Conversion ... 526
   - Overview ... 526
   - Configuration ... 526
6. Create Cookie ... 527
   - Overview ... 527
   - Configuration ... 527
7. Create REST Request ... 528
   - Overview ... 528
   - Configuration ... 528
8. Set HTTP Verb ... 529
   - Overview ... 529
   - Configuration ... 529
9. Insert MTOM Attachment ... 530
   - Overview ... 530
   - Configuration ... 531
10. JSON to XML ... 532
    - Overview ... 532
    - Configuration ... 532
    - Examples ... 532
11. Extract MTOM Attachment ... 535
    - Overview ... 535
    - Configuration ... 536
12. Load File ... 537
    - Overview ... 537
    - Configuration ... 537
13. Remove Attachments ... 538
    - Overview ... 538
    - Configuration ... 538
14. Remove HTTP Header ... 539
    - Overview ... 539
    - Configuration ... 539
15. JSON Remove Node ... 540
    - Overview ... 540
    - Configuration ... 540
    - Examples ... 540
16. Remove XML Node ... 543
    - Overview ... 543
    - Configuration ... 543
17. Restore Message ... 544
    - Overview ... 544
    - Configuration ... 544
18. Store Message ... 545
    - Overview ... 545
    - Configuration ... 545
19. Set Message ... 546
    - Overview ... 546
    - Configuration ... 546
20. XSLT Transformation ... 547
    - Overview ... 547
    - Stylesheet Location ... 547
    - Stylesheet Parameters ... 547

Advanced 547
21. XML to JSON 549
Overview 549
Configuration 549

18. Encryption

1. Generate Key 550
Overview 550
Configuration 550
2. PGP Decrypt 551
Overview 551
Configuration 551
3. PGP Encrypt 552
Overview 552
Configuration 552
4. SMIME Decryption 554
Overview 554
Configuration 554
5. SMIME Encryption 555
Overview 555
General Configuration 555
Recipients 555
Advanced 555
6. XML-Decryption 556
Overview 556
Configuration 556
Auto-generation using the XML Decryption Wizard 556
7. XML-Decryption Settings 557
Overview 557
XML Encryption Overview 557
Node(s) to Decrypt 559
Decryption Key 560
Options 560
Auto-generation using the XML Decryption Wizard 561
8. XML-Encryption 562
Overview 562
Configuration 562
Auto-generation using the XML Encryption Settings Wizard 562
9. XML-Encryption Settings 563
Overview 563
XML Encryption Overview 563
Encryption Key 565
Key Info 566
Recipients 569
What to Encrypt 571
Advanced 571
Auto-generation using the XML Encryption Settings Wizard 572
10. XML Encryption Wizard 573
Overview 573
Configuration 573

19. Integrity

1. XML Signature Generation 574
Overview 574
Signing Key 574
What to Sign 579
Where to Place Signature 584
Advanced 585
Additional 585

Algorithm Suite 587
Options 588
2. XML Signature Verification 591
Overview 591
Signature Verification 591
What Must Be Signed 591
Advanced 592
3. PGP Sign 593
Overview 593
Configuration 593
4. PGP Verify 595
Overview 595
Configuration 595
5. SMIME Sign 597
Overview 597
Configuration 597
6. SMIME Verify 598
Overview 598
Configuration 598
20. Fault Handlers .....
1. Generic Error 599
Overview 599
General Configuration 599
Generic Error Contents 599
2. JSON Error 601
Overview 601
General Configuration 601
JSON Error Contents 601
Customized JSON Errors 603
3. SOAP Fault 604
Overview 604
SOAP Fault Format 604
SOAP Fault Contents 604
Customized SOAP Faults 605
21. Monitoring .....
1. System Alerting 608
Overview 608
Configuring an Alert Destination 608
Configuring an Alert Filter 612
2. Audit Log Settings 615
Overview 615
Configuring Log Output 615
Log to Text File 615
Log to XML File 616
Log to Database 616
Log to Local Syslog 617
Log to Remote Syslog 617
Log to System Console 617
3. Access Log Settings 618
Overview 618
Log Format 618
Configuring the Access Log 618
4. Log Level and Message 620
Overview 620
Configuration 620
5. Log Message Payload 622
Overview 622

Configuration ... 622
6. Log Access Filter ... 623
Overview ... 623
Log Format ... 623
Configuration ... 624
7. Service Level Agreement (SLA) Filter ... 625
Overview ... 625
Response Time Requirements ... 625
HTTP Status Requirements ... 626
Communications Failure Requirements ... 627
Select Alerting System ... 628
8. Set Service Context ... 629
Overview ... 629
Configuration ... 629
22. OAuth ...
1. API Gateway OAuth 2.0 Introduction ... 630
Overview ... 630
OAuth 2.0 Definitions ... 630
OAuth 2.0 Authentication Flows ... 630
Further Information ... 631
2. Configuring and Managing OAuth 2.0 ... 632
Overview ... 632
Enabling OAuth 2.0 Management ... 632
Pre-registered Client Applications ... 633
Managing Registered Clients ... 633
Sample Clients ... 635
Token Management ... 635
API Manager REST API ... 639
Database-Backed API Manager ... 641
OAuth Database Schemas ... 642
OpenSSL Commands ... 643
OAuth 2.0 Message Attributes ... 643
3. API Gateway OAuth 2.0 Authentication Flows ... 649
Overview ... 649
Authorization Code (or Web Server) Flow ... 649
Implicit Grant (or User Agent) Flow ... 654
Resource Owner Password Credentials Flow ... 658
Client Credentials Grant Flow ... 660
OAuth 2.0 JWT Flow ... 662
Revoke Token ... 664
Token Info Service ... 666
4. OAuth Access Token Information ... 669
Overview ... 669
Access Token Info Settings ... 669
Monitoring ... 669
5. Access Token using Authorization Code ... 671
Overview ... 671
Application Validation ... 671
Access Token ... 672
Monitoring ... 672
6. Access Token using Client Credentials ... 673
Overview ... 673
Application Validation ... 673
Access Token ... 673
Monitoring ... 674
7. Access Token using JWT ... 675
Overview ... 675

Application Validation 675
Access Token 675
Monitoring 676
8. Authorization Code Flow 677
Overview 677
Validation/Templates 677
Authz Code Details 678
Access Token Details 678
Monitoring 679
9. Authorize Transaction 680
Overview 680
Validation/Templates 680
Authz Code Details 680
Access Token Details 681
Monitoring 682
10. Refresh Access Token 683
Overview 683
Application Validation 683
Access Token 683
Monitoring 684
11. Resource Owner Credentials 685
Overview 685
Application Validation 685
Access Token 686
Monitoring 686
12. Revoke a Token 687
Overview 687
Revoke Token Settings 687
Monitoring 687
13. Validate Access Token 688
Overview 688
Configuration 688
23. Oracle Access Manager ....
1. Oracle Access Manager Authorization 689
Overview 689
Configuration 689
2. Oracle Access Manager Log in with Certificate 690
Overview 690
General Configuration 690
Resource Configuration 690
Session Configuration 690
OAM Access Server SDK Configuration 691
3. Logout from Oracle Access Manager SSO Session 692
Overview 692
Configuration 692
4. Oracle Access Manager SSO Token Validation 693
Overview 693
Configuration 693
24. Oracle Entitlements Server ....
1. Oracle Entitlements Server 10g Authorization 694
Overview 694
General 694
Settings 694
Application Context 695
2. Get Roles from Oracle Entitlements Server 10g 696
Overview 696
General 696

Settings ... 696
Application Context ... 696
3. Oracle Entitlements Server 11g Authorization ... 697
Overview ... 697
Configuration ... 697
25. Resolvers ...
1. Relative Path Resolver ... 698
Overview ... 698
Configuration ... 698
2. SOAP Action Resolver ... 699
Overview ... 699
Configuration ... 699
3. Operation Name ... 700
Overview ... 700
Configuration ... 700
26. Routing ...
1. Getting Started with Routing Configuration ... 701
Overview ... 701
Proxy or Endpoint Server ... 701
Service Virtualization ... 701
Choosing the Correct Routing Filters ... 701
Case 1: Proxy without Service Virtualization ... 702
Case 2: Proxy with Service Virtualization ... 703
Case 3: Endpoint without Service Virtualization ... 704
Case 4: Endpoint with Service Virtualization ... 705
Case 5: Simple Redirect ... 706
Case 6: Routing on to an HTTP Proxy ... 707
Summary ... 708
2. Routing Wizard ... 710
Overview ... 710
Configuration ... 710
3. Call Internal Service ... 712
Overview ... 712
Configuration ... 712
4. Connection ... 713
Overview ... 713
General Configuration ... 713
Trusted Certificates ... 713
Client SSL Authentication ... 713
HTTP Authentication ... 713
Kerberos Authentication ... 713
Behavior ... 714
Advanced ... 715
5. Connect to URL ... 717
Overview ... 717
General Configuration ... 717
Trusted Certificates ... 717
Client SSL Authentication ... 717
HTTP Authentication ... 717
Kerberos Authentication ... 718
Behavior ... 718
Advanced ... 720
Request Details ... 721
6. Dynamic Router ... 722
Overview ... 722
Configuration ... 722
7. Extract Path Parameters ... 723

Overview 723
Configuration 723
Required Input and Generated Output 724
Possible Outcomes 724
8. File Download 725
Overview 725
General Settings 725
File Details 725
Connection Type 725
FTP and FTPS Connections 726
FTPS Connections 726
SFTP Connections 726
9. File Upload 727
Overview 727
General Settings 727
File Details 727
Connection Type 728
FTP and FTPS Connections 728
FTPS Connections 728
SFTP Connections 729
10. HTTP Redirect 730
Overview 730
Configuration 730
11. HTTP Status Code 731
Overview 731
Configuration 731
12. Insert WS-Addressing 732
Overview 732
Configuration 732
13. Messaging System Filter 733
Overview 733
Request Settings 733
Response Settings 735
14. Read WS-Addressing 737
Overview 737
Configuration 737
15. Rewrite URL 738
Overview 738
Configuration 738
16. Save to File 739
Overview 739
Configuration 739
17. SMTP Routing 740
Overview 740
General Settings 740
Message Settings 740
18. Static Router 741
Overview 741
Configuration 741
19. TIBCO Rendezvous Routing 742
Overview 742
Configuration 742
20. TIBCO Enterprise Messaging Service Routing Filter 743
Overview 743
Connection 743
Request 743
Response 744
21. TIBCO Enterprise Messaging Service Connection 746

Overview 746
Configuration 746
22. Wait for Response Packets 748
Overview 748
Packet Sniffer Configuration 748
Sniffing Response Packets 749
23. Proxy Servers 750
Overview 750
Configuration 750
27. Security Services ....
1. DSS Signature Generation Service 751
Overview 751
Configuration 751
2. DSS Signature Verification 752
Overview 752
Configuration 752
3. Encrypt and Decrypt Web Services 753
Overview 753
Configuration 753
4. STS Web Service 754
Overview 754
Configuration 754
28. WS-Trust ....
1. Consume WS-Trust Message 755
Overview 755
Consume WS-Trust Message Types 755
Message Consumption 755
Advanced 756
2. Create WS-Trust Message 757
Overview 757
Create WS-Trust Message Type 757
Message Creation 757
RST Creation 758
RSTR Creation 758
Advanced Settings 758
29. Extensibility ....
1. Advanced Filter View 760
Overview 760
Configuration 760
2. Selecting Configuration Values at Runtime 761
Overview 761
Selector Syntax 761
Example Selector Expressions 762
Extracting Message Attributes 763
3. Key Property Stores 764
Overview 764
KPS Backing Data Stores 764
Configuring a Key Property Store 764
4. Scripting Language Filter 766
Overview 766
Writing a Script 766
Configuring a Script Filter 767
Adding a Script to the Library 767
5. Writing a Custom Filter using the Oracle API Gateway SDK 769
Overview 769
Policies, Filters, and Message Attributes 769
Oracle API Gateway SDK Overview 770

Tutorial Prerequisites 770
Oracle API Gateway SDK Sample Overview 770
Step 1: Create the Typedocs 771
Step 2: Create the Filter Class 774
Step 3: Create Processor Class 776
Step 4: Create Policy Studio Classes 779
Step 5: Build Classes 785
Step 6: Load TypeDocs 786
Step 7: Construct a Policy 791
Step 8: Configure the SimpleFilter 793
Conclusion 797
30. Utility ....
1. Abort Filter 798
Overview 798
Configuration 798
2. Check Group Membership 799
Overview 799
Configuration 799
Possible Paths 799
3. Configuration Web Service 800
Overview 800
4. Copy/Modify Attributes 801
Overview 801
Configuration 801
5. Evaluate Expression 802
Overview 802
Configuration 802
6. Execute External Process 803
Overview 803
Configuration 803
7. False Filter 804
Overview 804
Configuration 804
8. HTTP Parser 805
Overview 805
Configuration 805
9. Insert BST 806
Overview 806
Configuration 806
10. Invoke Policy per Message Body 807
Overview 807
Configuration 807
11. Locate XML Nodes 808
Overview 808
Configuration 808
12. Pause Filter 810
Overview 810
Configuration 810
13. Policy Shortcut 811
Overview 811
Configuration 811
14. Policy Shortcut Chain 812
Overview 812
General Configuration 812
Add a Policy Shortcut 812
Edit a Policy Shortcut 813
15. Quote of the Day 814

Overview 814
Configuration 814
16. Reflect Message Filter 815
Overview 815
Configuration 815
17. Reflect Message And Attributes Filter 816
Overview 816
Configuration 816
18. Remove Attribute 817
Overview 817
Configuration 817
19. Set Response Status 818
Overview 818
Configuration 818
20. Set Attribute 819
Overview 819
Configuration 819
21. String Replace Filter 820
Overview 820
Configuration 820
22. Switch on Attribute Value 821
Overview 821
Configuration 821
Adding a Switch Case 821
23. Time Filter 823
Overview 823
General Configuration 823
Basic Time Options 823
Advanced Time Options 824
24. Trace Filter 825
Overview 825
Configuration 825
25. True Filter 826
Overview 826
Configuration 826
31. Web Services ....
1. Web Service Filter 827
Overview 827
General Settings 827
Routing 827
Validation 827
Message Interception Points 828
WSDL 830
Monitoring 831
2. Return WSDL 832
Overview 832
Configuration 832
3. Set Web Service Context 833
Overview 833
Configuration 833
32. Common Configuration ....
1. Authentication Repository 834
Overview 834
Local Repositories 834
LDAP Repositories 834
CA SiteMinder Repositories 837
Database Repositories 838

Entrust GetAccess Repositories 840
Oracle Access Manager Repositories 840
Oracle Entitlements Server 10g Repositories 842
RADIUS Repositories 842
RSA Access Manager Repositories 843
Tivoli Repositories 843
2. Certificate Chain Check 844
Overview 844
Configuration 844
3. Certificate Validation 845
Overview 845
Configuration 845
Configuring URL Groups 846
4. Compressed Content Encoding 848
Overview 848
Encoding of HTTP Responses 848
Encoding of HTTP Requests 848
Delimiting the End of an HTTP Message 848
Configuring Content Encoding 849
Further Information 850
5. Configuring Connection Groups 851
Overview 851
Configuring a Connection Group 851
Configuring a Connection 851
6. Configuring Cron Expressions 852
Overview 852
Creating a Cron Expression using the Time Tabs 852
Entering a Cron Expression 855
Testing the Cron Expression 855
Further Information 855
7. Database Connection 856
Overview 856
Configuring the Database Connection 856
Database Connection Pool Settings 857
Connection Validation 857
Test the Connection 857
8. Database Query 858
Overview 858
Configuration 858
9. Configuring ICAP Servers 860
Overview 860
General Settings 860
Server Settings 860
Security Settings 860
Advanced Settings 861
Further Information 861
10. Configuring LDAP Directories 862
Overview 862
General Configuration 862
Authentication Configuration 862
Testing the LDAP Connection 863
Additional JNDI Properties 864
11. RADIUS Clients 865
Overview 865
Configuration 865
12. SAML PDP Response XML-Signature Verification 866
Overview 866
Configuration 866

13. Signature Location 869
Overview 869
Configuration 869
14. SMTP Servers 872
Overview 872
Configuration 872
15. Configuring a Transparent Proxy 873
Overview 873
Configuring Transparent Proxy Mode for Incoming Interfaces 873
Configuring Transparent Proxy Mode for Outgoing Calls 873
Configuration Example 873
16. Retrieving WSDL Files from a UDDI Registry 876
Overview 876
UDDI: A Brief Introduction 876
UDDI Definitions 876
Configuring a Registry Connection 878
WSDL Search 878
Quick Search 879
Name Search 880
Advanced Search 880
Advanced Options 882
Publish 884
17. Connecting to a UDDI Registry 885
Overview 885
Configuring a Registry Connection 885
Securing a Connection to a UDDI Registry 886
18. Publishing WSDL Files to a UDDI Registry 888
Overview 888
Finding WSDL Files 888
Publishing WSDL Files 888
Step 1: Enter Virtualized Service Address and WSDL URL for Publishing in UDDI Registry 888
Step 2: View WSDL to UDDI Mapping Result 889
Step 3: Select a Registry for Publishing 890
Step 4: Select a Duplicate Publishing Approach 890
Step 5: Create or Search for Business 891
Step 6: Publish WSDL 892
19. LDAP User Search 893
Configure Directory Search 893
20. Configuring URL Groups 894
Overview 894
Configuration 894
21. What To Sign 895
Overview 895
ID Configuration 895
Node Locations 897
XPath Configuration 897
XPath Predicates 898
Message Attribute 898
22. Configuring XPath Expressions 899
Overview 899
Manual Configuration 899
XPath Wizard 900
33. Reference ....
Message Attribute Reference 902
Message Filter Reference 936
WS-Policy Reference 972
Glossary of Terms 975

Oracle API Gateway Overview

Overview

Oracle API Gateway is a comprehensive platform for managing, delivering, and securing Web APIs. It provides integration, acceleration, governance, and security for API and SOA-based systems. Oracle API Gateway is available on Windows, Linux, and Solaris (for more details, see the *Oracle API Gateway Installation and Configuration Guide*). This topic describes the high-level functionality available in the Oracle API Gateway.

Integration

Oracle API Gateway provides the following integration features:

Identity Management
Oracle API Gateway integrates with existing third-party Identity Management (IM) infrastructures to perform authentication and authorization of message traffic. For example, integration is provided with LDAP, Microsoft Active Directory, Oracle Access Manager, CA SiteMinder, Entrust GetAccess, IBM Tivoli Access Manager, RSA Access Manager, and other IM products. The API Gateway also interoperates with leading integration products and platforms (for example, Microsoft .NET, Oracle WebLogic, IBM WebSphere, and SAP NetWeaver).

Scalability
The API Gateway is designed to offer a highly flexible and scalable solution architecture. Administrators can deploy new API Gateway instances as needed, and deploy the same or different policies across a group of API Gateway instances as required. This enables administrators to apply polices at any point in their system. Policy enforcement points can be distributed around the network, anywhere traffic is being passed.

Pluggable Pipeline
The API Gateway's internal message-handling pipeline is extensible, enabling extra access control and content-filtering rules to be added with ease. Customers do not have to wait for a full product release before receiving updates of support for emerging standards and for additional adapters.

REST APIs
The API Gateway's REST support enables you to make enterprise application data and operations available using Web APIs. For example, you can convert a legacy SOAP service, and deploy it as a REST API to be consumed by mobile apps. REST-to-SOAP conversion is easy to achieve using the API Gateway. It can expose REST APIs that map to SOAP services, dynamically creating a SOAP request based on the REST API call.

Internationalization
The API Gateway includes support for multi-byte message data and a wide range of international languages and character sets. For example, this includes requests in languages such as Chinese, German, French, Spanish, Danish, Serbian, Russian, Japanese, Korean, Greek, Arabic, Hebrew, and so on. The API Gateway supports character sets such as `UTF-8`, `KO-I8`, `UTF-16`, `UTF-32`, `ISO-8859-1`, `EUC-JP`, `US-ASCII`, `ISO-8859-7`, and so on.

Performance

Oracle API Gateway accelerates performance as follows:

Processing Offload
You can use the API Gateway to offload the heavy lifting of XML from application servers, and on to the network. This frees up resources on application servers and enables applications to run faster. The patented high-performance core XML Acceleration engine (VXA), coupled with hardware acceleration ensures wirespeed network performance.

VXA Platform
The core VXA engine is integrated into the API Gateway to accelerate the essential XML security primitives. This engine provides XML processing at faster levels than those performed by common JAXP implementations in application servers and other applications that sit downstream from the API Gateway. The VXA engine performs Document Object Model

(DOM) processing, XPath, JSON Path, XSLT conversion, and validation of XML and JSON.

Data Enrichment
The API Gateway can automatically populate content in XML and JSON documents from sources such as databases. By putting this functionality on to the network infrastructure, data is automatically populated in messages before they reach the consuming services. This simplifies and accelerates applications in ESBs and application servers.

Governance

Oracle API Gateway provides the following governance features:

Ease of Deployment
The API Gateway includes many features that speed up deployment. For example, certificates and private keys, necessary for XML security functions, are issued on board. The API Gateway has a *deny-by-default* defense posture, to detect and block unauthorized deployments of services. Policies can be re-applied across multiple endpoints using simple menus. Policies can also be imported and exported as XML files. This minimizes time needed to replicate policies across multiple API Gateways, or to move from a staging system to production environment.

Centralized Management
A web-based system management dashboard provides centralized control of API Gateways in your domain. API Gateway Manager provides quick and easy access to enable you to manage your API Gateways and services. For example, you can use monitoring and a traffic log to monitor messages sent through API Gateways in your domain. All monitoring data can be aggregated across multiple API Gateway instances in a group or domain.

The Oracle Policy Studio tool enables administrators to add security and management policies to the API Gateway, and to manage policy versions across multiple API Gateways. This enables enterprise policy management to be brought under centralized control, rather than be managed separately on each API Gateway.

Reporting
The Oracle API Gateway Analytics tool provides auditing and reporting on usage across all entry points and creates comprehensive reports to meet operational and compliance requirements. Oracle API Gateway Analytics also provides root cause analysis by identifying common failure points in multi-service transactions. If a service fails, and impacts the transaction as a whole, API Gateway Analytics can detect this and generate alerts.

Traffic Throttling
The Oracle API Gateway protects services from unanticipated traffic spikes by smoothing out traffic. It also limits clients to agreed service consumption levels in accordance with service usage agreements. This enables Oracle customers to charge their clients for different levels of service usage.

Security

Oracle API Gateway includes the following security features:

Identity Mediation
Through its support for a wide range of security standards, Oracle API Gateway enables identity mediation between different identity schemes. For example, the API Gateway can authenticate external clients by username and password, but then issue SAML tokens that are used for identity propagation to application servers.

API Management
The API Gateway enables you to secure Web APIs against attack and abuse. It also enables you to govern and meter access to and usage of Web APIs. The API Gateway provides support for API management security standards such as OAuth. This enables you to share private resources with third-party websites without needing to provide credentials.

Application-level Networking
The API Gateway routes data based on sender identity, content, and type. This enables messages to be sent to the appropriate application in a secure manner. It also enables *service virtualization*, where services are exposed to clients with virtual addresses to mask their actual addresses for security and application delivery. In this way, the API Gateway acts as an important control point for network traffic by shielding endpoint services from direct access.

Audit Trail

The API Gateway satisfies audit requirements by enabling service transactions to be archived in a tamper-proof store for subsequent audit. Oracle also facilitates privacy compliance support by allowing sensitive information, such as customer names, to be encrypted or stripped out of message traffic.

Oracle API Gateway Architecture

Overview

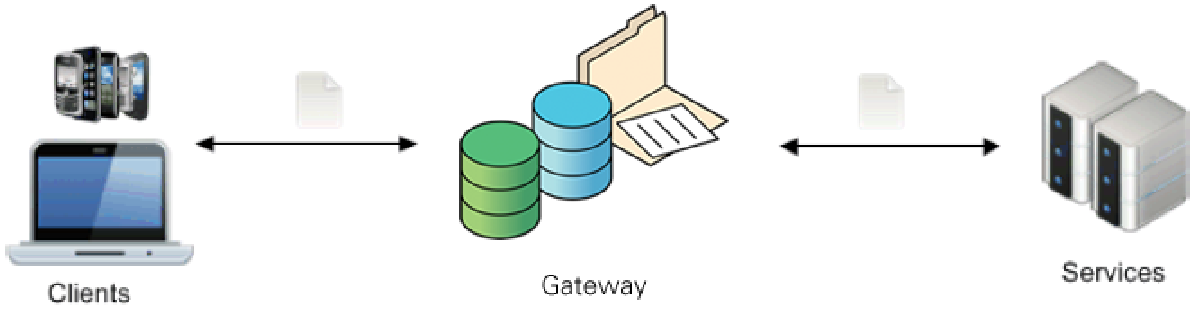
This topic provides a high-level overview of the basic Oracle API Gateway product architecture, and describes its main components and user roles. It also describes the highly scalable and reliable group-based architecture, which enables you to manage API Gateways across your organization. For an introduction to product features and benefits, see the *Oracle API Gateway Overview* topic.

Note

Oracle API Gateway is available on Windows, Linux, and Solaris (for more details, see the *Oracle API Gateway Installation and Configuration Guide*).

Basic Architecture

This section provides a simple high-level overview of the Oracle API Gateway architecture. The following diagram shows the main components:

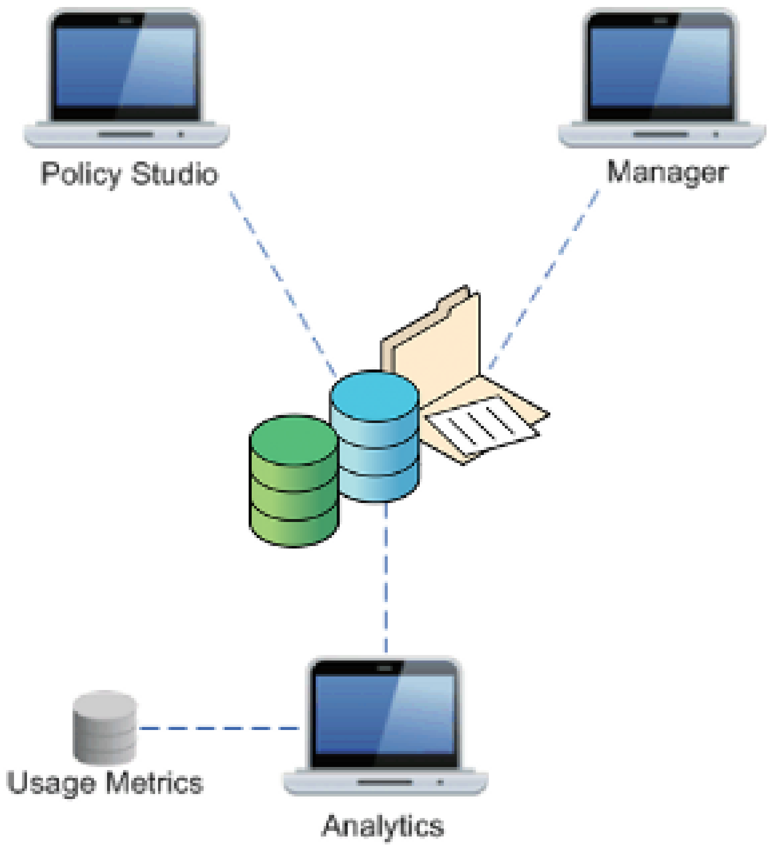


This diagram is a simplified view that includes clients, a single API Gateway, and enterprise services. However, you can deploy multiple API Gateways to suit the needs of your distributed environment.

API Gateway
The API Gateway integrates, accelerates, governs, and secures Web API and SOA-based systems. For example, it can perform application networking by routing traffic based on content and sender, and by performing message content screening. The API Gateway applies policies to incoming messages by running message filters on requests. The API Gateway supports a wide range of message transports, protocols, and formats (for example, XML, JSON, SOAP, REST, HTTP, JMS, TIBCO, FTP, SMTP, POP, and so on). For more details on API Gateway features, see *Oracle API Gateway Overview*.

In a typical deployment scenario, Oracle API Gateway components are deployed in the demilitarized zone (DMZ). The connection between the client and the API Gateway is protected by a perimeter firewall, and the connection between the API Gateway and the back-end service by a Network Address Translation (NAT) firewall.

Configuration and Management Tools
The following diagram shows a simplified view of the tools used to configure and manage the API Gateway:



These tools are described in the context of the main API Gateway user roles in the sections that follow.

Policy Development

A Oracle API Gateway policy developer typically performs the following tasks:

- Develops API Gateway policies and solution packs.
- Customizes and extends the API Gateway using scripting.
- Creates Java classes and/or custom filters using the API Gateway filter SDK.
- Uses the Policy Studio, API Gateway Explorer, and API Gateway Manager tools.

Policy Studio
Policy Studio is a policy development and configuration tool that enables policy developers to easily configure API Gateway policies and settings to control and protect deployed API services and Web services. For example, Policy Studio enables you to create and assign policies, configure the full range of API Gateway configuration settings, and manage API Gateway deployments. Policy Studio is typically installed on a separate machine from the API Gateway to enable remote administration.

API Gateway Explorer
API Gateway Explorer is an API service and Web service test client used by policy developers to generate test messages, which are sent to the API Gateway and back to API Gateway Explorer. API Gateway Explorer supports both REST-based and SOAP-based invocations, and is available as a separately installed tool.

API Gateway Administration

The API Gateway administrator typically performs the following tasks:

- Manages, monitors, and troubleshoots the API Gateway.
- Configures and manages the domain, group, and API Gateway hierarchy.
- Uses the API Gateway Manager tool.

For example, if a client presents an invalid SSL certificate, the API Gateway administrator needs to be alerted and works with the client to address the issue. The API Gateway administrator also debugs transactions on a day-to-day basis. For example, if a partner reports that his file is blocked, the API Gateway administrator uses traffic monitoring to find the transaction and figure out what went wrong.

API Gateway Manager
API Gateway Manager is a centralized web-based dashboad that enables administrators to control and manage API Gateways and groups in a domain. API Gateway Manager connects to the Node Manager on each host, and displays aggregated monitoring data from multiple API Gateway instances. For example, this includes real-time statistics, traffic log, log files, and alerts. You can view all monitoring data at the level of a domain, group, and API Gateway, depending on which of these components are selected. For more details, see *Starting the API Gateway Tools*.

API Service Administration

The API service administrator typically performs the following tasks:

- Manages, monitors, and troubleshoots the API Services that are virtualized on the API Gateway.
- Has expertise of the services and APIs (what they do, and why they are used).
- Does not manage and troubleshoot the API Gateway, and does not have expertise of API Gateway operation.
- Uses the API Service Manager, traffic monitoring, and real-time monitoring components in API Gateway Manager tool, and the Oracle API Gateway Analytics tool.

For example, the API service administrator is interested in who is using an API, its usage by time of day, how its usage compares to other APIs (is it going up or down over time, are clients reaching their quotas, is it meeting its Service Level Agreement (SLA), and so on).

Oracle API Gateway Analytics
Oracle API Gateway Analytics is a separately installed tool used by administrators to generate reports and charts based on usage metrics for all services and API Gateways in a domain. API Gateway Analytics provides integration with databases such as MySQL Server, MS SQL Server, and Oracle. API Gateway Analytics includes both real-time and historical metrics. For example, the API service administrator can generate and store reports that monitor which authenticated clients are calling which API services over time.

System Administration

The system administrator typically performs the following tasks:

- Installs and monitors the API Gateway in a system and/or Virtual Machine (VM) production environment.
- Has expertise in the system/VM environment and the tools that the API Gateway runs in.
- Does not use API Gateway management tools.

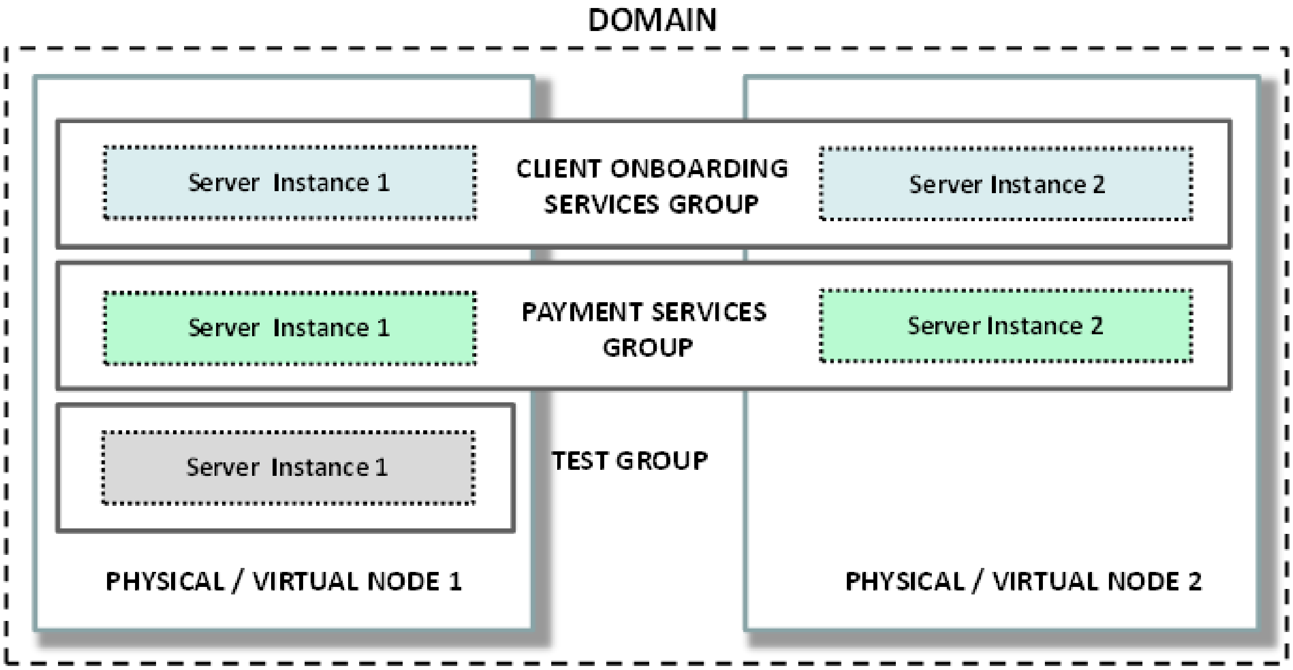
Managed Domain Architecture

This section drills down to describe the API Gateway product architecture in more detail. It describes the API Gateway's group-based domain architecture, which enables you to break down your projects into logical groups and manage configuration across your organization. This provides manageability and scalability, and enables you to perform load balancing and failover across distributed deployments. This group-based architecture includes the following main components:

Domain
A domain is the set of all hosts running API Gateway instances, which are managed centrally by the API Gateway Manager tool. A host is defined as a physical machine. A domain administration password is used to secure the domain's Certificate Authority private key.

Group
A group is a number of API Gateway instances (one or many) that all run the same configuration. API Gateways *always* run in a group, and each API Gateway can only be a member of one group. You can deploy, manage, and monitor a group of API Gateways using the Policy Studio and the browser-based API Gateway Manager.

A group normally runs across more than one physical host machine (see the following diagram). However, a group can also include more than one API Gateway instance on the same host. Each API Gateway in the group runs the same configuration. Each API Gateway has its own deployment descriptor file (`envSettings.properties`). This enables the API Gateways in the group to use different host-specific settings as required (port bindings, certificates, and so on). A group also has a deployment descriptor, which specifies settings values that are the same across the group but may differ in different environments. A *standalone* API Gateway runs in a group of one member (**TEST GROUP** in the diagram).



For example, you can use dedicated groups and API Gateway instances for high value services and to avoid potential down time due to lower value services. This makes API Gateway upgrades and maintenance much easier. You can also create dedicated API Gateway instances for specific API Gateway users or groups. When you create an API Gateway instance, the newly created configuration files are stored separately from the API Gateway executables, which makes them easier to maintain.

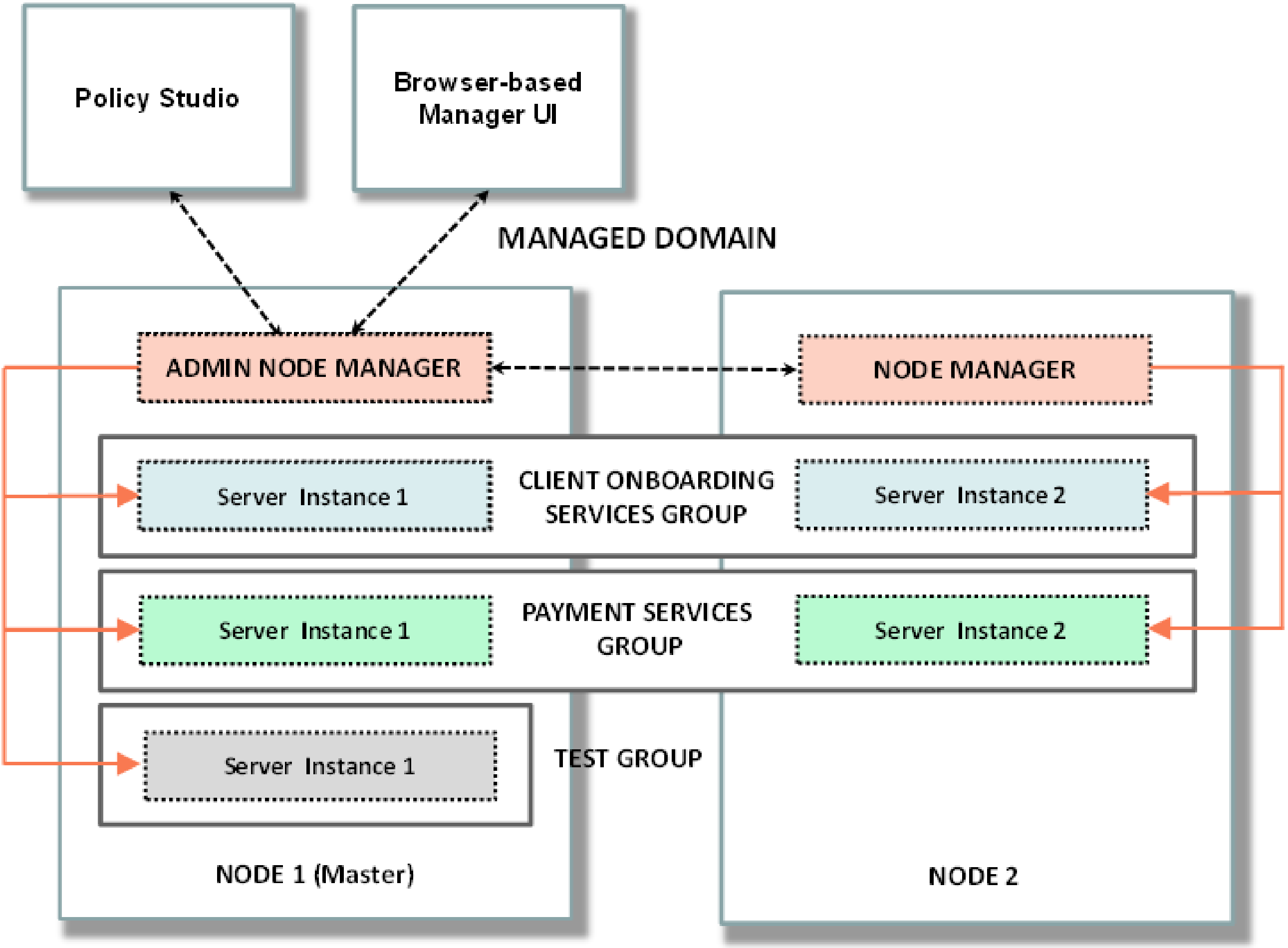
Note

You cannot deploy configuration to a single API Gateway instance, the deployment must occur at group level to ensure that all API Gateways in the group are running the same configuration. API Gateway names are unique in a group, and group names are unique in the domain.

Node Manager

The Node Manager is a server-side process that runs on each host in the domain, which manages and monitors the API Gateway instances on that host. Only one Node Manager runs per host. Communication between the Node Manager and the API Gateway is secured using SSL by default. The Policy Studio and the browser-based API Gateway Manager are clients of the Node Manager.

The first Node Manager added in a domain is known as the *Admin Node Manager*. This is the Node Manager that the user connects to using the API Gateway Manager, Policy Studio, or API Gateway Explorer clients. The Admin Node Manager acts as the *master* Node Manager. It performs Role-Based Access Control (RBAC), and forwards requests to other Node Managers when required. The Admin Node Manager also manages and deploys configuration to the API Gateway instance(s) in a domain.



In addition, the Node Manager also supports process management (for example, starting and stopping API Gateway instances) using system tools such as `initd` on UNIX and Service Control Manager on Windows.

Note

The Node Manager is not critical to the running of API Gateway instances, or to the business services protected by the API Gateway. If the Node Manager is not running, the API Gateway instances can still process requests for business services. However, the Node Manager must be running to manage and monitor the API Gateway, or to make updates to the API Gateway configuration. In addition, the Node Manager is required to create API Gateway instances.

Tags

API Gateway instances can have associated tags. A tag is a form of metadata consisting of a case-sensitive key-value pair. This enables you to create user-friendly names that help to organize, search, and browse API Gateway instances using API Gateway Manager and Policy Studio. You can add tags when the API Gateway is created or updated. For example, you could define a tag with `key="QAStatus"` and `value=“Passed iteration one of performance testing”`.

Groups do not have tags. However, you can apply a tag at the group level, which simply creates the same tag for each API Gateway in the Group. You cannot aggregate data based on tags. Tags are purely an internal client-side (API Gateway Manager and Policy Studio) tool, which can be used to search for API Gateways that match a specific tag value. In the API Gateway Manager, you can use tags to search for API Gateways in a table view of the entire domain. Tags are stored on disk in a configuration file specific to each API Gateway.

Further Information

For more details on API Gateway architecture and concepts, see *Oracle API Gateway Concepts*.

Oracle API Gateway Concepts

Overview

This topic explains the main concepts in the Oracle API Gateway architecture and shows examples of how they are displayed in the API Gateway management tools (Policy Studio, API Service Manager, and Policy Studio). For example, these include concepts such as filters, policies, message attributes, and listeners.

Product Concepts

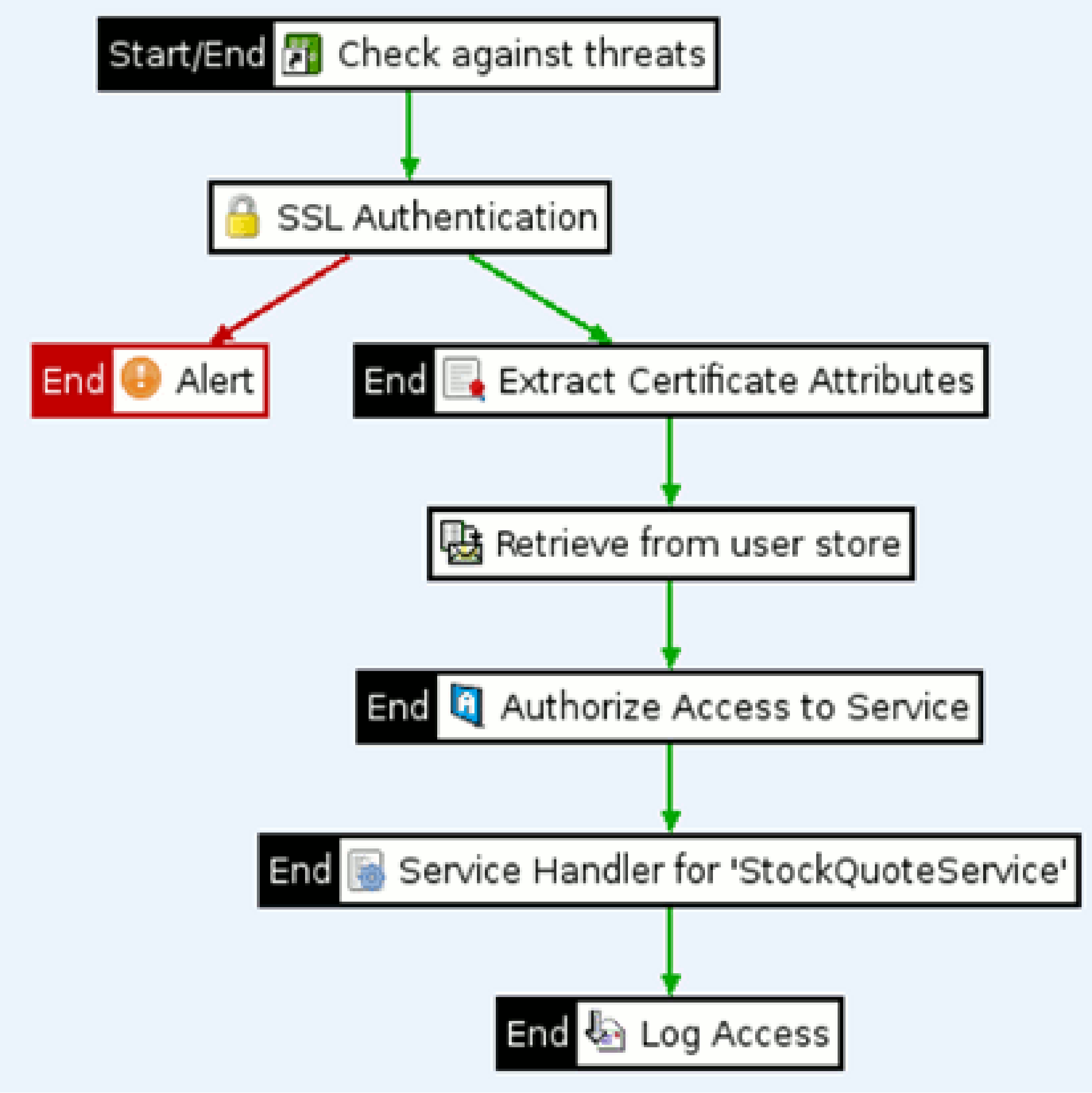
The main concepts in the Oracle API Gateway product architecture that are represented in the API Gateway management tools are as follows:

Filter
A filter is an executable rule that performs a specific type of processing on a message. For example, the **Message Size** filter rejects messages that are greater or less than a specified size. There are many categories of message filters available with the API Gateway, including authentication, authorization, content filtering, signing, and conversion. In the Policy Studio, a filter is displayed as a block of business logic that forms part of an execution flow known as a policy.

Policy
A policy is a network of message filters in which each filter is a modular unit that processes a message. A message can traverse different paths through the policy, depending on which filters succeed or fail. For example, this enables you to configure policies that route messages that pass a **Schema Validation** filter to a back-end system, and route messages that pass a different **Schema Validation** filter to a different system. A policy can also contain other policies, which enables you to build modular reusable policies.

In the Policy Studio, the policy is displayed as a path through a set of filters. Each filter can have only one **Success Path** and one **Failure Path**. You can use these success and failure paths to create sophisticated rules. For example, if the incoming data matches schema A, scan for attachments and route to service A, otherwise route to service B. You can configure the colors used to display success paths and failure paths in the Policy Studio **Preferences** menu. You can also specify to **Show Link Labels** (**S** or **F**).

The following example screen shot shows an example policy with success paths and a single failure path:

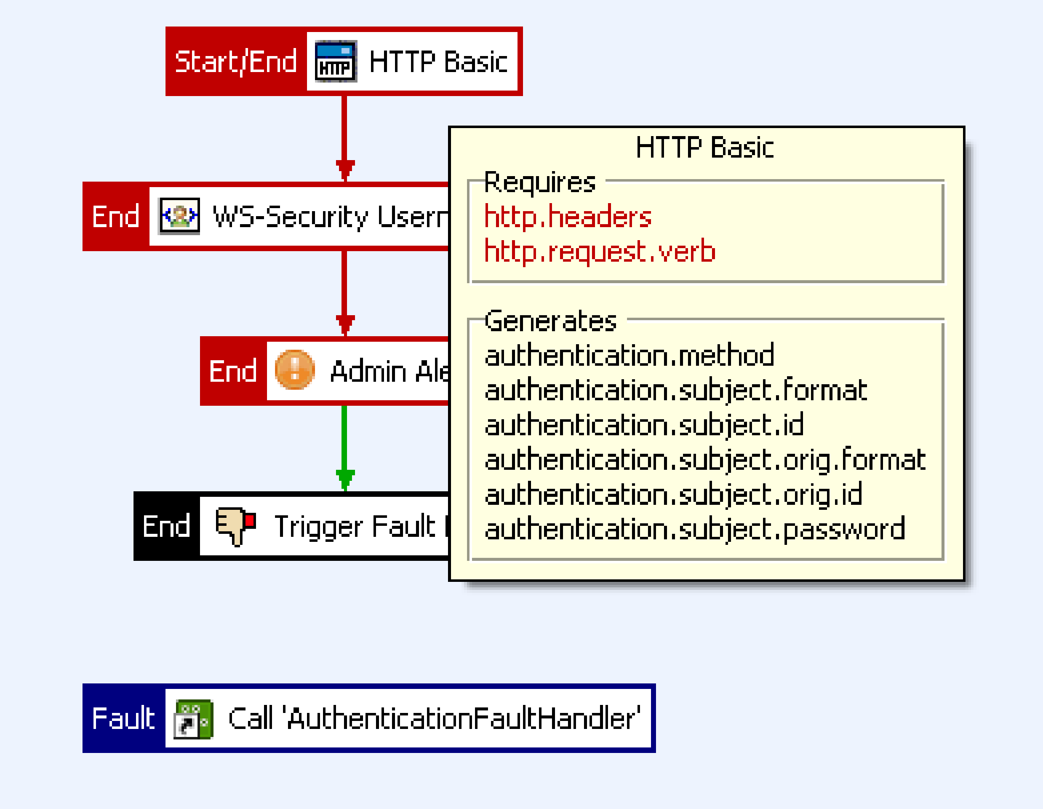


A policy must have a **Start** filter (in this example, **Check against threats**). Filters labeled **End** stop the execution of the policy if the filter execution fails. A filter labeled **Start/End** indicates that the policy execution starts there, and that the policy stops executing if this filter fails. A policy with a single filter labeled **Start/End** is also valid.

Message Attributes

Each filter requires input data and produces output data. This data is stored in message attributes, and you can access their values in API Gateway configuration using a selector syntax (for example, `${attribute.name}`). You can also use specific filters to create your own message attributes, and to set their values. The full list of message attributes flowing through a policy is displayed when you right-click the Policy Studio canvas, and select **Show All Attributes**. You can also hover your mouse over a filter to see its inputs and outputs. The **Trace** filter enables you to trace message attribute values at execution time.

The following example screen shot shows the attributes displayed when hovering over an **HTTP Basic** authentication filter:



HTTP Basic

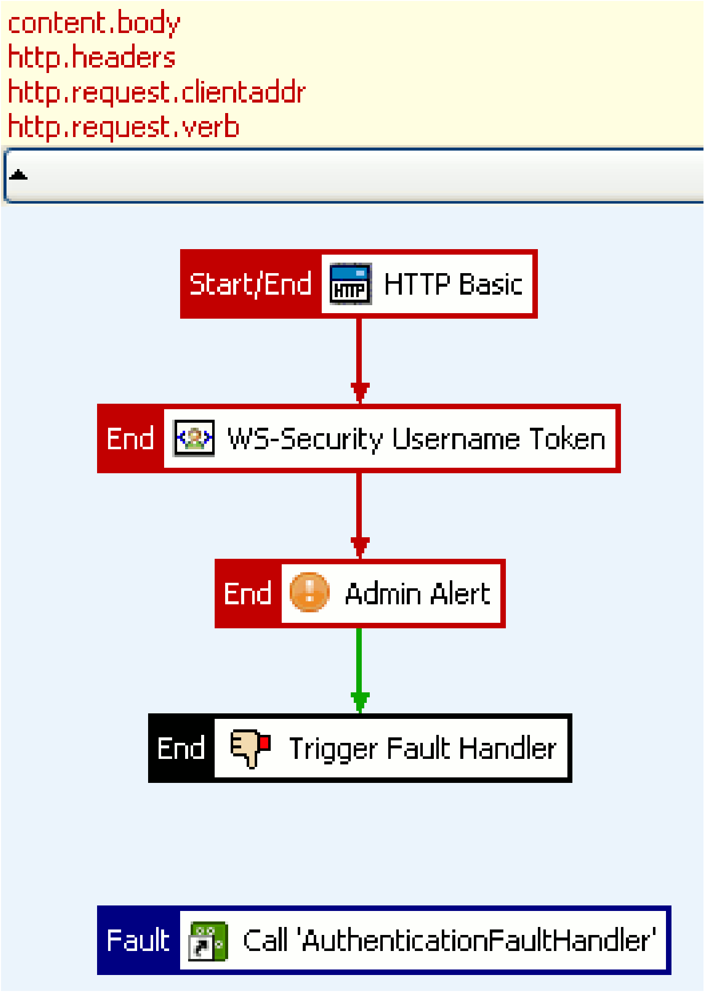
Requires
http.headers
http.request.verb

Generates
authentication.method
authentication.subject.format
authentication.subject.id
authentication.subject.orig.format
authentication.subject.orig.id
authentication.subject.password

If a filter requires an attribute as input that has not been generated in the previous execution steps, the filter is displayed in a different color in the Policy Studio (default is red). You can configure the color used to display missing attributes in the Policy Studio **Preferences** menu. Alternatively, you can also view all required attributes by right-clicking the canvas, and selecting **Show All Attributes**.

A missing attribute may indicate a problem that you need to investigate (for example, in the chaining of filters or policies, or that the policy can not run on its own). This is often the case for reusable filters, such as those displayed in the previous example.

At the policy level, you also can click the horizontal bar at the top of the Policy Studio canvas to show the list of all attributes required as input to the entire policy. If any attributes are generated by the policy, you can click a corresponding bar at the bottom to see a list of generated attributes. The following example screen shot shows the attributes required by an **Authenticate** policy:

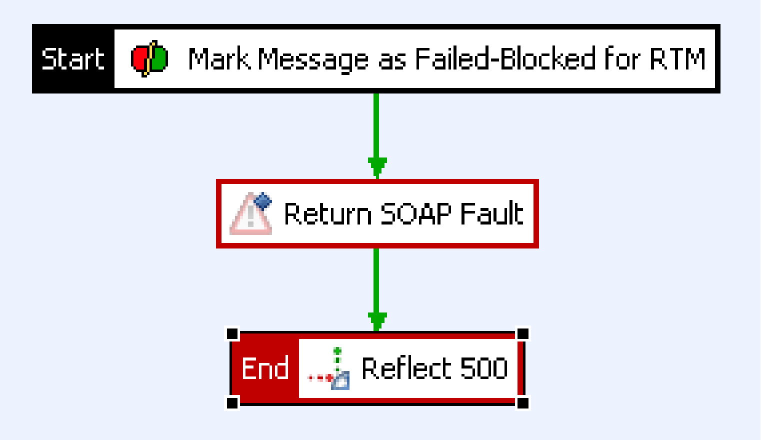


Selector
A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata (for example, in message attributes, a Key Property Store (KPS), or environment variables). For example, the following selector returns the value of a message attribute:

```
${http.request.clientaddr}
```

Selectors are powerful a feature for System Integrators (SIs) and Independent Software Vendors (ISVs) when extending the API Gateway to integrate with other systems. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Faults
When a SOAP transaction fails, you can use a SOAP fault to return error information to the SOAP client. By default, the API Gateway returns a basic SOAP fault to the client when a message filter fails. You can add a **SOAP Fault** filter to a policy to return more detailed error information to the client. For example, the previous example screen shot shows an **AuthenticationFaultHandler**, which is a policy shortcut to the following fault handler policy:



Policy Shortcut

A policy shortcut enables you to create a link from one policy to another policy. For example, you could create a policy that inserts security tokens into a message, and another that adds HTTP headers. You can then create a third policy that calls the other two policies using **Policy Shortcut** filters.

A policy shortcut chain enables you to run a series of policies in sequence without needing to create a policy containing policy shortcuts. In this way, you can create modular policies to perform certain specific tasks, such as authentication, content filtering, returning faults, or logging, and then link these policies together in a sequence using a policy shortcut chain. You can also use API Service Manager to automate the creation of a policy shortcut chain simply by dragging and dropping existing policies into a composite policy.

Alerts

The API Gateway can send alert messages for specified events to various alerting destinations. System alerts are usually sent when a filter fails, but they can also be used for notification purposes. For example, the API Gateway can send system alerts to the following destinations:

- Email Recipient
- Check Point Firewall-1 (OPSEC)
- Local Sylsog
- Remote Sylsog
- SNMP Network Management System
- Twitter
- Windows Event Log

You can configure alert destinations, and then add an **Alert** filter to a policy, specifying the appropriate alert destination.

Policy Container

A policy container is used to group similar policies together (for example, all authentication or logging policies), or policies that relate to a particular service. A number of useful policies are provided in the **Policy Library** container (for example, policies that return faults, and policies that block threatening content). You can add your own policies to this container, and add your own policy containers to suit your requirements.

Policy Context

Policies can execute in a specified context. For example, you can set a context by associating a relative execution path or listener with a policy. When a policy is called from another policy, the context is set to the calling policy name (for example, **Authenticate**). In the Policy Studio, you can select a context from the **Context** drop-down list at the bottom of the policy canvas. The Policy Studio then displays whether the attributes required for execution are available in that context. The **Context** list includes all connected relative paths, listeners, Web Services, SMTP services, and policy shortcuts that use the selected policy. Click the **View navigator node** button to display the selected context.

Process

A Process is an instance of the API Gateway capable of running on a host. You can use Policy Studio to configure and deploy API Gateway Processes. You can configure various features at the Process level (for example, HTTP(S) interfaces, file transfer services, JMS services, and Remote Hosts).

Listeners

You can define different types of listeners and associate them with specific policies. For example, listeners include the following types:

- HTTP/HTTPS
- Directory Scanner
- POP mail server connection
- JMS connection
- TIBCO RV/EMS connection

The API Gateway can be used to provide protocol mediation (for example, receiving a SOAP request over JMS, and transforming it into a SOAP/HTTP request to a back-end service). For HTTP/HTTPS listeners, policies are linked to a relative path. Otherwise, policies are linked to the listener itself. You can associate a single policy with multiple listeners.

Remote Hosts

You can define a remote host when you need more control of the connection settings to a particular server. The available connection settings include the following:

- HTTP version
- IP addresses
- Timeouts
- Buffers
- Caches

For example, by default, the API Gateway uses HTTP 1.1. You can force it to use HTTP 1.0 using Remote Host settings. You must also define a Remote Host if you want to track real-time metrics for a particular host.

Servlet Applications

The API Gateway provides a Web server and servlet application server that can be used to host static content (for example, documentation for your project), or servlets providing internal services. Static content or servlets can be accessed from a policy using the **Call Internal Service** filter. This feature is not meant to replace an enterprise J2EE server, but rather to enable you to write functionality using technology such as servlets.

Configuration Profile

A Configuration Profile contains the configuration information required to run the API Gateway. For example, a specific Configuration Profile instance can store certificates, users, core policies and services, external connections, or listeners. A Configuration Profile can have a number of versions, which are created by users. You can use the Policy Studio to deploy Configuration Profile versions to API Gateway Processes, and to copy existing versions to create new Configuration Profiles.

Service Virtualization

When you register an API service or Web Service, and deploy it to the API Gateway, the API Gateway virtualizes the service. Instead of connecting to the service directly, clients connect through the API Gateway. The API Gateway can then apply policies to messages sent to the destination service (for example, to enable security, monitoring, and acceleration).

Starting the API Gateway Tools

Overview

This topic shows the preliminary steps required before you begin working through the Getting Started tutorial. It explains how to start the API Gateway, the web-based API Gateway Manager tools and the Policy Studio. Finally, this topic shows the main steps in the Getting Started tutorial.

Before you Begin

Before you start the API Gateway tools, do the following:

Check System Requirements
You should ensure that your target machine and platform are supported.

Install the API Gateway and Policy Studio
If you have not already done so, see the *Oracle API Gateway Installation and Configuration Guide*.

Configure a Managed Domain
If you have not already created a domain, you can use the `managedomain` script to configure a domain. You should also ensure that the Admin Node Manager is running.

Create the API Gateway Instance
If you have not already created a domain, you can use the `managedomain` script create the API Gateway Instance. For more details, see the *Oracle API Gateway Installation and Configuration Guide*.

Launching API Gateway Manager

To access the web-based API Gateway Manager tools, perform the following steps:

Note

You must ensure that the Admin Node Manager is running before you can access the web-based API Gateway Manager tools.

1. Enter the following URL:

   ```
   https://HOST:8090/
   ```

   `HOST` refers to the hostname or IP address of the machine on which the API Gateway is running (for example, `https://localhost:8090/`).
2. Enter your user name and password. The default values are as follows:

User Name	`admin`
Password	`changeme`

3. Click the appropriate button in the API Gateway Manager screen in the browser. The **Dashboard** view is displayed by default.

The API Gateway Manager includes the following main views:

- **Dashboard**: overall system health, message traffic summary, and topology (domain, hosts, API Gateways, and groups).
- **Monitoring**: real-time monitoring of all the traffic processed by the API Gateway. Includes statistics at the system level and for services invoked and remote hosts connected to.
- **API Service Manager**: enables you to manage API services, Web services, and policies (for example, virtualize services, and assign policies to them).
- **Traffic**: a message log of the HTTP, HTTPS, JMS, and FTP traffic processed by the API Gateway.
- **Logs**: API Gateway trace log, audit log, and access log files.
- **Events**: API Gateway log points, alerts, and SLA alerts.
- **Settings**: Enables you to configure dynamic API Gateway logging.

Starting Policy Studio

To start Policy Studio, perform the following steps:

1. In your Policy Studio installation directory, enter the `policystudio` command.
2. In the **Policy Studio**, click a link to connect to the Admin Node Manager session.
3. In the **Open Connection** dialog, click **OK** to accept the default settings. For more details, see *Connection Details*.
4. The **Oracle API Gateway** instance is displayed in the **Topology** view.
5. In the **Topology** view, double-click the **Oracle API Gateway** process to load the configuration for the active API Gateway.
6. If a passphrase has been set, enter it in the **Enter Passphrase** dialog, and click **OK**. Alternatively, if no passphrase has been set, click **OK**.

Policy Studio enables you to perform the full range of API Gateway configuration and management tasks. This includes tasks such as create and assign policies, import Services, optimize API Gateway configuration settings, and manage API Gateway deployments. For more details on using the Policy Studio to manage API Gateway processes and configurations, see *Getting Started with Managing Deployments*.

Getting Started Tutorial

When you have completed the preliminary steps, and started the API Gateway components, you can then start working through the Getting Started Tutorial. This tutorial is based on a simple Apache Axis Service. It includes the following main steps:

1. *Virtualizing a Service*
2. *Monitoring Services*
3. *Troubleshooting*

Virtualizing a Service

Overview

This topic explains how to virtualize an example service using the web-based API Service Manager tool available in API Gateway Manager. It uses a simple Google Search REST API service. This topic assumes you have already performed all the steps described in *Starting the API Gateway Tools*.

Accessing the Example Service

The example Google Search REST API service is based on Asynchronous JavaScript and XML (AJAX). This service supports an HTTP GET method, and returns a JSON-encoded response with an embedded status code. The following URL shows an example Google Search method call:

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=Einstein
```

The following example shows an extract from a corresponding JSON-encoded response:

```
{
  "responseData": {
      "results": [
          {
             "GsearchResultClass": "GwebSearch",
             "unescapedUrl": "http://en.wikipedia.org/wiki/Albert_Einstein",
             "url": "http://en.wikipedia.org/wiki/Albert_Einstein",
             "visibleUrl": "en.wikipedia.org",
             "cacheUrl": "http://www.google.com/search?q=cache:dx1QfR_DgCUJ:
               en.wikipedia.org",
             "title": "Albert <b>Einstein</b> - Wikipedia, the free encyclopedia",
             "titleNoFormatting": "Albert Einstein - Wikipedia, the free encyclopedia",
             "content": "Albert <b>Einstein</b> was a German theoretical physicist
             who developed the theory of  general relativity, effecting a revolution
             in physics ...
          },
          ....
          {
             "GsearchResultClass": "GwebSearch",
             "unescapedUrl": "http://www.einstein.edu/",
             "url": "http://www.einstein.edu/", "visibleUrl": "www.einstein.edu",
             "cacheUrl": "http://www.google.com/search?q=cache:np0jSqbXd7EJ:
              www.einstein.edu",
             "title": "<b>Einstein</b> Healthcare Network | Hospitals in Philadelphia,
              PA Area", "titleNoFormatting": "Einstein Healthcare Network | Hospitals
              in Philadelphia, PA Area", "content": "Philadelphia's largest independent
              Academic Medical Center with 7 hospitals in the Philadelphia & Montgomery
              Area."
          }
       ...
      }
  },
  "responseDetails": null,
  "responseStatus": 200
}
```

Important

The Google Search API service is provided for illustration only, and is not intended for production use. For more details, see https://developers.google.com/web-search/terms

Creating a Workspace in API Service Manager

You must first create a workspace in API Service Manager as follows:

1. Click the **API Service Manager** button in the API Gateway Manager toolbar.
2. Select the group and API Gateway instance that you wish to use to virtualize the service (for example, the `Group1` and `APIServer1`).
3. Click the **Create** button to create the workspace.
4. Enter the configuration passphrase (if one exists for this API Gateway instance), and click **OK**.
5. Click the **Virtualize API Service** button on the left in the toolbar of the **API Services** tab to launch the **New API Service** wizard.

Step 1—Basic Information

The first step in the **New API Service** wizard is to specify the service URL. Perform the following steps:

1. Click **No, my Service will be defined manually**, and enter `GoogleSearch` in the **Name** field.
2. Enter the following URL in the **Destination URL** field:

   ```
   http://ajax.googleapis.com/ajax/services/search/web?v=1.0
   ```

   Alternatively, if you wish to virtualize service with a WSDL file, click **Yes, I know a URL from which to get a WSDL**, and enter a **WSDL URL** for the service. For more details, see *Managing API Services*.
3. Click **Next** to specify how the service is exposed.

Step 2—Service Exposure

The second step in the wizard enables you to specify how the service is exposed. For example, this includes details such as the protocol (`HTTP`), services group (`Default Services`), and relative path (`/ajax/services/search/web`). For example, you may wish to virtualize a service on a different relative path. You can also click **Show Details** to view the default port address.

Click **Next** to use the default values for the example service.

Step 3—Request Processing

The third step in the wizard enables you to specify policy packages used for request processing (for example, an OAuth policy package for authentication). Click **Next** to use no request processing for the example service.

For more details on policy packages, see *Configuring Policy Packages*

Step 4—Routing

The fourth step in the wizard enables you to specify policy packages used for routing (for example, a JMS policy package). Click **Next** to use the **Default URL-based Routing** policy package for the example service.

Step 5—Response Processing

The fifth step in the wizard enables you to specify policy packages used for response processing (for example, a policy package that removes sensitive information such as credit card details from the message). Click **Next** no response processing for the example service.

Step 6—Monitoring

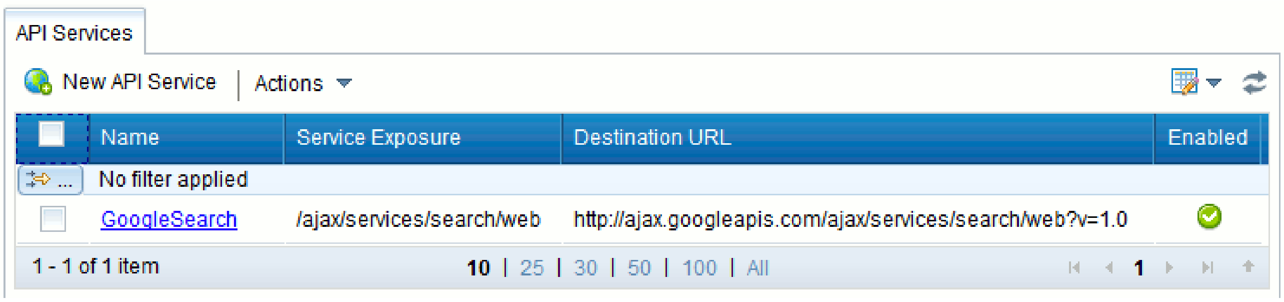
The sixth step in the wizard enables you to specify the monitoring options for the service. For example, these include monitoring the service usage, usage per client, and client usage. By default, the `authentication.subject.id` message attribute is used to identify the client. Click **Next** to use the default values for the example service.

Step 7—Tags

The final step in the wizard enables you to specify tags for this service. Tags are user-friendly names to help organize, search, and browse API Gateways and services in API Gateway Manager and Policy Studio. Perform the following steps:

1. Click the green plus icon to add a tag.
2. Enter a **Tag** name (for example, `Dept`).
3. Enter a **Value** (for example, `QA`).
4. Click **Finish**.

The virtualized service is displayed on the **API Services** tab. For more details, see *Managing API Services*.



□	Name	Service Exposure	Destination URL	Enabled
□	GoogleSearch	/ajax/services/search/web	http://ajax.googleapis.com/ajax/services/search/web?v=1.0	✓

Deploying to a Group

When you have completed the steps in the wizard, you must deploy the updated configuration to a API Gateway group, or a subset of API Gateways in a group. Perform the following steps:

1. Click **Actions** -> **Deploy** on the left in the **API Services** tab.
2. In the **Deployment Wizard**, select the group and API Gateway instance(s) to which you wish to deploy the current working configuration, and click the **Next**.
3. Enter a comment for this deployment (for example, `registering google search service`).
4. Click **Deploy**.
5. Click **Finish**.

Note

Services virtualizeed using API Service Manager and deployed to the API Gateway can also be viewed in the Policy Studio tree under the **Business Services** -> **API Service Manager** node. For details on using Policy Studio to import services, see *Web Service Repository*.

Accessing the Virtualized Service

When you virtualize a service, and deploy it to the API Gateway, the API Gateway protects the service. This means that instead of connecting to the service directly, clients connect through the API Gateway. The API Gateway can then apply policies to messages sent to the destination service (for example, to enable security, routing, monitoring, or acceleration).

When the virtualized service is deployed to the API Gateway, the published URL now uses the host and port of the API Gateway. In the case of the example service, you can test the new URL on which the service is available to clients in your browser. Using the default service exposure settings, the virtualized example service is availalbe on the following URL path:

```
http://HOST:8080/ajax/services/search/web
```

where `HOST` is the machine on which the API Gateway is running.

Monitoring a Service

When you have virtualized the example Service, the next step is to monitor the service using the API Gateway Manager monitoring tools. For details, see *Monitoring Services*.

Monitoring Services

Overview

This topic explains how to monitor example services using the API Gateway Manager monitoring tools. It assumes that you have already performed the steps in the following topics:

1. *Starting the API Gateway Tools*
2. *Virtualizing a Service*

Enabling Monitoring

You must first ensure that traffic monitoring and real-time monitoring are enabled:

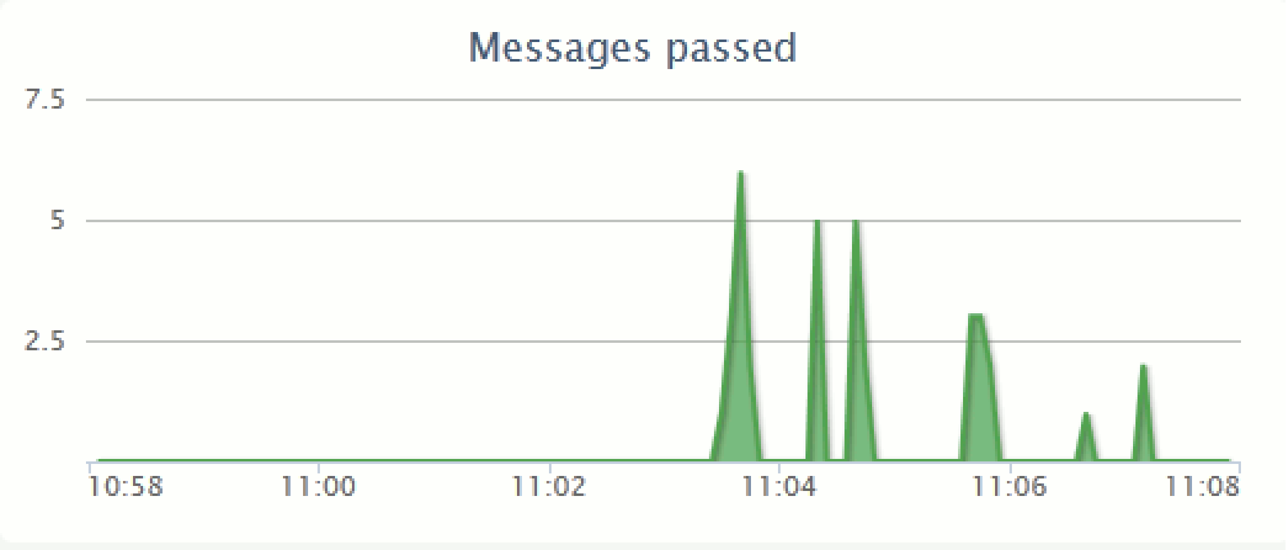
1. In the Policy Studio tree, select the **Settings** node, and select the **Traffic Monitor** tab at the bottom.
2. On the **Traffic Monitor** tab, ensure **Enable Traffic Monitor** is selected.
3. Select the **Metrics** tab at the bottom, and ensure **Enable real time monitoring** is selected.
4. Click the **Deploy** icon in the Policy Studio toolbar to deploy these settings to the API Gateway. Alternatively, press F6.

Important

Traffic monitoring is required for the purposes of this demo. However, enabling traffic monitoring may have a negative impact on performance. If you wish to maximize performance, you can disable these settings. For more details, see *Configuring Traffic Monitoring*.

Viewing Real-time Monitoring

You can view a wide range of monitoring data in the API Service Manager. For example, this includes message status, message traffic, filter execution path, message content, system, services, and remote hosts. You can view real-time traffic monitoring summary data on the main **Dashboard** tab in the **TRAFFIC** section. The following example shows the number of messages that have been passed by the API Gateway on to a service:



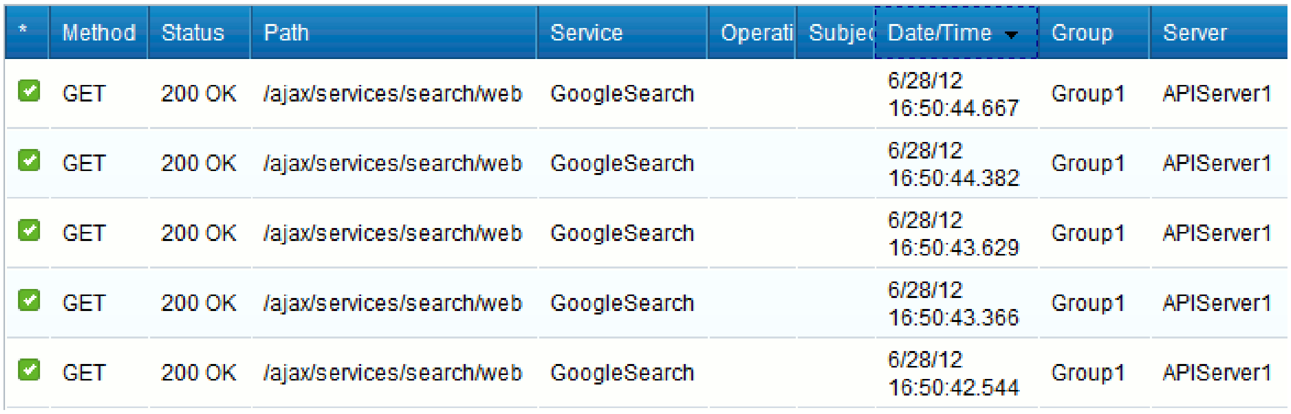
Each time you send test messages through the API Gateway to an example service (for example, using API Gateway Explorer or the Send Request (SR) tool), the message status is displayed in the **TRAFFIC** section.

Viewing Message Traffic

You can use the traffic monitoring tools in API Gateway Manager for operational diagnostics and root cause analysis. The **Traffic** view provides a web-based message log of the HTTP, HTTPS, JMS, and FTP traffic processed by the API Gateway. You can perform tasks such as the following:

- Filter messages on a range of criteria (for example, transaction ID, service name, or remote host)
- Drill down to view message contents
- View performance statistics (for example, number of requests, average bytes sent, or average duration)

For example, you can click the **Traffic** button in the API Service Manager to view summary information for each message sent to the API Gateway. Alternatively, you can click one of the summary charts displayed on the **Dashboard** (for example, **Messages passed** or **Messages failed**). This displays the message traffic automatically filtered according to your selection. The following simple example shows the details displayed on the **Traffic** tab for **Messages passed** by the API Gateway:



*	Method	Status	Path	Service	Operati	Subjec	Date/Time	Group	Server
✓	GET	200 OK	/ajax/services/search/web	GoogleSearch			6/28/12 16:50:44.667	Group1	APIServer1
✓	GET	200 OK	/ajax/services/search/web	GoogleSearch			6/28/12 16:50:44.382	Group1	APIServer1
✓	GET	200 OK	/ajax/services/search/web	GoogleSearch			6/28/12 16:50:43.629	Group1	APIServer1
✓	GET	200 OK	/ajax/services/search/web	GoogleSearch			6/28/12 16:50:43.366	Group1	APIServer1
✓	GET	200 OK	/ajax/services/search/web	GoogleSearch			6/28/12 16:50:42.544	Group1	APIServer1

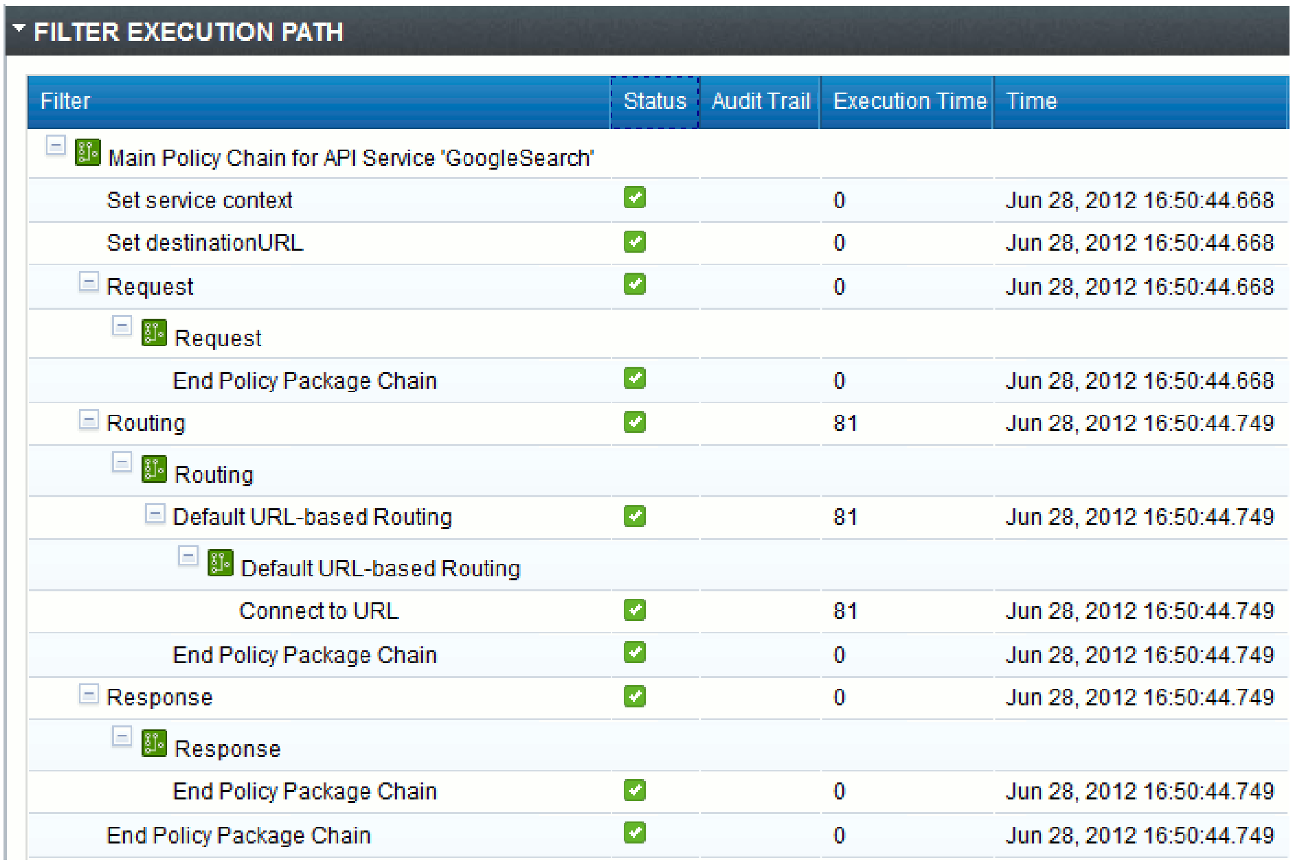
Filtering Message Traffic

In the **SELECTION** pane on the left of the **Traffic** tab, you can click the **Apply** button to filter the messages displayed based on a range of criteria. For example, the default filters include **REQUEST FROM** (Client or API Gateway), **MAX RESULTS PER SERVER**, **TRANSACTION STATUS**, and **TIME INTERVAL**.

You can click **Add Filter** to search on different criteria (for example, Service Name, Remote Host, Authentication Subject, Transaction ID, and Operation). The API Gateway inserts a transaction ID in all HTTP and HTTPS traffic in a header named `X-CorrelationID`. When you have selected your search criteria, click the **Apply** button.

Viewing Message Content

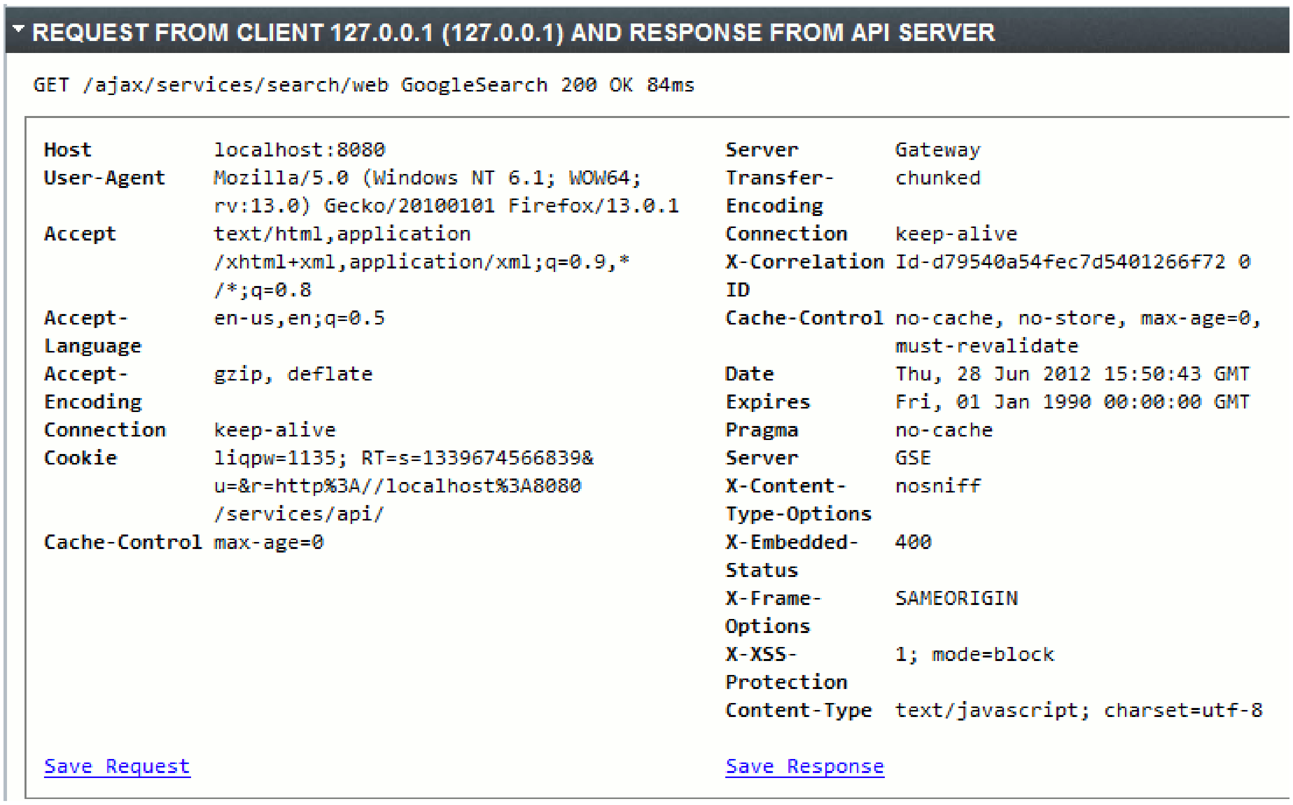
When you click a selected message listed on the **Traffic** tab, this displays the message filter execution path and the contents of each request message and response message. The following example displays the message path for a simple Google Search message:



FILTER EXECUTION PATH

Filter	Status	Audit Trail	Execution Time	Time
Main Policy Chain for API Service 'GoogleSearch'				
Set service context	✓		0	Jun 28, 2012 16:50:44.668
Set destinationURL	✓		0	Jun 28, 2012 16:50:44.668
Request	✓		0	Jun 28, 2012 16:50:44.668
Request				
End Policy Package Chain	✓		0	Jun 28, 2012 16:50:44.668
Routing	✓		81	Jun 28, 2012 16:50:44.749
Routing				
Default URL-based Routing	✓		81	Jun 28, 2012 16:50:44.749
Default URL-based Routing				
Connect to URL	✓		81	Jun 28, 2012 16:50:44.749
End Policy Package Chain	✓		0	Jun 28, 2012 16:50:44.749
Response	✓		0	Jun 28, 2012 16:50:44.749
Response				
End Policy Package Chain	✓		0	Jun 28, 2012 16:50:44.749
End Policy Package Chain	✓		0	Jun 28, 2012 16:50:44.749

The following example shows the corresponding message content for the selected message displayed below:



▾ REQUEST FROM CLIENT 127.0.0.1 (127.0.0.1) AND RESPONSE FROM API SERVER

```
GET /ajax/services/search/web GoogleSearch 200 OK 84ms
```

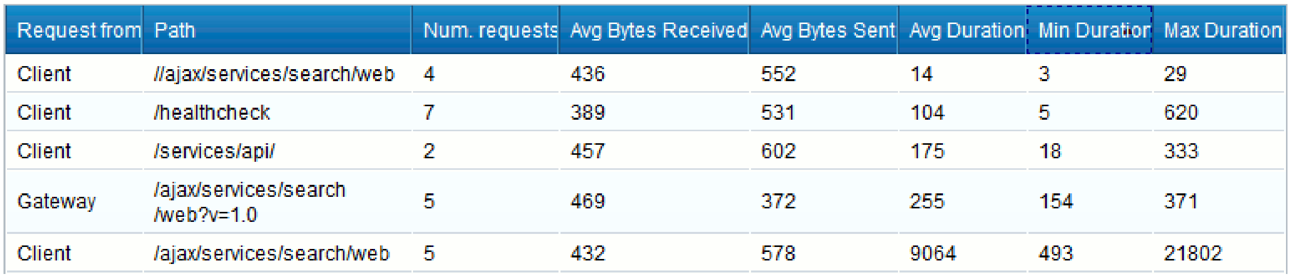
Host	localhost:8080
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0) Gecko/20100101 Firefox/13.0.1
Accept	text/html,application /xhtml+xml,application/xml;q=0.9,* /*;q=0.8
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip, deflate
Connection	keep-alive
Cookie	liqpw=1135; RT=s=1339674566839& u=&r=http%3A//localhost%3A8080 /services/api/
Cache-Control	max-age=0

Server	Gateway
Transfer-Encoding	chunked
Connection	keep-alive
X-Correlation ID	Id-d79540a54fec7d5401266f72 0
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Date	Thu, 28 Jun 2012 15:50:43 GMT
Expires	Fri, 01 Jan 1990 00:00:00 GMT
Pragma	no-cache
Server	GSE
X-Content-Type-Options	nosniff
X-Embedded-Status	400
X-Frame-Options	SAMEORIGIN
X-XSS-Protection	1; mode=block
Content-Type	text/javascript; charset=utf-8

You can click **Save Request** or **Save Response** to download the message contents and save them to a file.

Viewing Performance Statistics

The **Performance** tab displays performance statistics for the HTTP and HTTPS traffic processed by the API Gateway. For example, these include the number of requests, average bytes sent, and average duration. For example, the **Performance** page is displayed as follows:



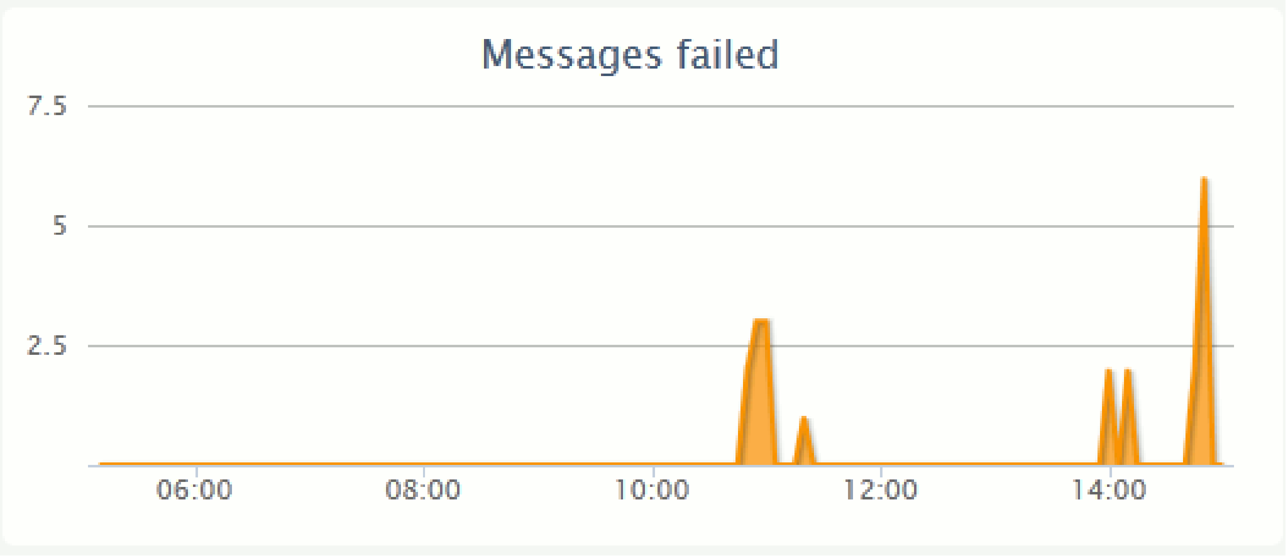
Request from	Path	Num. requests	Avg Bytes Received	Avg Bytes Sent	Avg Duration	Min Duratior	Max Duration
Client	//ajax/services/search/web	4	436	552	14	3	29
Client	/healthcheck	7	389	531	104	5	620
Client	/services/api/	2	457	602	175	18	333
Gateway	/ajax/services/search /web?v=1.0	5	469	372	255	154	371
Client	/ajax/services/search/web	5	432	578	9064	493	21802

Filtering Performance Statistics

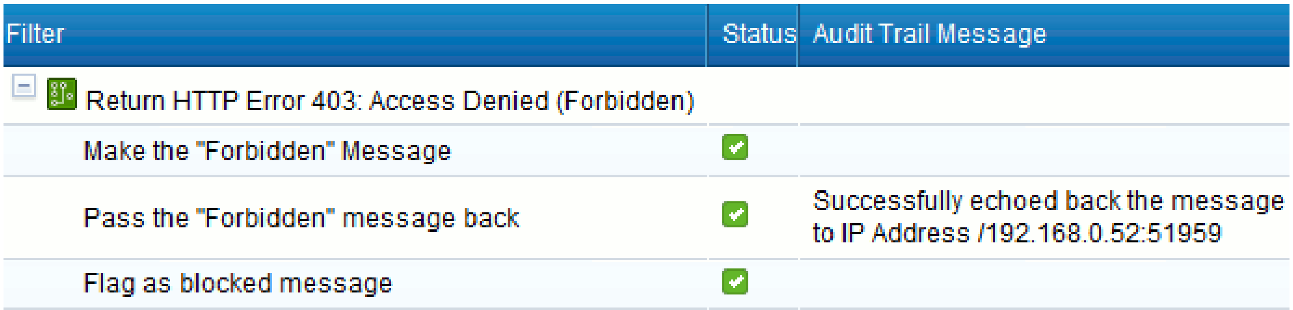
You can click the **Apply** in the left pane to filter the performance statistics displayed based on different criteria. By default, the statistics are grouped by path name, with a time interval of 1 day. You can select different criteria from the **GROUP BY** and **TIME INTERVAL** lists. When you have selected your search criteria, click the **Apply** button.

Detecting Malformed Messages

Messages with malformed content or an incorrect relative path are blocked by the API Gateway and displayed on the **Dashboard** tab in the **TRAFFIC** section as follows:



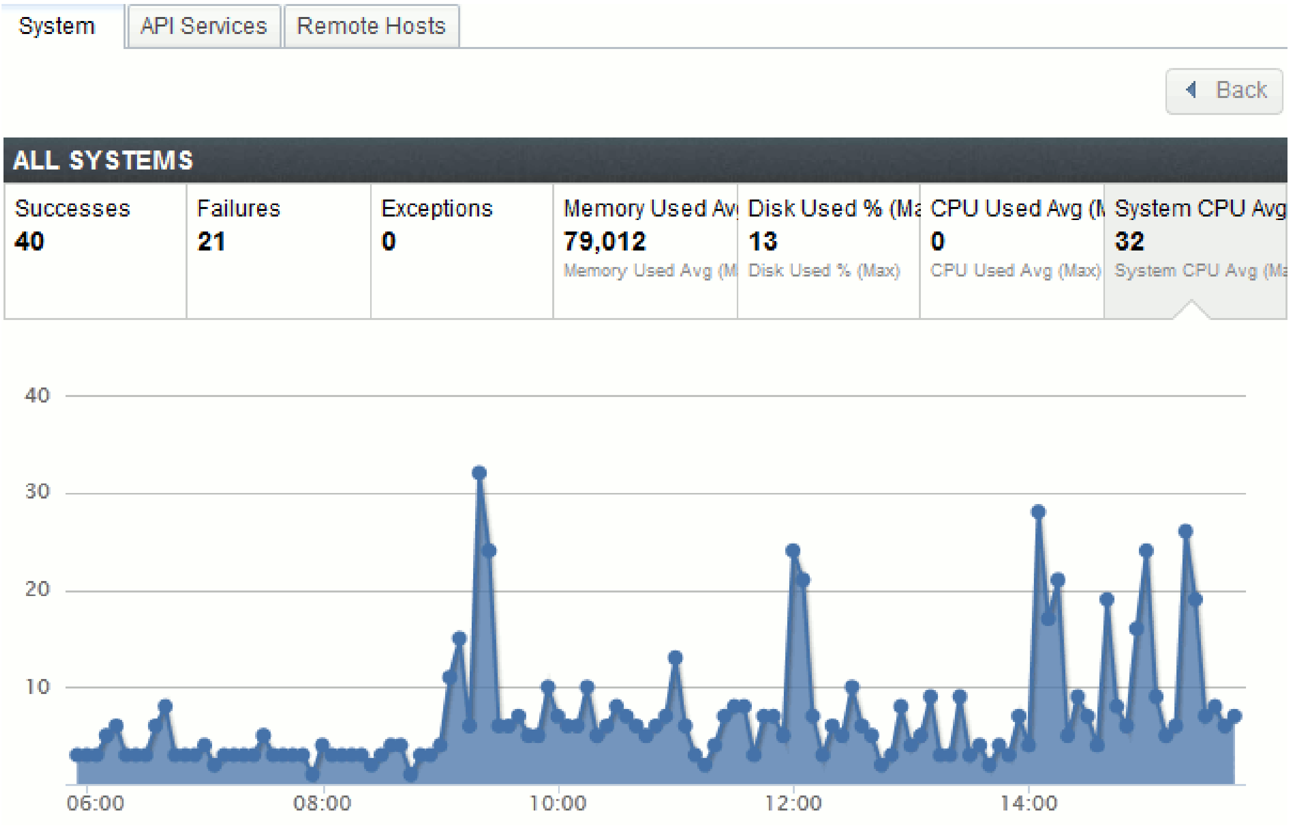
You can click the chart to display the list of failed messages automatically filtered on the **Traffic** tab. Click a message in the list to display the filter execution path and message content. The following example shows the execution path of a malformed message that has been blocked by the API Gateway:



Filter	Status	Audit Trail Message
Return HTTP Error 403: Access Denied (Forbidden)		
Make the "Forbidden" Message	✓	
Pass the "Forbidden" message back	✓	Successfully echoed back the message to IP Address /192.168.0.52:51959
Flag as blocked message	✓	

Monitoring System Data

The **Monitoring** view enables you to monitor **System**, **API Services**, and **Remote Host** statistics. For example, on the **System** tab, when you click a panel in the **ALL SYSTEMS** section at the top of the screen, a graph for the selected setting is displayed below. The following example shows the graph displayed for the **System CPU Avg (Max)** setting selected on the right:



Successes	Failures	Exceptions	Memory Used Av	Disk Used % (Ma	CPU Used Avg (N	System CPU Avg
40	21	0	79,012	13	0	32
			Memory Used Avg (M	Disk Used % (Max)	CPU Used Avg (Max)	System CPU Avg (Ma

Configuring Trace and Log Settings

You can click the **Settings** button in the toolbar to configure trace, logging, and monitoring settings on-the-fly. These are dynamic settings, so you do not need to refresh or deploy to the API Gateway. For example, the top-level **SYSTEM SETTINGS** enable you to specify whether inbound and outbound transactions, the policy path, and message trace are recorded. You can also select an HTTP interface in the tree on the left to configure the **INTERFACE SETTINGS**, **TRAFFIC MONITORING SETTINGS**, and interface trace level.

Select the API Gateway process on the left to configure the process trace level. Finally, you can also select a Relative Path or Service on the left, and select the **SERVICE SETTINGS**, **LOGGING LEVEL**, or **PAYLOAD LOGGING** on the right.

For more details on how to configure tracing for the API Gateway and logging for specific message filters. For details, see the topic on *Troubleshooting*.

Using Oracle API Gateway Analytics

This tutorial shows how to monitor an example service using the monitoring tools provided with the API Gateway. Oracle API Gateway Analytics is a separately installed component that enables you to monitor services and to generate reports on the stored message traffic history in your domain. For more details, see *Using Oracle API Gateway Analytics*.

Troubleshooting

Overview

The **Logs** view in API Gateway Manager enables you to view and search the contents of API Gateway trace log, audit log, and access log files. This topic explains how to configure tracing for the API Gateway, and how to configure logging for message filters to provide an audit trail of message transactions.

Viewing API Gateway Trace Files

Each time the API Gateway starts up, by default, it outputs a trace file to the API Gateway `trace` directory (for example, `INSTALL_DIR\groups\group-2\server1\trace`). The following example shows an extract from a default API Gateway trace file:

```
INFO    15/Jun/2012:09:54:01.047 [1b10] Realtime monitoring enabled
INFO    15/Jun/2012:09:54:01.060 [1b10] Storing metrics in database disabled
INFO    15/Jun/2012:09:54:03.229 [1b10] cert store configured
INFO    15/Jun/2012:09:54:03.248 [1b10] keypairs configured
...
```

The trace file output takes the following format:

```
TraceLevel   Timestamp [thread-id] TraceMessage
```

For example, the first line in the above extract is described as follows:

TraceLevel	INFO
Timestamp	15/Jun/2012:09:54:01.047 (*day*:*hours*:*minutes*:*seconds*:*milliseconds*)
Thread-id	[1b10]
TraceMessage	Realtime monitoring enabled

Setting API Gateway Trace Levels

The possible trace levels in order of least to most verbose output are as follows:

- `FATAL`
- `ERROR`
- `INFO`
- `DEBUG`
- `DATA`

where `FATAL` is the least verbose and `DATA` is the most verbose. The default trace level is `INFO`.

Setting Trace Levels
You can set the trace level using the following different approaches:

Startup trace	When the Admin Node Manager is starting up, it gets its

	trace level from the `tracelevel` attribute of the `SystemSettings` element in `/system/conf/nodemanager.xml`. You can set the trace level in this file if you need to diagnose boot up issues.
Default Settings trace	When the API Gateway has started, it reads its trace level from the Default Settings for the API Gateway process. To set this trace level in the Policy Studio, click the **Settings** node in the Policy Studio tree, select a **Trace level** from the drop-down list.
Interface level trace	You can configure an HTTP/HTTPS interface with a different trace level to that specified in the Default Settings. For example, the default API Gateway management port (`8090`) has a trace level set to `ERROR` to ensure it is not too verbose. To configure the trace level for an interface in the Policy Studio, right-click the interface under the **Listeners** node, select **Edit**, and select a **Trace level** from the drop-down list.
Dynamic trace	You can also change dynamic API Gateway trace levels on-the-fly in API Gateway Manager. For more details, see the section called “Configuring Trace and Log Settings”.

Configuring API Gateway Trace Files

By default, the most recent trace file is named *`servername`*`.trc` (for example, `server1.trc`). Older trace files are versioned with the highest version number as oldest (for example, `server1.trc.0`, `server1.trc.1` `server1.trc.2`, `server1.trc.3`, and so on).

You can configure the settings for trace file output in `INSTALL_DIR/system/conf/trace.xml`, which is included by `INSTALL_DIR/system/conf/nodemanager.xml`. By default, `trace.xml` contains the following setting:

```
<FileRolloverTrace maxfiles="500" />
```

This setting means that the API Gateway writes trace output to `nodemanager.trc` in the `trace` directory of the API Gateway installation. And the maximum number of files that the `trace` directory can contain is `500`.

FileRolloverTrace Attributes

The `FileRolloverTrace` element can contain the following attributes:

`filename`	File name used for trace output. Defaults to the `tracecomponent` attribute read from the `SystemSettings` element.
`directory`	Directory where the trace file is written. Defaults to `INSTALL_DIR/trace` when not specified.
`maxlen`	Maximum size of the trace file before it rolls over to a new file. Defaults to `16` MB.
`maxfiles`	Maximum number of files that the `trace` directory contains for this `filename`. Defaults to the maximum integer value (`2147483647`).
`rollDaily`	Whether the trace file is rolled at the start of the day. Defaults to `true`.

Writing Trace Output to Syslog

On UNIX and Linux, you can send API Gateway trace output to `syslog`. In your `INSTALL_DIR/system/conf/trace.xml` file, add a `SyslogTrace` element, and specify a `facility`. For example:

```
<SyslogTrace facility="local0"/>
```

Running Trace at DEBUG level

When troubleshooting, it can be useful to set to the trace level to `DEBUG` for more verbose output. When running a trace at `DEBUG` level, the API Gateway outputs the status of every policy and filter that it processes into the trace file.

Debugging a Filter

The trace output for a specific filter takes the following format:

```
Filter name {
    Trace for the filter is indented
    to the following point to make it clear
    to identify output from the filter
} status, in x milliseconds
```

The status is `0`, `1`, or `2`, depending if the filter failed, succeeded, or aborted. For example, the result of an **WS-Security Username Token** filter running successfully is as follows:

```
DEBUG 12:43:59:093 [11a4] run filter [WS-Security Username Token] {
DEBUG 12:43:59:093 [11a4]    WsUsernameTokenFilter.invoke: Verify username and password
DEBUG 12:43:59:093 [11a4]    WsAuthN.getWSUsernameTokenDetails:
                             Get token from actor=current actor
DEBUG 12:43:59:093 [11a4]    Version handler -  creating a new ws block
DEBUG 12:43:59:108 [11a4]    Version handler - adding the ws element to the wsnodelist
DEBUG 12:43:59:108 [11a4]    Version handler -  number of ws blocks found:1
DEBUG 12:43:59:124 [11a4]    No timestamp passed in WS block, no need to check timestamp
DEBUG 12:43:59:139 [11a4]    WsAuthN.getWSUsernameTokenDetails: Check <Created> element
                             in token. Value=2010-08-06T11:43:43Z
DEBUG 12:43:59:139 [11a4]    WS Nonce TimeStamp Max Size is 1000 and wsNonces cache is 4
DEBUG 12:43:59:139 [11a4]    Add WS nonce for key [joe:2010-08-06T11:43:43Z].
                             New cache size [5].
DEBUG 12:43:59:155 [11a4]    WsBasicAuthN.getUsername: Getting username
DEBUG 12:43:59:171 [11a4]    WS-Security UsernameToken authN via CLEAR password
DEBUG 12:43:59:171 [11a4]    VordelRepository.checkCredentials: username=joe
DEBUG 12:43:59:186 [11a4] } = 1, in 62 milliseconds
```

Debugging a policy

The trace output for a policy shows it running with all its contained filters, and takes the following format:

```
policy Name {
    Filter 1{
        Trace for the filter
    } status, in x milliseconds
    Filter 2{
        Trace for the filter
    } status, in x milliseconds
}
```

For example, the following extract shows a policy called when running a simple service:

```
DEBUG ... run circuit "/axis/services/urn:cominfo"...
DEBUG ... run filter [Service Handler for 'ComInfoServiceService'] {
DEBUG ...   Set the service name to be ComInfoServiceService
DEBUG ...   Web Service context already set to ComInfoServiceService
DEBUG ...   close content stream
DEBUG ...   Calling the Operation Processor Chain [1. Request from Client]...
DEBUG ...   run filter [1. Request from Client] {
DEBUG ...      run filter [Before Operation-specific Policy] {
DEBUG ...         run circuit "WS-Security UsernameToken AuthN"...
DEBUG ...           run filter [WS-Security Username Token] {
                     ...
DEBUG ...           } = 1, in 62 milliseconds
DEBUG ...           ..."WS-Security UsernameToken AuthN" complete.
DEBUG ...       } = 1, in 74 milliseconds
...
```

Debugging at Startup

When running a startup trace with a `DEBUG` level set in the `SystemSettings`, the API Gateway outputs the configuration that it is loading. This can often help to debug any incorrectly configured items at start up, for example:

```
DEBUG 14:38:54:206 [1ee0] configure loadable module type RemoteHost, load order = 500000
DEBUG 14:38:54:206 [1ee0] RemoteHost {
DEBUG 14:38:54:206 [1ee0]     ESPK: 1035
DEBUG 14:38:54:206 [1ee0]     ParentPK: 113
DEBUG 14:38:54:206 [1ee0]     Key Fields:
DEBUG 14:38:54:206 [1ee0]         name: {csdwmp3308.wellsfargo.com}
DEBUG 14:38:54:206 [1ee0]         port: {7010}
DEBUG 14:38:54:221 [1ee0]     Fields:
DEBUG 14:38:54:221 [1ee0]         maxConnections: {128}
DEBUG 14:38:54:268 [1ee0]         turnMode: {off}
DEBUG 14:38:54:268 [1ee0]         inputBufSize: {8192}
DEBUG 14:38:54:268 [1ee0]         includeContentLengthRequest: {0}
DEBUG 14:38:54:268 [1ee0]         idletimeout: {15000}
DEBUG 14:38:54:268 [1ee0]         activetimeout: {30000}
DEBUG 14:38:54:268 [1ee0]         forceHTTP10: {0}
DEBUG 14:38:54:268 [1ee0]         turnProtocol: {http}
DEBUG 14:38:54:268 [1ee0]         includeContentLengthResponse: {0}
DEBUG 14:38:54:268 [1ee0]         addressCacheTime: {300000}
DEBUG 14:38:54:268 [1ee0]         outputBufSize: {8192}
DEBUG 14:38:54:268 [1ee0]         sessionCacheSize: {32}
DEBUG 14:38:54:268 [1ee0]         _version: {1}
DEBUG 14:38:54:268 [1ee0]         loadorder: {500000}
DEBUG 14:38:54:268 [1ee0]         class: {com.vordel.dwe.NativeModule}
DEBUG 14:38:54:268 [1ee0] }
```

For details on setting trace levels and running a startup trace, see the section called “Setting API Gateway Trace Levels”.

Running Trace at DATA level

When the trace level is set to `DATA`, the API Gateway writes the contents of the messages that it receives and sends to the trace file. This enables you to see what messages the API Gateway has received and sent (for example, to reassemble a received or sent message).

Note

On Windows, you can not rely on the console output because it truncates large messages.

Searching by Thread

Every HTTP request handled by the API Gateway is processed in its own thread, and threads can be reused when an HTTP transaction is complete. You can see what has happened to a message in the API Gateway by following the trace by thread ID. Because multiple messages can be processed by the API Gateway at the same time, it is useful to eliminate threads that you are not interested in by searching for items that only match the thread you want.

You can do this using the search feature in the API Gateway Manager **Logs** view. Enter the thread you wish to search for in the **Only show lines with text** textbox, and click **Refresh**. Alternatively, you can do this on the command line using vi by specifying the thread ID as a pattern to search for in the trace file. In this example, the thread ID is `145c`:

```
:g!/145c/d
```

The following example shows the `DATA` trace when a message is sent by the API Gateway (message starts with `snd`): >

```
DATA 17:45:35:718 [145c]  snd 1495: <POST /axis/services/urn:cominfo HTTP/
     1.1Connection: closeContent-Length: 1295User-Agent: VordelSOAPAction:
     ""Via: 1.0 devsupport2 (Vordel)Host: devsupport2:7070Content-Type:
     text/xml<?xml version="1.0" encoding="UTF-8" standalone="no"?>
     <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <soap:Header>
       <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
       oasis-200401-wss-wssecurity-secext-1.0.xsd">
       <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
         oasis-200401-wss-wssecurity-utility-1.0.xsd"
         wsu:Id="Id-00000128d05aca81-00000000009d04dc-10">
              <wsse:Username>joeuser</wsse:Username>
              <wsse:Nonce EncodingType="utf-8">
                  gmP9GCjoe+YIuK1einlENA==
              </wsse:Nonce>
              <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/
              oasis-200401-wss-username-token-profile-1.0#PasswordText">
                    joepwd
              </wsse:Password>
              <wsu:Created>2010-05-25T16:45:30Z</wsu:Created>
         </wsse:UsernameToken>
         </wsse:Security>
      </soap:Header>
      <soap:Body>
          <ns1:getInfo xmlns:ns1="http://stock.samples">
              <symbol xsi:type="xsd:string">CSCO</symbol>
              <info xsi:type="xsd:string">address</info>
          </ns1:getInfo>
      </soap:Body>
    </soap:Envelope>
>
```

The following example shows the `DATA` trace when a message is received by the API Gateway (message starts with `rcv`):

```
DATA 17:45:35:734 [145c]  rcv 557: <HTTP/1.0 200 OKSet-Cookie: 8Set-Cookie2:
     8Content-Type:
     text/xml; charset=utf-8<?xml version="1.0" encoding="UTF-8"?>
     <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
     XMLSchema-instance">
     <soapenv:Body>
        <ns1:getInfoResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/" xmlns:ns1="http://stock.samples">
            <getInfoReturn xsi:type="xsd:string">San Jose, CA</getInfoReturn>
        </ns1:getInfoResponse>
      </soapenv:Body>
     </soapenv:Envelope>
>
```

If you want to see what has been received by the API Gateway on this thread, run the following command:

```
:g!/145c] rcv/d
```

All `snd` and `rcv` trace statements start and end with < and > respectively. If you are assembling a message by hand from the `DATA` trace, remember to remove characters. In addition, the sending and/or receiving of a single message may span multiple trace statements.

Integrating Trace Output with Apache log4J

Apache log4j is included on the API Gateway classpath. This is because some third-party products that the API Gateway interoperates with require log4j. The configuration for log4j is found in the API Gateway `INSTALL_DIR/system/lib` directory in the `log4j.properties` file.

For example, to specify that the log4j appender sends output to the API Gateway trace file, add the following setting to your `log4j.properties` file:

```
log4j.rootLogger=DEBUG, A1, Vordel
log4j.appender.Vordel=com.vordel.trace.VordelTraceAppender
```

Configuring Logging Output

The API Gateway provides detailed logging for specific message filters (for example, the request, the time of the request, where the request was routed to, and the response returned to the client). You can configure logging to a number of different locations:

- Text file
- XML file
- Database
- Local syslog
- Remote syslog
- System console

To configure where logging information is sent, perform the following steps:

1. In the Policy Studio tree, select the **Settings** node.
2. Click the **Audit Log** tab at the bottom.
3. Specify the required settings on the appropriate tab (for example, **Text File**, **Database**, or **XML File**).
4. Click **OK**.
5. Click the **Deploy** button in the toolbar to deploy your settings to the API Gateway.

For details on configuring all the available options, see the topic on *Audit Log Settings*.

Configuring Log Level and Message

You can configure the log level and log message for a specific filter as follows:

1. In the Policy Studio tree, click a policy to display it in the canvas on the right (for example, **WS-Security UsernameToken AuthN** created in the Getting Started tutorial).
2. Double-click the **WS-Security UsernameToken** filter on the canvas to edit it.
3. Click **Next** to display the **Log Level and Message** screen.
4. Select the **Fatal** and **Failure** log levels for troubleshooting.
5. Specify any non-default log messages if required.

6. Click **Finish**.
7. Click the **Deploy** button in the toolbar to deploy your settings to the API Gateway.

For more details, see the *Log Level and Message* topic.

Further Details
For details on logging the message payload at any point in a policy, see the *Log Message Payload* topic. For details on logging the access details of a message (for example, remote hostname, user login name, and authenticated user name), see the *Access Log Settings* topic.

Getting Help

Context-sensitive help is available from the Policy Studio screens. Click the **Help** button on any screen to display the relevant help page for that screen. If you require further information or assistance, please contact the Oracle Support Team (see *Oracle Contact Details*).

Contacting Support
It is important to include as much information as possible when sending support emails to the Oracle Support team. This helps to diagnose and solve the problem in a more efficient manner. The following information should be included with any support query:

- Name and version of the product (for example, Oracle API Gateway 11.1.2.1.0).
- Details of patches that were applied to the product, if any.
- Platform on which the product is running.
- A clear (step-by-step) description of the problem or query.
- If you have encountered an error, the error message should be included in the email. It is also useful to include any relevant trace files from the `/trace` directory of your product installation, preferably with the trace level set to `DEBUG`.

License Acknowledgments

Overview

Oracle API Gateway uses several third-party toolkits to perform specific types of processing. In accordance with the Licensing Agreements for these toolkits, the relevant acknowledgments are listed below.

Acknowledgments

Apache Software Foundation:
This product includes software developed by the Apache Software Foundation [http://www.apache.org/].

OpenSSL Project:
This product includes software developed by the OpenSSL Project [http://www.openssl.org/] for use in the OpenSSL Toolkit.

Eric Young:
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

James Cooper:
This product includes software developed by James Cooper.

Oracle Contact Details

Contact Details

For online technical assistance, and a complete list of locations, primary service hours, and telephone numbers, contact Oracle Technical Support at the following address:
https://support.oracle.com

Configuring the Sample Policies

Overview

This topic introduces and explains how to setup the example policies available in the `samples` directory of your API Gateway installation. These include the following:

- **Conversion**: exposes a SOAP service over REST.
- **Security**:
  - verifies the digital signature on the request and creates a signature on the response.
  - decrypts the request and encrypts part of the response.
- **Throttling**: limits the number of calls for a service.
- **Virtualized Service**: combines threat protection, content-based routing (target a server according to request contents), and message transformation.

Tip

If you are new to the API Gateway, you should first read the following topics to get familiar with the main concepts and basic steps:

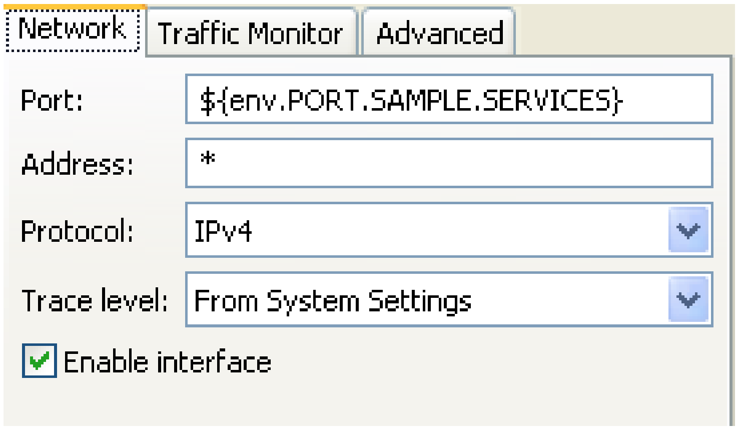
- *Oracle API Gateway Architecture*
- *Oracle API Gateway Concepts*
- *Starting the API Gateway Tools*

This guide assumes that you have already installed and started the API Gateway and Policy Studio. The API Gateway Explorer client GUI testing tool is optional.

Enabling the Sample Services Interface

The HTTP interface for the sample policy services is disabled by default. To enable this interface in the Policy Studio, perform the following steps:

1. In the navigation tree, select **Listeners** -> **Oracle API Gateway** -> **Sample Services** -> ***:${env.PORT_SAMPLE_SERVICES}**.
2. Right-click, and select **Edit** to display the **Configure HTTP Interface** dialog.



Port: ${env.PORT.SAMPLE.SERVICES}

Address: *

Protocol: IPv4

Trace level: From System Settings

☑ Enable interface

3. Select the **Enable interface** setting.
4. Click **OK**.

Alternatively, you can also enable this HTTP interface using the web-based API Gateway Manager tool running on `http://HOST:8090`, where `HOST` is the machine on which the Node Manager is running.

1. Click the **Settings** button in the API Gateway Manager toolbar.
2. Select the HTTP interface node under **Sample Services** on the left.
3. Select the **Interface Enabled** setting on the right.
4. Click the **Apply** button.

Note

Settings made in the web-based API Gateway Manager tool are dynamic settings only, which are not persisted.

Configuring a Different Sample Services Interface

All sample policy services are defined in an HTTP Services group named `Sample Services`. This group uses an HTTP interface running on the port specified in the `${env.PORT_SAMPLE_SERVICES}` environment variable. This external environment variable is set to `8081` by default. If you wish to use a different port, you must configure this variable in the `INSTALL_DIR/conf/envSettings.props` file. For example, you could add the following entry:

```
env.PORT.SAMPLE.SERVICES=8082
```

For more details on setting external environment variables, see the topic on *Deploying the API Gateway in Multiple Environments*.

StockQuote Demo Service

All sample policies use a demo service named `StockQuote`, which is implemented using a set of policies. This service exposes two operations:

- **getPrice**: the policy for this operation uses a sample script to randomly calculate a quote value. Each call to `getPrice()` returns a different value.
- **update**: returns an `Accepted` HTTP code (202).

The `StockQuote` service is exposed on the following relative paths:

- `/stockquote/instance1`
- `/stockquote/instance2`

These relative paths are used in the Virtualized Service sample for content-based routing.

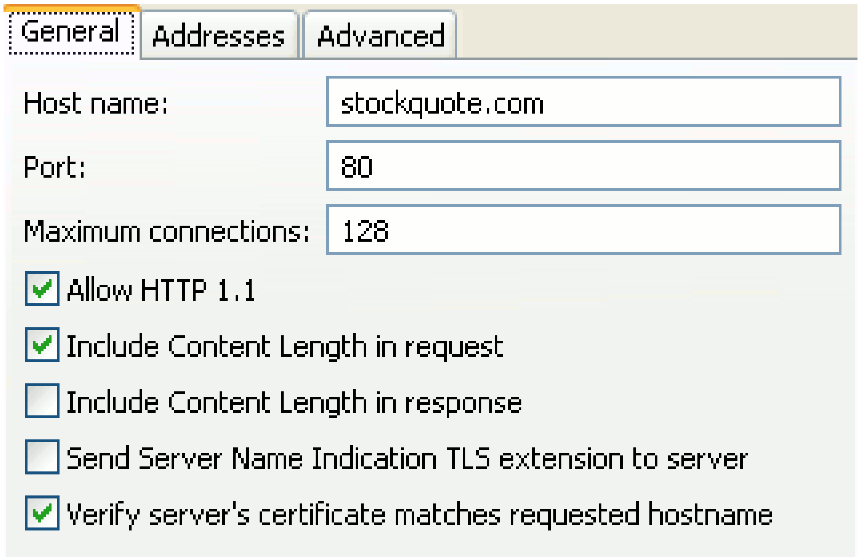
Sample Service URL
A **Connect to URL** filter with the following URL is used to invoke the `StockQuote` service from each of the sample policies:

```
http://stockquote.com/stockquote/instance1
```

The first part of this URL uses a *Remote Host* definition of `stockquote.com`. Remote Hosts are logical names that decouple the hostname in a URL from the server (or group of servers) that handles the request.

Remote Host Settings

In the Policy Studio, the Remote Host configuration is displayed under the process name (`Oracle API Gateway`) in the navigation tree, and is named `stockquote.com:80`. To view its settings, right-click, and select **Edit** to view the **Remote Host Settings** dialog:



Host name: stockquote.com

Port: 80

Maximum connections: 128

☑ Allow HTTP 1.1

☑ Include Content Length in request

□ Include Content Length in response

□ Send Server Name Indication TLS extension to server

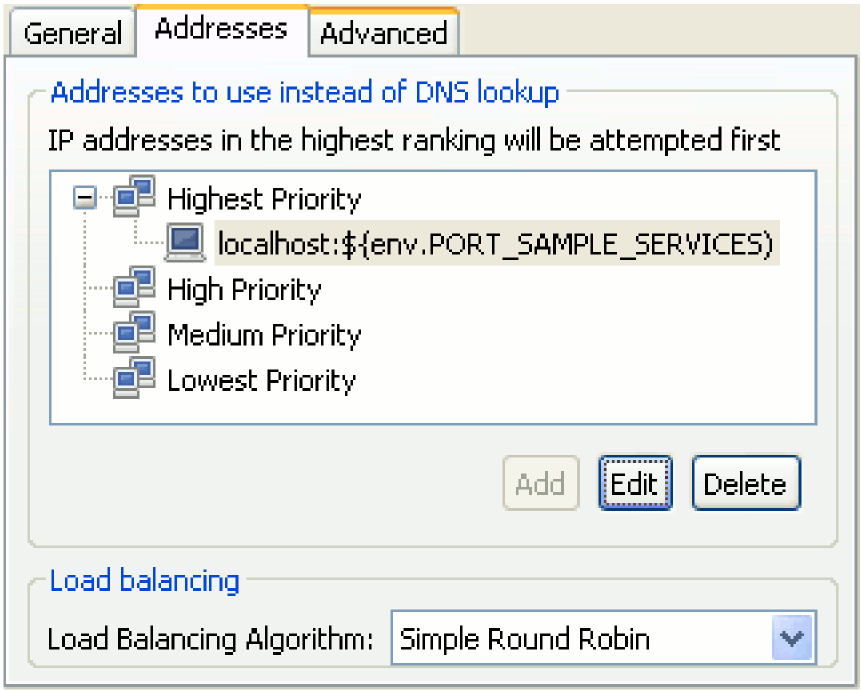
☑ Verify server's certificate matches requested hostname

On the **General** tab, the Remote Host is set to:

- Use HTTP 1.1.
- Use port `80` by default.
- Include the `ContentLength` header in the request to the back-end server.
- In case of an SSL connection, verify the Distinguished Name (DN) in the certificate presented by the server against the server's hostname.

Remote Host Address Settings

On the **Addresses** tab, the Remote Host is set to send requests to `localhost:${env.PORT_SAMPLE_SERVICES}`, which resolves to `localhost:8081` by default. You could also specify several servers in the **Addresses** list, and the API Gateway would load balance the requests across servers in the same group using the specified algorithm.



IP addresses in the highest ranking will be attempted first

- Highest Priority
  - localhost:${env.PORT_SAMPLE_SERVICES}
- High Priority
- Medium Priority
- Lowest Priority

Load Balancing Algorithm: Simple Round Robin

For more details on these settings, see *Remote Host Settings*.

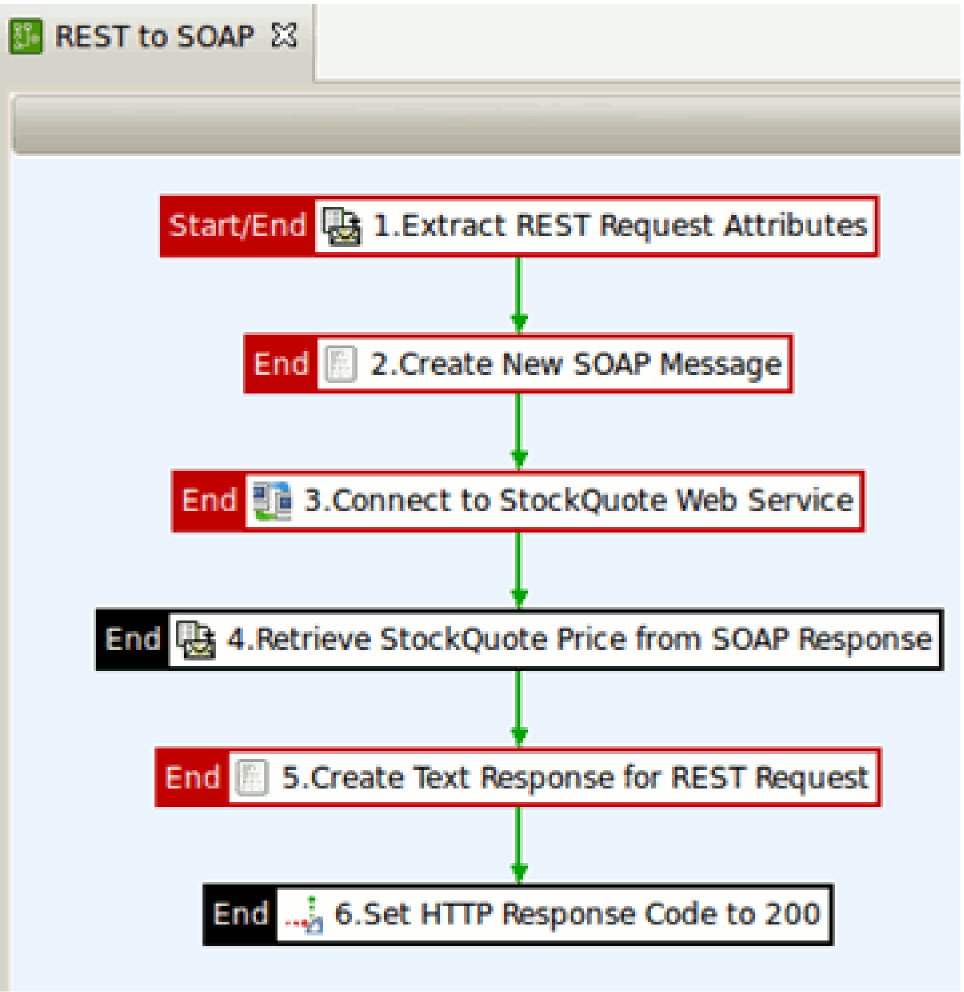
Conversion Sample Policy

Overview

The Conversion sample policy takes a REST-style request and converts it into a SOAP call. This topic describes the **REST to SOAP** policy, and explains how to run this sample.

REST to SOAP Policy

The **REST to SOAP** policy is as follows:


REST to SOAP
Start/End
1.Extract REST Request Attributes
End
2.Create New SOAP Message
End
3.Connect to StockQuote Web Service
End
4.Retrieve StockQuote Price from SOAP Response
End
5.Create Text Response for REST Request
End
6.Set HTTP Response Code to 200

The **REST to SOAP** policy performs the following tasks:

1. Extracts the information from the request (a message attribute is created for each query string and/or HTTP header).
2. Creates a SOAP message using the **Set Message** filter.
3. Sends the request to the `StockQuote` demo service.
4. Extracts the value of the stock from the response using XPath.
5. Creates a plain text response.

6. Sets the HTTP status code to 200.

Running the Conversion Sample

You can call the sample service using the Send Request (`sr`) command or the API Gateway Explorer GUI:

SR Command
Enter the following command:

```
sr http://HOSTNAME:8081/rest2soap?symbol=ABC
```

For more details, see the topic on *Stress Testing with Send Request (SR)*.

API Gateway Explorer
Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/rest2soap?symbol=ABC
   ```

2. Select `GET` as the verb.
3. Click the **Run** button.

For more details, see the topic on *Sending a Request with API Gateway Explorer*.

Security Sample Policies

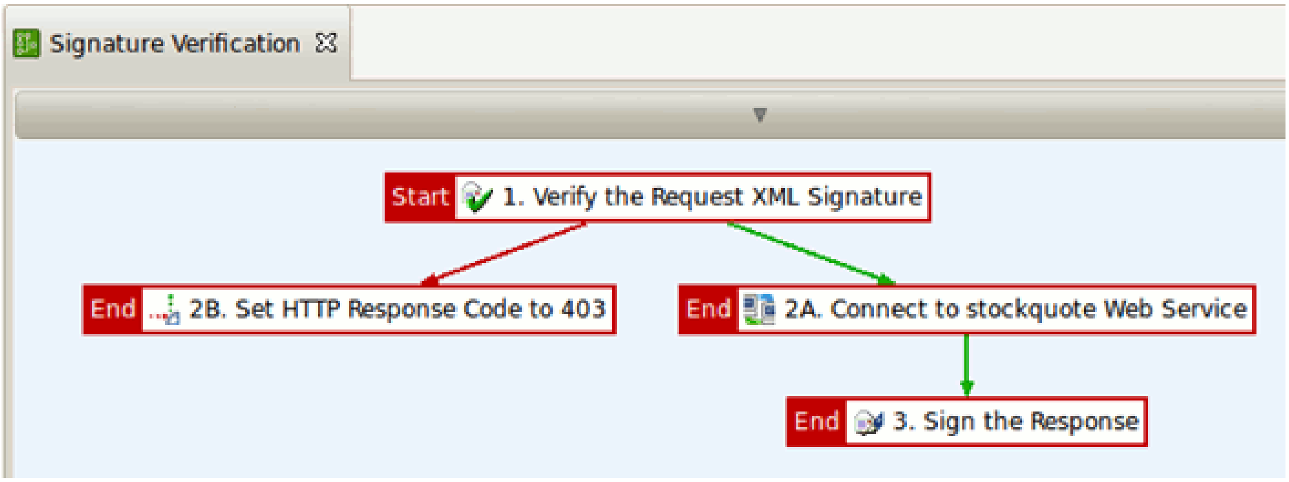
Overview

The security sample policies demonstrate digital signature verification and cryptographic operations (encryption and decryption). This topic describes the sample policies, and explains how to run these samples.

Signature Verification

The Signature Verification sample policy sends a digitally signed version of the `StockQuote` request to the API Gateway. The message carries the signature into the Web Service header. A sample certificate/key pair (`Samples Test Certificate`) is used to sign the message and verify the signature. Signature verification is used for authentication purposes, and therefore an HTTP 403 error code is returned if a problem occurs.

Signature Verification Policy
The **Signature Verification** policy is as follows:


Signature Verification
Start
1. Verify the Request XML Signature
End
2B. Set HTTP Response Code to 403
End
2A. Connect to stockquote Web Service
End
3. Sign the Response

The **Signature Verification** policy performs the following tasks:

1. The signature contained in the request is verified. The signature must be located in a WS Security block.
2. If the verification is successful, the `StockQuote` demo service is invoked.
3. The response body is signed and returned to the client.
4. If the verification fails, an HTTP 403 error code is returned to the client.

Running the Signature Verification Sample
You can call the sample service using the Send Request (`sr`) command or the API Gateway Explorer GUI:

SR Command
Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/Security/SignatureVerification/Request.xml
http://localhost:8081/signatureverification
```

For more details, see the topic on *Stress Testing with Send Request (SR)*.

API Gateway Explorer

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://hostname:8081/signatureverification
   ```

2. Select POST as the **Verb**.
3. Click the **Close** button.
4. Select **File** -> **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/Security/SignatureVerification/Request.xml
   ```

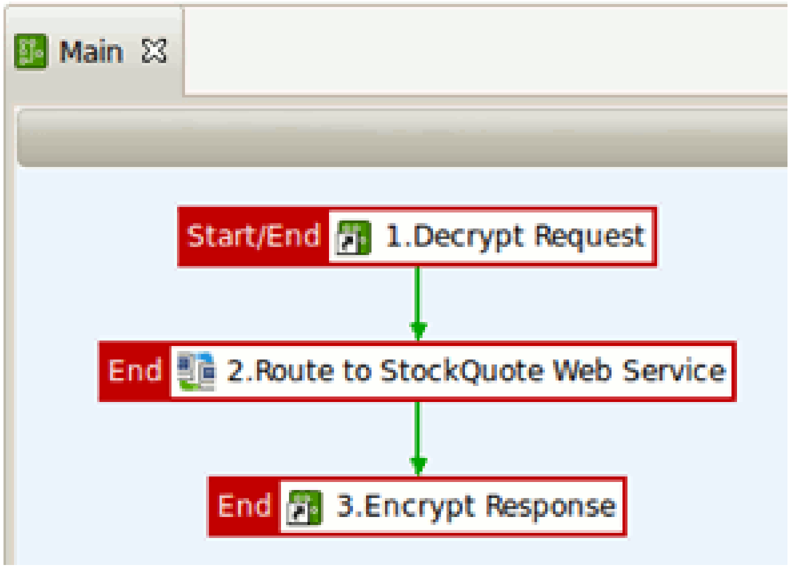
5. Click the Send Request button.

For more details, see the topic on *Sending a Request with API Gateway Explorer*.

Encryption and Decryption

This sample uses XML decryption on the request and applies encryption on the response. The sample policy includes a **Main** policy, which chains together the calls that decrypt the request, the invocation of the back-end service, and the encryption of the response.

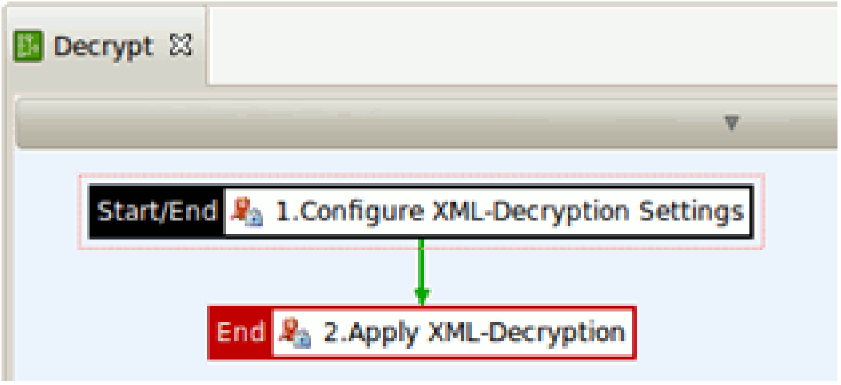
Main Policy
The **Main** policy is as follows:



The **Main** policy performs the following tasks:

1. **Decrypt Request** is a policy shortcut, which invokes another policy that takes the inbound request and decrypts it.
2. The decrypted request is routed to the back-end service.
3. The **Encrypt Response** policy shortcut invokes a policy that encrypts the response from the back-end service.

Decrypt Policy
The **Decrypt** policy is as follows:

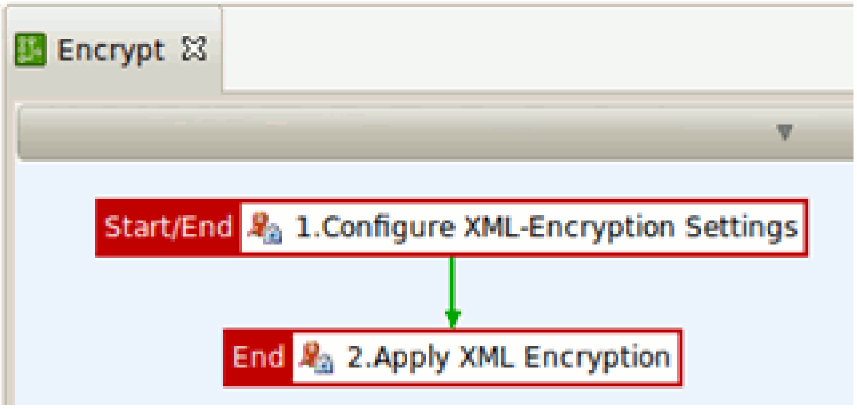


The **Encrypt** policy performs the following tasks:

1. The decryption settings are defined: what to decrypt and which key to use.
2. The XML decryption is executed based on the defined settings.

Encrypt Policy

The **Encrypt** policy is as follows:



The **Encrypt** policy performs the following tasks:

1. The encryption settings are defined: what to encrypt, which symmetric key to use, which certificate to use, and how to encrypt (algorithm and where to place the encryption information).
2. The XML encryption is executed based on the defined settings.

Running the Encryption and Decryption Sample

You can call the sample service using the Send Request (`sr`) command or the API Gateway Explorer GUI:

SR Command

Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/Security/Encryption/Request.xml
http://HOSTNAME:8081/encryption
```

For more details, see the topic on *Stress Testing with Send Request (SR)*.

API Gateway Explorer
Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/encryption
   ```

2. Select `POST` as the **Verb**.
3. Click the **Close** button.
4. Select **File** -> **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/Security/Encryption/Request.xml
   ```

5. Click the Send Request button.

For more details, see the topic on *Sending a Request with API Gateway Explorer*.

Throttling Sample Policy

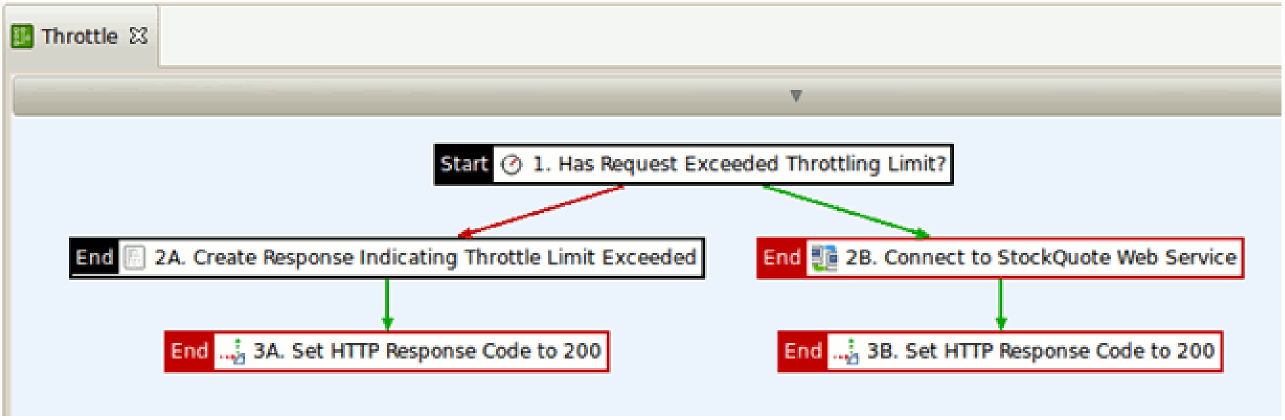
Overview

The Throttling sample policy is used to limit the number of calls for a service. This topic describes the **Throttle** policy, and explains how to run this sample.

Throttling refers to restricting incoming connections and the number of messages to be processed. It can be applied to XML, SOAP, REST, or any payload, request, or protocol. Traffic can be regulated for a single API Gateway or for a cluster of API Gateways. You can apply traffic restrictions rules for a service, an operation, or even time of day. For example, these restrictions can be applied depending on the service name, user identity, IP address, content from the payload, protocol headers, and so on.

Throttling Policy

The **Throttle** policy is as follows:


Throttle
Start
1. Has Request Exceeded Throttling Limit?
End
2A. Create Response Indicating Throttle Limit Exceeded
End
2B. Connect to StockQuote Web Service
End
3A. Set HTTP Response Code to 200
End
3B. Set HTTP Response Code to 200

The **Throttle** policy performs the following tasks:

1. The first filter checks whether the limit has been reached. The limit is set to 3 requests per 15 sec. The caller's IP address is used to track the consumer ID. The counter is kept in a local cache.
2. If the limit has been reached, an error message is created, and the response status code is set to 500.
3. If the authorized limit has not been reached, the back-end service is invoked, and the HTTP status code is set to 200.

Running the Throttling Sample

You can call the sample service using the Send Request (`sr`) command or the API Gateway Explorer GUI:

SR Command
Enter the following command:

```
sr -f
INSTALL_DIR/samples/SamplePolicies/Throttling/Request.xml http://HOSTNAME:8081/throttle
```

For more details, see the topic on *Stress Testing with Send Request (SR)*.

API Gateway Explorer

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/throttle
   ```

2. Select `POST` as the verb.
3. Click the **Close** button.
4. Select **File** -> **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/Throttling/Request.xml
   ```

5. Click the Send Request button.

For more details, see the topic on *Sending a Request with API Gateway Explorer*.

Virtualized Service Sample Policy

Overview

The Virtualized Service sample policy is more advanced and combines the following features:

- Content filtering, XML complexity, and message size filters to block unwanted SOAP messages.
- Content filtering to block unwanted REST requests.
- Fault handling.
- Content-based routing.

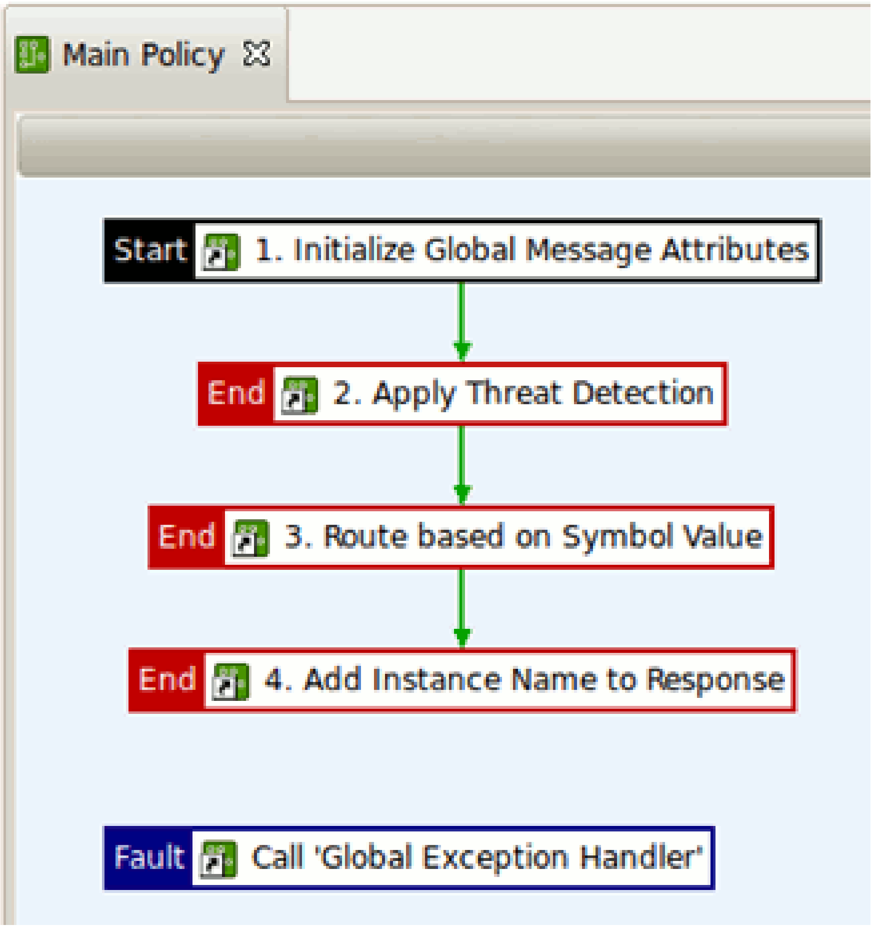
This topic describes the policies displayed in the **Sample Policies** -> **Web Services** -> **Virtualized StockQuote Service** policy container in the Policy Studio, and explains how to run this sample.

Virtualized Service policies

The **Virtualized StockQuote Service** sample policy container includes the following policies:

- Virtualized service main policy
- Threat protection policy
- Content-based routing policies
- Response transformation policy

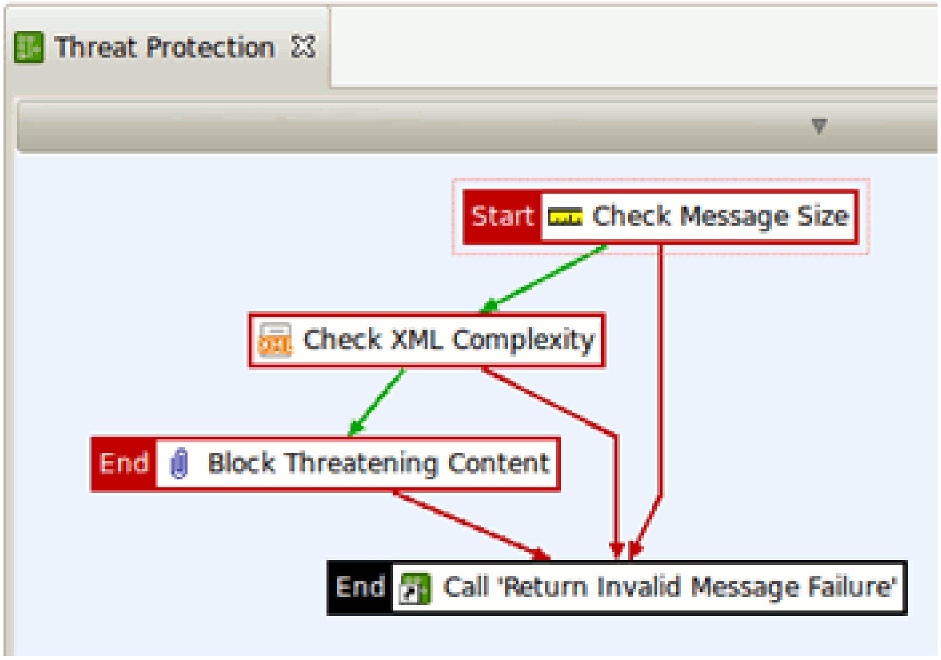
Virtualized Service Main Policy
The **Main Policy** is as follows:


Main Policy
Start
1. Initialize Global Message Attributes
End
2. Apply Threat Detection
End
3. Route based on Symbol Value
End
4. Add Instance Name to Response
Fault
Call 'Global Exception Handler'

The **Main Policy** uses policy shortcuts to perform the following tasks:

1. The main fault handler relies on some variables to be initialized, which is performed as soon as the policy is entered.
2. The **Threat Detection** policy is applied to the incoming SOAP message and HTTP headers.
3. The symbol value is extracted from the incoming message, and used to decide whether the request should be sent to one server instance or another.
4. The name of the instance that served the request is added to the response.
5. In case of errors, a global fault handler is invoked. This is used to return a custom error message to the user.

Threat Protection Policy
The **Threat Protection** policy is as follows:



The **Threat Protection** policy performs the following tasks:

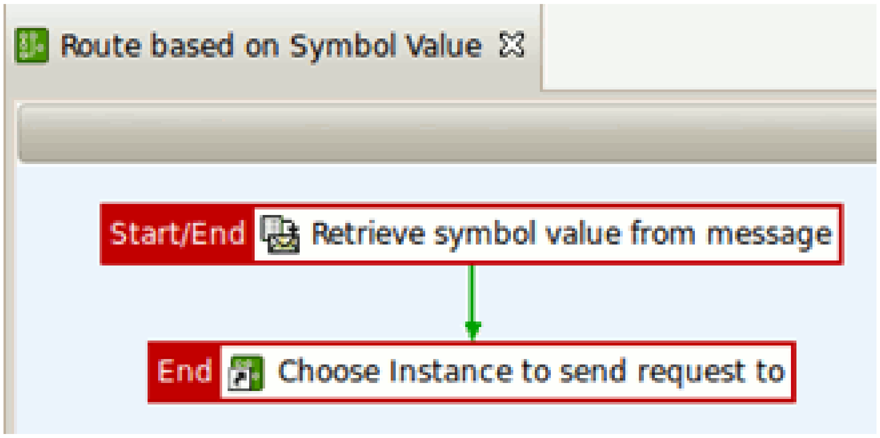
1. The incoming request size (including attachments) is checked to be less than 1500 KB.
2. The complexity of the XML is checked in terms of number of nodes, attributes per node, or number of child nodes per node.
3. XML and eventually HTTP headers are checked for threatening content such as SQL injection or XML processing instructions.
4. If any of these filters return an error, the corresponding error handler is called. The error handler is implemented as a policy that sets the value of the error code and message for this error, and then re-throws the exception so that the global fault handler catches it.

Content-based Routing Policies

The **Route Based on Symbol Value** policy extracts the contents of the symbol XML node and checks whether the first letter's value is between `A-L` or `K-Z`. Depending on the result, it routes the request to the first or second instance of the `StockQuote` server. These servers are simulated by the following Relative Path URIs defined in the API Gateway:

- `/stockquote/instance1`
- `/stockquote/instance2`

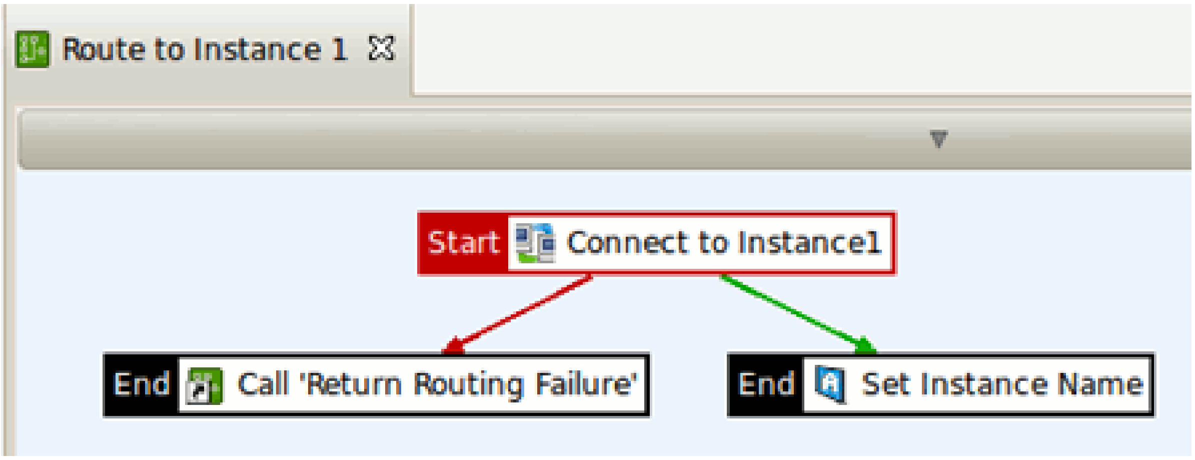
The **Route Based on Symbol Value** policy is as follows:


Route based on Symbol Value
Start/End
Retrieve symbol value from message
End
Choose Instance to send request to

The **Route Based on Symbol Node** policy performs the following tasks:

1. The value of the symbol node is extracted from the request using XPath. The result is placed in a message attribute named `message.symbol.value`.
2. A **Switch on attribute value** filter is used to check the value of the message attribute (using a regular expression), and a different policy is called to send the request to `instance1` or `instance2`.

The **Route to Instance1** policy is as follows:



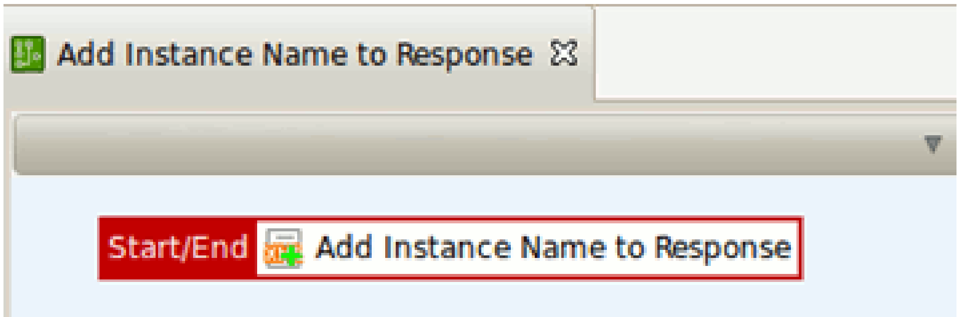
The **Route to Instance1** policy (called from the Switch filter) performs the following tasks:

1. Connects to the `instance1` URI .
2. If successful, the instance name (`instance1`) is placed in a message attribute (`stockquote.instance.name`). This is used later on to insert the instance name into the response.

The **Route to Instance2** policy performs the same tasks but using the `instance2` URI instead.

Response Transformation Policy

When the response is obtained from the back-end server, the **Add Instance Name to Response** policy changes it to insert the instance name into a new XML node (`instanceName`). The **Add Instance Name to Response** policy is as follows:



This policy adds the instance name (the value of the `stockquote.message.name` message attribute) to the response, using an **Add XML node** filter, as part of the `SOAPbody`. XPath is used to define where the new node must be added.

Running the Virtualized Service Sample

You can call the sample service using the Send Request (`sr`) command or the API Gateway Explorer GUI:

SR Command
Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
http://HOSTNAME:8081/main/stockquote
```

For more details, see the topic on *Stress Testing with Send Request (SR)*.

API Gateway Explorer
Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/main/stockquote
   ```

2. Select `POST` as the verb.
3. Click the **Close** button.
4. Select **File** -> **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
   ```

5. Click the Send Request button.

For more details, see the topic on *Sending a Request with API Gateway Explorer*.

Stress Testing with Send Request (SR)

Overview

The API Gateway provides a command-line tool for stress testing named Send Request (SR). The SR tool is available in the following directory of your API Gateway installation:

Windows	`INSTALL_DIR\Win32\lib`
Solaris/Linux	`INSTALL_DIR/posix/lib`
64-bit Linux	`INSTALL_DIR/Linux.x86_64/bin`

The SR tool is also available from the root directory of the API Gateway Explorer installation.

Important

On Linux, the `LD_LIBRARY_PATH` environment variable must be set to the directory from which you are running the SR tool.

On Linux and Solaris, you must use the `vrun sr` command. For example:

```
vrun sr http://testhost:8080/stockquote
```

Basic SR Examples

The following are some basic examples of using the SR command:

HTTP GET:

```
sr http://testhost:8080/stockquote
```

POST file contents (content-type inferred from file extension):

```
sr -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send XML file with SOAP Action 10 times:

```
sr -c 10 -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send XML file with SOAP Action 10 times in 3 parallel clients:

```
sr -c 10 -p 3 -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send the same request quietly:

```
sr -c 10 -p 3 -qq -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Run test for 10 seconds:

```
sr -d 10 -qq -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

POST file contents with SOAP Action:

```
sr -f StockQuoteRequest.xml -A SOAPAction:getPrice http://testhost:8080/stockquote
```

Advanced SR Examples

The following are some advanced examples of using the SR command:

Send form.xml to http://192.168.0.49:8080/healthcheck split at 171 character size, and trickle 200 millisecond delay between each send with a 200 Content-Length header:

```
sr -h 192.168.0.49 -s 8080 -u /healthcheck -b 171 -t 200 -f form.xml
-a "Content-Type:application/x-www-form-urlenprogramlistingd" -a "Content-Length:200"
```

Send a multipart message to http://192.168.0.19:8080/test, 2 XML docs are attached to message:

```
sr -h 192.168.0.49 -s 8080 -u /test -{ -a Content-Type:text/xml -f soap.txt
-a Content-Type:text/xml -f attachment.xml -a Content-Type:text/xml -} -A c-timestamp:1234
```

Send only headers using a GET over one-way SSL running 10 parallel threads for 86400 seconds (1 day) using super quiet mode:

```
sr -h 192.168.0.54 -C -s 8443 -u /nextgen -f test_req.xml -a givenName:SHViZXJ0
-a sn:RmFuc3dvcnRo -v GET -p10 -d86400 -qq
```

Send query string over mutual SSL presenting client certificate and key doing a GET running 10 parallel threads for 86400 seconds (1 day) using super quiet mode:

```
sr -h 192.168.0.54 -C -s 8443 -X client.pem -K client.key
-u "https://localhost:8443/idp?TargetResource=http://vordel.test.com" -f test_req.xml
-v GET -p10 -d86400 -qq
```

Send zip file in users home directory to testhost on port 8080 with /zip URI, save the resulting response content into the result.zip file, and do this silently:

```
sr -f ~/test.zip -h testhost -s 8080 -u /zip -a Content-Type:application/zip
-J result.zip -qq
```

SR Arguments

The main arguments to the SR command include the following:

Argument	***Description***
`--help`	List all arguments
`-a attribute:value`	Set the HTTP request header (for example, `-a Content-Type:text/xml`)
`-c [request-count]`	Number of requests to send per process
`-d [seconds]`	Duration to run test for

Argument	***Description***
`-f [content-filename]`	File to send as the request
`-h [host]`	Name of destination host
`-i [filename]`	Destination of statistics data
`-l [file]`	Destination of diagnostic logging
`-m`	Recycle SSL sessions (use multiple times)
`-n`	Enable nagle algorithm for transmission
`-o [output]`	Output statistics information every [milliseconds] (only with `-d`)
`-p [connections]`	Number of parallel client connections (threads) to simulate
`-q , -qq, -qqq`	Quiet modes (quiet, very quiet, very very quiet)
`-r`	Do not send HTTP Request line
`-s [service]`	Port or service name of destination (default is `8080`)
`-t [milliseconds]`	Trickle: delay between sending each character
`-u [uri]`	Target URI to place in request
`-v [verb]`	Set the HTTP verb to use in the request (default is `POST`)
`-w [milliseconds]`	Wait for [milliseconds] between each request
`-x [chunksize]`	Chunk-encode output
`-y [cipherlist]`	SSL ciphers to use (see OpenSSL manpage ciphers(1))
`-z`	Randomize chunk sizes up to limit set by `-x`
`-A attribute:value`	Set the HTTP request header (for example, `-a Content-Type:text/xml`) in the outermost attachment
`-B`	Buckets for response-time samples
`-C`	enCrypt (use SSL protocol)
`-I [filename]`	File for Input (received) data (`- = stdout`)
`-K`	RSA Private Key
`-L`	Line-buffer `stdout` and `stderr`
`-M`	Multiplier for response-time samples
`-N`	origiN for response-time samples
`-O [filename]`	File for Output (sent) data (`- = stdout`)
`-S [part-id]`	Start-part for multipart message
`-U [count]`	reUse each connection for `count` requests
`-V [version]`	Sets the HTTP version (`1.0`, `1.1`)
`-X`	X.509 client certificate
`-Y [cipherlist]`	Show expanded form of `[cipherlist]`
`[-{/-}`	Create multipart body (nestable: use `-f` for leaves)

Further Information

For a listing of all arguments, enter `sr --help`. For more information, and details on advanced use, see the `srman-page.pdf` file in your `sr` installation directory.

Sending a Request with API Gateway Explorer

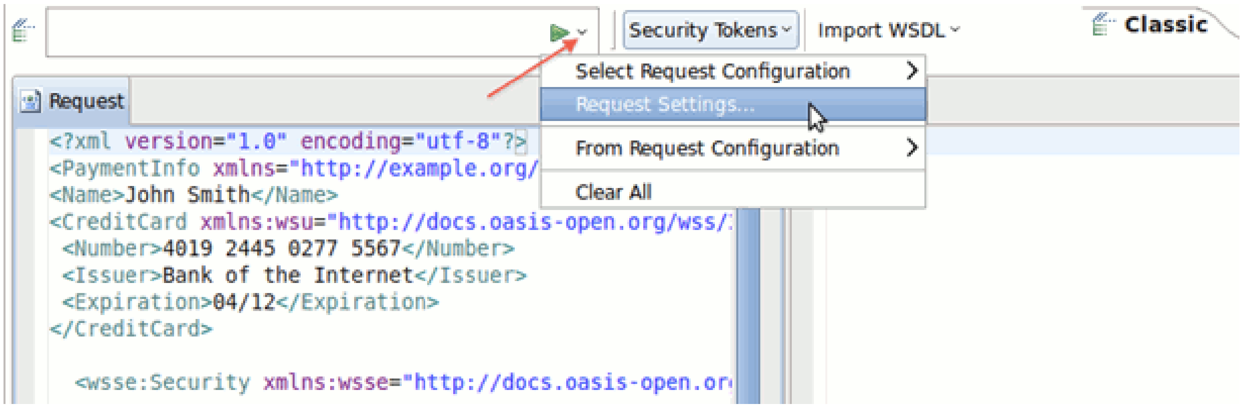
Overview

This topic describes how to create and send a request in the API Gateway Explorer test client GUI. You can start API Gateway Explorer using the `apigatewayexplorer` command from the installation directory.

Creating a Request in API Gateway Explorer

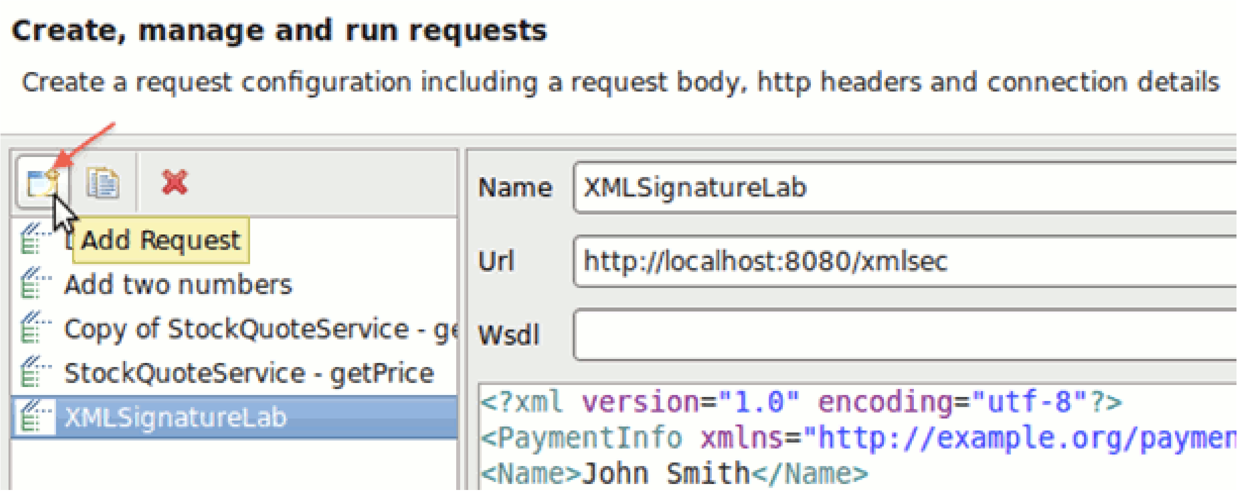
To create a request, perform the following steps:

1. Click the down arrow button beside the green triangular Send Request button in the toolbar, and select **Request Settings**:



```
<?xml version="1.0" encoding="utf-8"?>
<PaymentInfo xmlns="http://example.org/
<Name>John Smith</Name>
<CreditCard xmlns:wsu="http://docs.oasis-open.org/wss/
 <Number>4019 2445 0277 5567</Number>
 <Issuer>Bank of the Internet</Issuer>
 <Expiration>04/12</Expiration>
</CreditCard>

  <wsse:Security xmlns:wsse="http://docs.oasis-open.or
```

2. In the **Request Settings** dialog, click the **Add Request** button on the left in the toolbar:


Create, manage and run requests
Create a request configuration including a request body, http headers and connection details
Add Request
Add two numbers
Copy of StockQuoteService - ge
StockQuoteService - getPrice
XMLSignatureLab

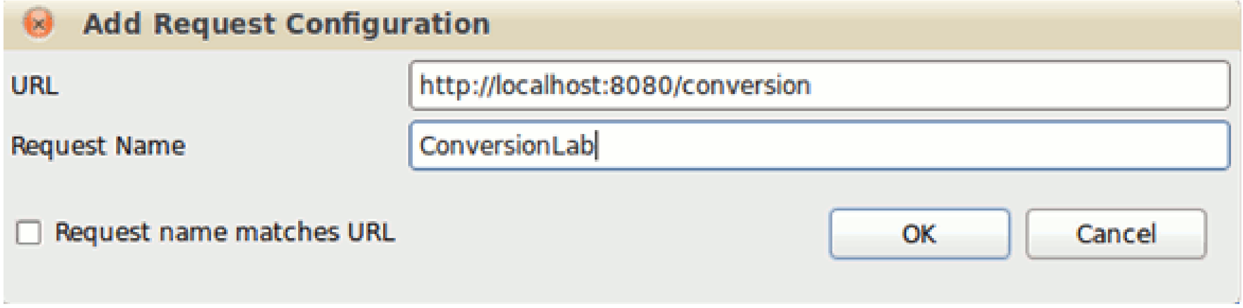
Name: XMLSignatureLab

Url: http://localhost:8080/xmlsec

Wsdl:

```
<?xml version="1.0" encoding="utf-8"?>
<PaymentInfo xmlns="http://example.org/paymen
<Name>John Smith</Name>
```

3. Enter the details for the request that you wish to execute in the **Add Request Configuration** dialog (for example: `http://localhost:8080/conversion`). If the **Request name matches URL** setting is not selected, you can supply a custom **Request Name** for this request.



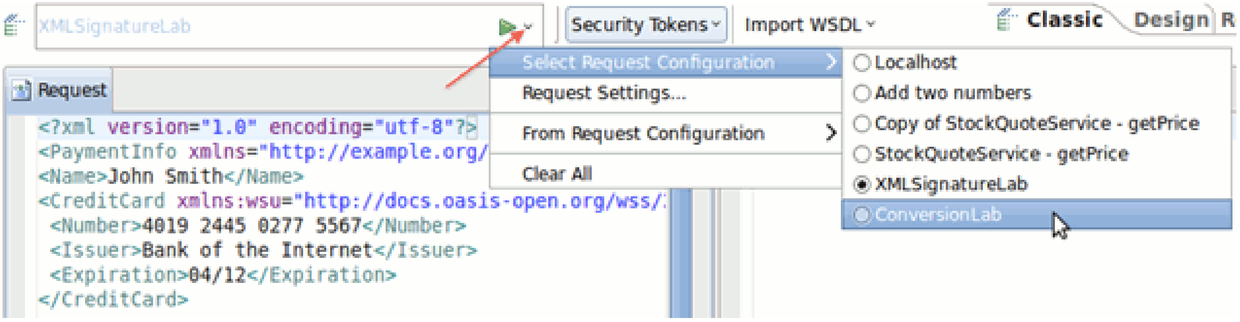
Add Request Configuration

URL: http://localhost:8080/conversion

Request Name: ConversionLab

□ Request name matches URL

4. Click **OK** to save the request configuration.
5. Select the request that you created in the **Select Request Configuration** menu:



```
<?xml version="1.0" encoding="utf-8"?>
<PaymentInfo xmlns="http://example.org/
<Name>John Smith</Name>
<CreditCard xmlns:wsu="http://docs.oasis-open.org/wss/
 <Number>4019 2445 0277 5567</Number>
 <Issuer>Bank of the Internet</Issuer>
 <Expiration>04/12</Expiration>
</CreditCard>
```

6. In the main menu, select **File** -> **Load Request**, and browse to the file that you wish to use as input for this request. For example, you can select the following file for the Virtualized Service sample:

```
INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
```

7. Click the green triangular Send Request button in the toolbar to send the request.

Further Information

For more details on using the API Gateway Explorer client GUI tool, see the API Gateway Explorer User Guide.

Introduction to API Service Manager

Overview

API Service Manager is a web-based system administration tool available in API Gateway Manager. It provides quick and easy access to enable you to manage your services and policies online. For example, you can perform tasks such as the following:

- *Managing API Services*
- Deploying to a Group
- Resetting Configuration

Accessing API Service Manager

You can access API Service Manager from the web-based API Gateway Manager tool. For more details, see the section called "Launching API Gateway Manager".

Creating your API Service Manager Workspace
When you have successfully authenticated in API Gateway Manager, you must first create a workspace for your API Gateway Manager working configuration:

1. Click the **API Service Manager** button in the API Gateway Manager toolbar.
2. Select the group and API Gateway instance that you wish to use (for example, `Group1` and `APIServer1`).
3. Click the **Create** button to create the workspace.
4. Enter the configuration passphrase (if one exists for this API Gateway instance), and click **OK** This creates a working copy of the selected configuration in your workspace, and displays the main **API Services** tab.

You can have zero or one working configuration in your workspace at a time, and you can deploy a configuration at any time to a group or a subset of API Gateways in a group. You can also reset the working configuration, or select a different one from the list of available API Gateway groups and instances.

Note

Session management (login and logout) is tracked in the API Gateway Manager session. When you log out of API Gateway Manager, your session expires. However, any updates that you have not yet deployed in API Service Manager remain unchanged in your working configuration. This configuration is persisted in your workspace and is available at your next login.

Deploying to a Group

To deploy configuration updates to a group of API Gateways or a subset of API Gateways in a group, perform the following steps:

1. Click **Actions** -> **Deploy** on the left in the **API Services** tab.
2. In the **Deployment Wizard**, select the group and API Gateway instance(s) to which you wish to deploy the current working configuration, and click the **Next**.
3. Enter a comment for this deployment (for example, `registering google search service`).
4. Click **Deploy**.
5. Click **Finish**.

Resetting your Configuration

To reset the current working configuration in API Service Manager, click **Actions** -> **Discard** on the left in the **API Services** tab, and click **OK**. This brings you back to the list of available API Gateway configurations, where you can select another configuration to create a new workspace.

Resetting the configuration resets the API Service Manager working configuration back to the currently deployed API Gateway configuration. This enables you to discard any undeployed configuration updates, in your workspace, and to obtain the latest deployed configuration updates from the Admin Node Manager.

Managing API Services

Overview

You can use the **API Services** tab in API Service Manager to virtualize services with the API Gateway. The Business Services repository stores service URLs, definitions and related information such as XML schemas. Clients can query this repository for service information (for example, URLs or WSDL files), and use it to send messages to the service through the API Gateway.

When you virtualize a service in the repository, the API Gateway exposes a *virtualized* version of the service. The host and port for the service are changed dynamically to point to the machine running the API Gateway. For example, the client can then send a message to the virtualized service through the API Gateway, without knowing its real location. Some policies are also automatically generated for the virtualized service. These include resolvers and connection filters, and are hidden by default in API Service Manager.

You can also use the **API Services** tab to perform tasks such as assigning policy packages to a service, configuring monitoring options, and assigning tags to help identify a service. A *policy package* is a sequence of modular, reusable message filters, each of which processes a message in a particular way. For example, a typical policy might contain an authentication filter (WS-Security UserNameToken), followed by several content-based filters (Schema Validation, Threatening Content, and Message Size). If all configured filters run successfully, the message is routed on to the configured destination.

Virtualizing a Service in API Service Manager

If you have not already done so, you must first perform the following steps to create a workspace in API Service Manager:

1. Click the **API Service Manager** button in the API Gateway Manager toolbar.
2. Select the group and API Gateway instance that you wish to use to virtualize the service (for example, `Group1`) and `APIServer1`.
3. Click the **Create** button to create the workspace.
4. Enter the configuration passphrase if one exists for this API Gateway instance, and click **OK**.
5. Click the **Virtualize API Service** button on the left in the toolbar of the **API Services** tab to launch the **New API Service** wizard.

Step 1—Basic Information

The first step in the **New API Service** wizard enables you to virtualize a service with or without a Web Services Definition Language (WSDL) file.

Virtualizing a REST API-based Service
To virtualize a REST API-based service without a WSDL file, perform the following steps:

1. Click **No, my Service will be defined manually**, and enter the details for your service, for example:
   - **Name**: `MyService`
   - **Destination URL**: `http://www.example.com/my_service`
2. Click **Next** to specify how service is exposed.

Virtualizing a Web Service
To virtualize an example Web service using the API Service Manager, perform the following steps:

1. Click **Yes, I know a URL from which to get a WSDL**, and enter a URL in the **WSDL URL** field, for example:

```
http://localhost:7070/axis/services/urn:xmltoday-delayed-quotes?wsdl
```

2. Click **Next** to view a WSDL import summary.
3. Click **Next** to specify how the service is exposed.

Step 2—Service Exposure

The second step in the wizard enables you to specify how the service is exposed. Perform the following steps:

1. Enter or select the protocol. Defaults to `HTTP`. You can also click **Show Details** to view the default port address (`${env.PORT.TRAFFIC}` defaults to `8080`).
2. Enter or select the services group. Defaults to `Default Services`.
3. Enter the relative path. Defaults to the path after the service domain name (for example, `my_service`). You may wish to virtualize the service on a different relative path.
4. Click **Next**.

Step 3—Request Processing

The third step in the wizard enables you to specify policy packages used for request processing. (for example, an OAuth policy package for authentication. Perform the following steps:

1. Click the green plus icon, and select a policy package from the list.
2. Select whether this policy package is `Required` or **Optional**. Defaults to `Required`.
3. Click the **Edit Parameters** icon to specify any policy parameters (for example, the value of a message attribute selector such as `${http.request.uri}`).
4. Repeat these steps to add more request processing policy packages.
5. Click **Next** when finished.

Note

You can use the Policy Studio to create reusabe policy packages that can be applied to services in API Service Manager. For more details, see *Configuring Policy Packages*.

Step 4—Routing

The fourth step in the wizard enables you to specify policy packages used for routing (for example, JMS). Perform the following steps:

1. Click the green plus icon, and select a policy package from the list.
2. Select whether this policy package is `Required` or **Optional**. Defaults to `Required`.
3. Click the **Edit Parameters** icon to specify any policy parameters (for example, the value of a message attribute selector such as `${http.headers}`).
4. Repeat these steps to add more routing policy packages.
5. Click **Next** when finished.

Step 5—Response Processing

The fifth step in the wizard enables you to specify policy packages used for response processing (for example, a policy package that removes sensitive information such as credit card details from the message). Perform the following steps:

1. Click the green plus icon, and select a policy package from the list.
2. Select whether this policy package is `Required` or **Optional**. Defaults to `Required`.
3. Click the **Edit Parameters** icon to specify any policy parameters (for example, the value of a message attribute selector such as `${content.body}`).
4. Repeat these steps to add more repsonse processing policy packages.
5. Click **Next** when finished.

Step 6—Monitoring

The sixth step in the wizard enables you to select the following monitoring options for the service:

- **Monitor API Service usage:**
  Specifies whether to store message metrics for this service. This is selected by default.
- **Monitor API Service usage per client:**
  Specifies whether to generate reports monitoring which authenticated clients are calling which services. This is selected by default.
- **Monitor client usage:**
  If you want to generate reports on authenticated clients, but are not interested in which services they are calling, select this option and deselect **Monitoring service usage per client**.
- **Message Attribute:**
  Enter the message attribute to use to identify authenticated clients. The default `authentication.subject.id` attribute stores the identifier of the authenticated user (for example, the username or user's X.509 Distinguished Name).

Click **Next** when finished.

Step 7—Tags

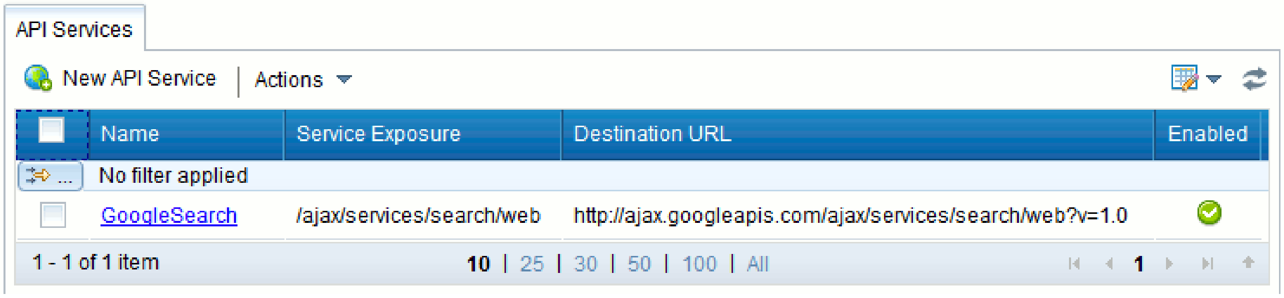
The final step in the wizard enables you to specify tags for this service. Tags are user-friendly names to help organize, search, and browse API Gateways and services in API Gateway Manager and Policy Studio. Perform the following steps:

1. Click the green plus icon to add a tag.
2. Enter a **Tag** name (for example, `Dept`).
3. Enter a **Value** (for example, `QA`).
4. Click **Finish**.

To view services by tag in API Gateway Manager, perform the following steps:

1. Click the **Show Columns** button on the right in the **API Services** toolbar.
2. Select the tag that you wish to display.
3. Click **Apply** to view tag in the list.

The virtualized service is displayed on the **API Services** tab:



□	Name	Service Exposure	Destination URL	Enabled
□	GoogleSearch	/ajax/services/search/web	http://ajax.googleapis.com/ajax/services/search/web?v=1.0	✓

Deploying to a Group

When you have completed the steps in the wizard, you must deploy the updated configuration to a API Gateway group, or a subset of API Gateways in a group, as follows:

1. Click **Actions** -> **Deploy** on the left in the **API Services** tab.
2. In the **Deployment Wizard**, select the group and API Gateway instance(s) to which you wish to deploy the current working configuration, and click the **Next**.
3. Enter a comment for this deployment (for example, `registering google search service`).
4. Click **Deploy**.
5. Click **Finish**.

Note

Services virtualized using API Service Manager and deployed to the API Gateway can also be viewed in the Policy Studio tree under the **Business Services** -> **API Service Manager** node. For details on using Policy Studio to import services, see *Web Service Repository*.

Configuring Security Policies from WSDL Files

Overview

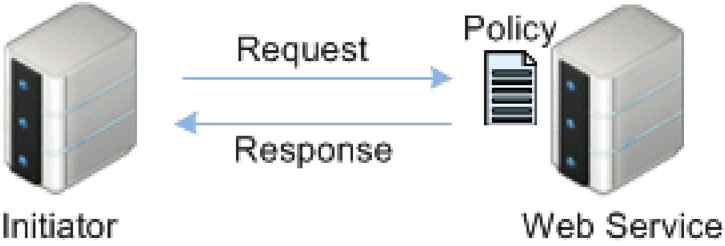
When you import a WSDL file into the Web Services Repository to virtualize and secure a protected Web Service, the Policy Studio automatically generates policies. For example, a **Service Handler** is created to control and validate requests to the Web Service and responses from the Web Service. The information used to configure the Service Handler is automatically taken from the operation definitions in the WSDL.

When clients must use message-level or transport-level security mechanisms to communicate with the Web Service, you can include WS-Policy assertions in the WSDL. These policy assertions can then be referenced at the operation or endpoint level in the WSDL. For example, a given Web Service may require the client to sign sensitive parts of the message and include a WS-Security UsernameToken to authenticate to the Web Service. You can include these policy requirements in the WSDL. Whenever a client retrieves the WSDL, it can automatically sign the relevant parts of the message and insert a valid UsernameToken.

When you import a WSDL file containing WS-Policy assertions into the Web Services Repository, you can select the operations that you want to protect as normal in the **Import WSDL** wizard. The **Secure Virtual Service** dialog enables you to specify the policy that the API Gateway enforces on the messages that it receives from a client. For more details, see *Securing a Virtual Service using Policies*.

In the **Policy Configuration Settings** wizard, you can configure specific filters to fulfill the security requirements specified by the policy assertions in the WSDL file. Most of these requirements are met without the need for human intervention. However, a small number of filters require the administrator to configure specific fields. For example, when signing or encrypting a message, you must specify the signing or encrypting key. When configuring the duration of a WS-Security Timestamp, you may need to specify longer or shorter than the default of one hour. However, most of the information required to configure these filters is set automatically based on the policy assertions in the WSDL file.

Using WS-Policy assertions, the Policy Studio can automatically generate complicated policies that can then be used to talk to the Web Services defined in the WSDL. The API Gateway then becomes the client or *initiator* of the Web Service, and is responsible for making sure the requests it sends to the service adhere to the security constraints specified in the policy:



In this way, administrators can configure complex policies to talk to secure Web Services with only a few clicks and minimal intervention. In addition, the API Gateway uses cryptographic acceleration to reduce the overhead associated with running the cryptographic operations required to secure the message.

Importing a WSDL File

When you import the WSDL for a Web Service into the Web Services Repository, the API Gateway exposes a *virtualized* version of this service. This involves changing the host and port where the Web Service is available to point to the machine running the API Gateway. In this way, a client can retrieve the WSDL for the virtualized Web Service from the API Gateway without knowing its real location.

To import the WSDL file into the Web Service Repository, complete the following steps:

1. In the Policy Studio tree view, expand the **Business Services** -> **Web Services Repository** node, and select the **Web Services** node.
2. Right-click the **Web Services** node, and select **Register Web Service**.
3. In the **Load WSDL** screen, browse to the location of the WSDL in the file system, enter the URL of the WSDL, or retrieve it from a UDDI (Universal Description, Discovery, and Integration) repository. Select as appropriate, and click **Next**.
4. The operations defined in the WSDL and exposed by the Web Service are listed on the **WSDL Operations** screen. Select the operations that you want to secure, and click **Next**.
5. You can also expose only the operations selected on the **WSDL Operations** screen in a slimmed down version of the Web Service. Select **Remove unselected operations from the WSDL** to remove operations that you do not want to secure from the virtualized service that is exposed to clients. Click **Next** to continue.
6. On the **WS-Policy Options** screen, if you wish to use a WS-Policy to secure the Web Service, select **Secure this virtualized service with a WS-Policy**. This means that the **Secure Virtual Service** dialog is then displayed after the **Import WSDL** wizard.
7. Ensure that **Use the WS-Policy in the WSDL to connect securely to the back-end Web Service** is selected. This is enabled (and selected by default) only if the selected WSDL file includes WS-Policy information. Click **Next** to continue.
8. Select the Relative Path where you want this service to be deployed (for example, **Default Services**).
9. Click **Finish**.

When you have completed the steps in the **Import WSDL** wizard, the **Secure Virtual Service** dialog is displayed. For details, see *Securing a Virtual Service using Policies*.

Configuring Policy Settings

Depending on the type and number of WS-Policy assertions in the WSDL, the **Policy Configuration Settings** wizard contains configuration screens for the filters used to implement the rules required by the assertions. The exact sequence of screens differs depending on the assertions specified in the WSDL.

For example, if an `sp:SamlToken` assertion is specified, the wizard contains a screen for the **Insert SAML Authentication Assertion** filter. Only certain fields must be specified by the administrator on this filter screen, while the rest are automatically populated based on the assertions and properties defined in the WSDL.

In the case of the **Sign Message** filter, the decision to use asymmetric or symmetric signatures is based on whether the policy uses an asymmetric or symmetric binding. The layout rules are determined by the `sp:Layout` assertion. The digest method, signature method, and key wrap algorithm (for symmetric signatures) are all populated automatically based on the contents of the `sp:AlgorithmSuite` assertion. The `KeyInfo` section of the XML Signature can be taken from various properties set in the WSDL. The parts of the message to be signed can be inferred from assertions such as `sp:SignedParts`, `sp:SignedElements`, and `SignedSupportingTokens`.

The same is true for the **XML Encryption Settings** filter where the encryption algorithms and key types can all be taken from the assertions in the WSDL. The `ConfirmationMethod` for SAML assertions can be inferred from the context of the `SamlToken` assertion. For example, an `sp:SamlToken` that appears as a child of the `sp:SignedSupportingTokens` assertion uses a `sender-vouches` confirmation method, whereas if it appears as a child of an `sp:EndorsingSupportingTokens` assertion, the `holder-of-key` confirmation method can be assumed.

In the **Policy Configuration Settings** wizard, if the API Gateway has also been configured as the *recipient* for the client, the **Configure Recipient Security Settings** screen is displayed first. For more details, see *Securing a Virtual Service using Policies*. The **Configure Initiator Security Settings** screen is displayed next. This enables you to specify the required filter settings when the API Gateway is configured as the initiator for the Web Service.

Configuring Policy Filters

The following tables list the types of filters that are created, and which fields must be completed by the administrator in the **Configure Initiator Security Settings** screen. For simplicity, the tables below list only the filters that require manual input from the administrator.

Insert WS-Security Timestamp

Field Name	*Description*
Expires In	You may want to specify a more appropriate lifetime for the assertion (instead of the default one hour) by configuring the various time period fields.

Sign Message

Field Name	*Description*
Signing Key	If the policy uses an asymmetric binding, on the **Asymmetric** tab, click the **Signing Key** button, and select a key from the Certificate Store to sign the message parts with. Alternatively, if the policy specifies a symmetric binding, on the **Symmetric** tab, click the **Signing Key** button, and select a key to wrap the symmetric signing key with.

Insert WS-Security Username

Field Name	*Description*
Username	Enter the username inserted into the WS-Security `UsernameToken` block. By default, the name of the authenticated user is used, which is stored in the `authentication.subject.id` message attribute. However, any user-specified value can be entered in this field.
Password	If the policy requires a password, the password for the user entered above must be specified here. You can use the default authenticated user password by selecting the `authentication.subject.password` message attribute. Alternatively, you can enter any suitable password manually entered if necessary. The decision to use a **Clear** or **Digest** password is taken from the corresponding policy assertions.

Insert SAML Authentication Assertion

Field Name	*Description*
Expire In	Specify a suitable lifetime for the SAML assertion by configuring the various time period fields.
Drift Time	You may need to specify a drift time value to allow for a time differential between the clock on the machine hosting the API Gateway and the machine hosting your Web Service.
Issuer Name	Select the alias of the certificate from the Certificate Store that you want to use to identify the issuer of the assertion.

	The alias name is used as the value of the `Issuer` attribute of the `saml:Assertion` element.
Holder of Key: Signing Key	In cases where the `sp:SamlToken` appears as a child of `EndorsingSupportingTokens` or an `InitiatorToken`, the `holder-of-key` SAML confirmation method is inferred. In this case, if an asymmetric binding is used, on the **Asymmetric** tab, specify a key from the Certificate Store by clicking the **Signing Key** button. Alternatively, if a symmetric binding is used in the policy, on the **Symmetric** tab, specify a key to use to encrypt the symmetric key with by clicking the **Signing Key** button.

Find Recipient Certificate for Encryption

Field Name	*Description*
Certificate Store	Select this option, and click the **Select** button to choose the recipient's certificate from the Certificate Store. The public key contained in this certificate is used to encrypt the message parts so that only the recipient is able to decrypt them using the corresponding private key.

Connect to URL

Field Name	*Description*
Trusted Certificates	To connect to an external Web Service over SSL, you need to trust that Web Service's SSL certificate. You can do this on the **Trusted Certificates** tab of the **Connect to URL** filter. Assuming you have already imported this certificate into the Trusted Certificate Store, simply select it from the list.
Client SSL Authentication	If the Web Service requires the client to present an SSL certificate to it during the SSL handshake, you must select that certificate from the list on the **Client SSL Authentication** tab. **Note** This certificate must have a private key associated with it that is also stored in the Certificate Store.

Extract MTOM Content

Field Name	*Description*

XPath Location	When the `wsoma:OptimizedMimeSerialization` WS-MTOMPolicy assertion is specified in a policy, you must configure an **Extract MTOM Content** filter. You need only configure an XPath expression to point to the base64-encoded element content that you want to extract and create an MTOM type attachment for.

Editing a Policy

You may wish to edit a previously configured WS-Policy (for example, to change the signing key in the auto-generated policy). You can do this by right-clicking the Web Service in the Policy Studio tree, and selecting **Configure Initiator WS-Policy** or **Configure Recipient WS-Policy**. These menu options are described as follows:

Configure Initiator WS-Policy:
If you have already configured an initiator WS-Policy, you can edit its filters using this menu option. However, if there was no WS-Policy in the imported WSDL file, you can not use this option. You can not add a WS-Policy to the Web Service because that would break the contract between the API Gateway and the back-end Web Service. If the contract for the Web Service changes (for example, a WS-Policy is applied to it at the back-end), you need to re-import the modified WSDL to reflect the changes.

Configure Recipient WS-Policy:
If a recipient WS-Policy was configured when the WSDL file was imported into the Web Service Repository, you can edit its filters using this option. If you did not configure a WS-Policy when importing the WSDL file (using the **Secure Virtual Service** dialog), when you select this option, the **Secure Virtual Service** dialog is displayed. This enables you to select a WS-policy to secure the service. The next time that you select the **Configure Recipient WS-Policy** option, you will edit this policy.

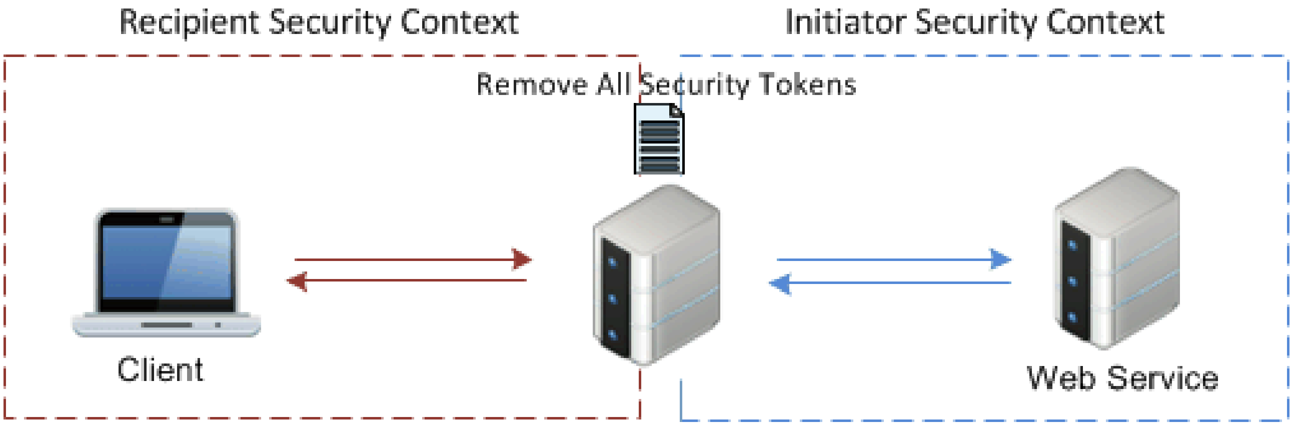
Removing Security Tokens

When you import a WSDL file containing a WS-Policy into the Web Service Repository, the **Remove All Security Tokens** filter is enabled in the **Service Handler** for the imported Web Service. You can view the configured policy by double clicking the **Service Handler**, and selecting the **Message Interception Points** -> **2. User-defined Request Hooks** tab.

The **Remove All Security Tokens** policy ensures that the following security contexts are kept separate:

- *Recipient security context*: This is between the client and the API Gateway, and is determined by the WS-Policy selected in the **Secure Virtual Service** dialog.
- *Initiator security context*: This is between the API Gateway and the back-end Web Service, and is determined by the WS-Policy contained in the imported WSDL for the back-end Web Service.

The **Remove All Security Tokens** policy prevents tokens from one context passing over into the other context, which could breach the security contract governing that context. This ensures that each security context receives a clean SOAP message, on which it can then act to enforce the security requirements of the relevant WS-Policy. The following diagram shows both security contexts and the **Remove All Security Tokens** policy:


Recipient Security Context
Initiator Security Context
Remove All Security Tokens
Client
Web Service

Initiator-side WS-Policy only
In this case, the WS-Policy is contained in the imported WSDL file. The WS-Policy defines the security contract between the API Gateway and the back-end Web Service defined in the WSDL. On the request side, any security tokens sent by the client to the API Gateway, which are out of scope of the initiator WS-Policy between the API Gateway and Web Service, are removed before the API Gateway starts enforcing the initiator WS-Policy on the request, and before it sends the request to the Web Service.

For example, if the client sends a `wsu:Timestamp` in the request message and the initiator policy stipulates that a `wsu:Timestamp` must be sent by the API Gateway to the Web Service, two timestamps could be sent in the request, which is invalid. This means that the timestamp and any other security tokens sent by the client to the API Gateway, which may contradict the rules in the initiator contract (between the API Gateway and Web Service) must be stripped out before the API Gateway starts adding security tokens to the message. This ensures that the message adheres to the initiator WS-Policy.

Similarly, any security tokens returned by the Web Service are only present because the Web Service complies with the contract between the Web Service and the API Gateway. Therefore, any tokens returned by the Web Service are only intended for use by the API Gateway. They are *not* intended for consumption by the client. In other words, the security context is only between the API Gateway and the Web Service. If the Web Service returns a `UsernameToken`, it is consumed by the API Gateway.

If a token must be returned to the client, this is a user-enforced rule, which is out of scope of the WS-Policy configuration in the WSDL. If necessary, you can override the default behavior by removing the **Remove All Security Tokens** filter from the **Service Handler** to allow the `UsernameToken` to be propagated to the client.

Initiator-side and Recipient-side WS-Policy
This occurs when you import a WSDL file that includes a WS-Policy (initiator case), and you also select a WS-Policy in the **Secure Virtual Service** dialog (recipient case). This scenario includes both the *recipient security context* between the client and the API Gateway, and the *initiator security context* between the API Gateway and the Web Service.

It is vital that these security contexts are kept separate because if tokens from one context pass over into the other context, it is highly likely that the security contract for that context will be breached. For example, if the recipient contract between the client and the API Gateway requires a `UsernameToken`, but the initiator contract between the API Gateway and the Web Service requires a SAML token, the `UsernameToken` must not pass over into the initiator context and be sent to the Web Service.

For more details on the **Secure Virtual Service** dialog (recipient case), see *Securing a Virtual Service using Policies*.

Important

The **Remove All Security Tokens** policy only applies when a WS-Policy is configured, and is *not* enabled when a WS-Policy is not configured. In addition, any non-standard behavior that requires a security token to be propagated over to another security context can be handled by disabling the **Remove All Security Tokens** policy in the **Service Handler** for the imported WSDL.

Further Information

For more details on configuring policies to protect your Web Services, see the following:

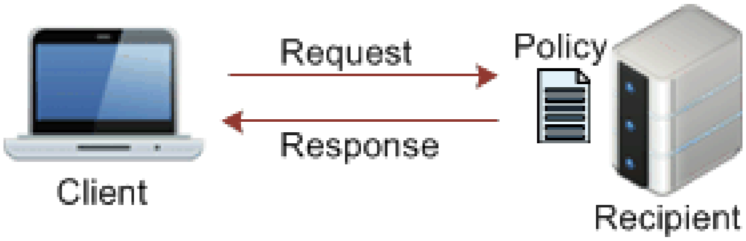
- *Securing a Virtual Service using Policies*
- *Configuring Policies Manually*
- *Web Service Filter*

The **Web Services** filter is the main filter generated when a WSDL file is imported into the Web Services Repository. It contains all the routing information and links all the policies that are to be run on the request and response messages into a logical flow.

Securing a Virtual Service using Policies

Overview

You can specify a WS-Policy to enforce security between a client and the API Gateway. In this deployment scenario, the API Gateway exhibits recipient-side WS-Policy behavior, where the API Gateway is the *recipient*, and the client is the *initiator*. The following architecture diagram shows where the recipient WS-Policy applies in a typical message flow between a client and the API Gateway:



When you import a WSDL file into the Web Services Repository, you can select the operations that you want to secure in the **Import WSDL** wizard. The **Secure Virtual Service** dialog then enables you to specify the policy that the API Gateway enforces on the messages that it receives from the client.

In the **Policy Configuration Settings** wizard, you can then configure specific fields in the filters that are necessary to fulfill the security requirements specified in the **Secure Virtual Service** dialog. Most of these requirements are met without the need for human intervention. However, a small number of filters require the administrator to configure specific fields. For example, when signing or encrypting a message, you must specify the signing or encrypting key. When configuring the duration of a WS-Security Timestamp, you may need to specify longer or shorter than the default of one hour. However, most of the information required to configure these filters is set automatically based on the selected WS-Policy.

Using policies in this way, the Policy Studio automatically generates the complicated policies that the API Gateway uses to talk to the client. The API Gateway then becomes the recipient of the client, and is responsible for enforcing the selected policies on the messages that it receives from the client. The main advantage is that administrators can configure complex policies to talk to clients in a secure manner with only a few clicks and minimal intervention.

Importing a WSDL File

When you import the WSDL for a Web Service into the Web Services Repository, the API Gateway exposes a *virtualized* version of this service. This involves changing the host and port where the Web Service is available to point to the machine running the API Gateway. In this way, a client can retrieve the WSDL for the virtualized Web Service from the API Gateway without knowing its real location.

To import the WSDL file into the Web Service Repository, complete the following steps:

1. In the Policy Studio tree view, expand the **Business Services** -> **Web Services Repository** node, and select the **Web Services** node.
2. Right-click the **Web Services** node, and select **Register Web Service**.
3. In the **Load WSDL** screen, browse to the location of the WSDL in the file system, enter the URL of the WSDL, or retrieve it from a UDDI (Universal Description, Discovery, and Integration) repository. Select as appropriate, and click **Next**.
4. The operations defined in the WSDL and exposed by the Web Service are listed on the **WSDL Operations** screen. Select the operations that you want to secure, and click **Next**.
5. You can also expose only the operations selected on the **WSDL Operations** screen in a slimmed down version of the Web Service. Select **Remove unselected operations from the WSDL** to remove operations that you do not want to secure from the virtualized service that is exposed to clients. Click **Next** to continue.

6. On the **WS-Policy Options** screen, select **Secure this virtualized service with a WS-Policy**. The means that the **Secure Virtual Service** dialog is displayed after the **Import WSDL** wizard.
7. If the WSDL file includes WS-Policy information, select **Use the WS-Policy in the WSDL to connect securely to the back-end Web Service**. Click **Next** to continue.
8. Select the Relative Path where you want this service to be deployed (for example, **Default Services**).
9. Click **Finish**.

When you have completed the steps in the **Import WSDL** wizard, the **Secure Virtual Service** dialog is displayed.

Configuring a Security Policy

The **Secure Virtual Service** dialog enables you to specify the policy that the API Gateway enforces on the messages that it receives from the client. To specify a policy, perform the following steps:

1. In the **Secure Virtual Service** dialog, in the **Security Policy** panel, select the **API Gateway Policy** from the drop-down list. A description for the currently selected policy is displayed in the dialog. For details on all the available policies, see the WS-Policy Reference.
2. In the **Message-Level Policy** panel, select a **Request Policy** from the drop-down list. The available policies are as follows:
   - Encrypt SOAP Body
   - Sign SOAP Body
   - Sign and Encrypt SOAP Body
3. Select a **Response Policy** from the drop-down list. The available policies are the same as for **Request Policy**.
4. Click **OK**.

The **Policy Configuration Settings** wizard is displayed. This enables you to set some of the fields in the filters that require human intervention (for example, the signing and encrypting key).

Configuring Policy Settings

Depending on the policy configured in the **Secure Virtual Service** dialog, the **Policy Configuration Settings** wizard displays configuration screens for the filters that implement the rules required by the configured policy. The exact sequence of screens differs depending on the policy that is selected.

For example, if a policy with a SAML token is selected, the **Validate SAML Authentication Assertion** filter is displayed instead of the **Validate WS-Security UsernameToken** filter. The effort in configuring these screens is minimal because the information is taken automatically from the WS-Policy assertions. For example, the layout, signing, encryption, and key wrapping algorithms, key referencing method, Username digest, and clear password are all automatically taken from the WS-Policy assertions. This means that the administrator has only to configure a small number of settings. For example, the signing key, encryption certificate, and Timestamp validity period).

The **Configure Recipient Security Settings** screen is first displayed in the wizard. This enables you to specify the required settings when the API Gateway is deployed as the recipient for the client. If your WSDL file includes WS-Policy assertions, the **Configure Initiator Security Settings** screen is then displayed in the wizard. This screen enables you to specify the required settings when the API Gateway is deployed as the initiator for the Web Service. For more details, see *Configuring Security Policies from WSDL Files*.

Configuring Policy Filters

The following tables show examples of the types of filters that are created, and which fields must be completed by the administrator in the **Configure Recipient Security Settings** screen. For simplicity, these tables list only filters that require manual input from the administrator.

Insert Timestamp Filter

Field Name	*Description*
Expires In	You may want to specify a more appropriate lifetime for the assertion (instead of the default one hour) by configuring the various time period fields.

Signed Parts Outbound Filter

Field Name	*Description*
Signing Key	If the policy uses an asymmetric binding, on the **Asymmetric** tab, click the **Signing Key** button, and select a key from the Certificate Store to sign the message parts with. Alternatively, if the policy specifies a symmetric binding, on the **Symmetric** tab, click the **Signing Key** button, and select a key to wrap the symmetric signing key with.

Find Recipient Certificate for Encryption

Field Name	*Description*
Certificate Store	Click the **Select** button to choose the recipient's certificate from the Certificate Store. The public key contained in this certificate is used to encrypt the message parts so that only the recipient is able to decrypt them using the corresponding private key.

Validate SAML Authentication Assertion

Field Name	*Description*
Drift Time	You may need to specify a drift time value to allow for a time differential between the clock on the machine hosting the API Gateway and the machine hosting your Web Service.
Trusted Issuers	On the **Trusted Issuers** tab, click **Add** to specify the Distinguished Name of a SAML Authority whose certificate has been added to the Certificate Store, and click **OK**. Repeat this step to add more SAML Authorities to the list of trusted issuers.

Configure SSL Certificate

Field Name	*Description*
X.509 Certificate	On the **Network** tab, click the **X.509 Certificate** button to create or import an SSL certificate.

SSL Server Name Identifier (SNI)	On the **Network** tab, click the **Add** button to configure a server name in the **SSL Server Name Identifier (SNI)** dialog. You can specify the server name in the **Client requests server name** field. Click the **Server assumes identity** button to import a Certificate Authority certificate into the Certificate Store.
Mutual Authentication	On the **Mutual Authentication** tab, select root Certificate Authorities trusted for mutual authentication.

Insert MTOM Content

Field Name	*Description*
XPath Location	When the `wsoma:OptimizedMimeSerialization` WS-MTOMPolicy assertion is specified in a policy, you must configure an **Insert MTOM Content** filter. You need only configure an XPath expression to point to the base64-encoded element content that you want to insert and create an MTOM type attachment for.

Editing a Security Policy

You may wish to edit a previously configured WS-Policy (for example, to change the signing key in the auto-generated policy). You can do this by right-clicking the Web Service in the Policy Studio tree, and selecting **Configure Initiator WS-Policy** or **Configure Recipient WS-Policy**. These menu options are described as follows:

Configure Initiator WS-Policy:
If you have already configured an initiator WS-Policy, you can edit its filters using this menu option. However, if there was no WS-Policy in the imported WSDL file, you can not use this option. You can not add a WS-Policy to the Web Service because that would break the contract between the API Gateway and the back-end Web Service. If the contract for the Web Service changes (for example, a WS-Policy is applied to it at the back-end), you need to re-import the modified WSDL to reflect the changes.

Configure Recipient WS-Policy:
If a recipient WS-Policy was configured when the WSDL file was imported into the Web Service Repository, you can edit its filters using this option. If you did not configure a WS-Policy when importing the WSDL file (using the **Secure Virtual Service** dialog), when you select this option, the **Secure Virtual Service** dialog is displayed. This enables you to select a WS-policy to secure the service. The next time that you select the **Configure Recipient WS-Policy** option, you will edit this policy.

Using WCF WS-Policies

The API Gateway provides four WS-Policies that are identical to those exposed by WCF (Windows Communication Foundation) Web Services. When one of these policies is exposed by a virtual service in the API Gateway, the `svcutil` .NET Web Services utility can consume the WS-Policy and auto-generate clients that communicate securely with the API Gateway.

The security settings for a WCF Web Service are configured in its `web.config` file, in which the `security` element determines the WS-Policy applied to the service. For example, the following extract from a WCF Web Service `web.config` file shows the configuration:

```
<customBinding>
  binding name="MyCustomBinding">
   <textMessageEncoding messageVersion="Soap11" />
   <security defaultAlgorithmSuite="Basic256"
     allowSerializedSigningTokenOnReply="true"
     authenticationMode="MutualCertificate" requireDerivedKeys="false"
     includeTimestamp="true" messageProtectionOrder="SignBeforeEncrypt"
     messageSecurityVersion="WSSecurity10..."
     requireSecurityContextCancellation="false">
   </security>
  </binding>
</customBinding>
```

In this example, the `authenticationMode` for a `customBinding` is set to `MutualCertificate`, which means that messages sent to and from the Web Service must be signed and encrypted with mutual certificates. The following example shows an example of the WCF `wsHttpBinding` configuration, which is less verbose: >

```
<wsHttpBinding>
 <binding name="MyWsHttpBinding">
 <security mode="Message">
  <message clientCredentialType="Certificate" />
 </security>
 </binding>
</wsHttpBinding>
```

The following table shows how the WCF WS-policies provided with the API Gateway correspond to a particular configuration of the `security` element in the WCF Web Service `web.config` file. As shown in the preceding examples, the configuration settings are slightly different, depending on the WCF binding (`customBinding` or `wsHttpBinding`). The following table shows the available settings:

WS-Policy Name	***WCF Binding***	***Authentication Mode***	***Security Mode***	***Client Credential Type***
WCF MutualCertificate Service	`customBinding`	`MutualCertificate`		
WCF UsernameForCertificate Service	`customBinding`	`UserNameForCertificate`		
WCF UsernameOverTransport Service	`customBinding`	`UsernameForTransport`		
WCF BrokeredX509Authentication Service	`wsHttpBinding`		`Message`	`Certificate`

Important

If you intend to consume the WS-Policy exposed by the API Gateway with a WCF client, you should use one of the WCF WS-Policies. All of these policies can be consumed seamlessly by the WCF `svcutil` utility to auto-generate secure clients. While the other WS-Policies exposed by the API Gateway can be consumed by `svcutil`, you need to make additional configuration changes to the auto-generated WCF client to communicate securely with the API Gateway. For more details on making any necessary configuration changes, see your WCF documentation.

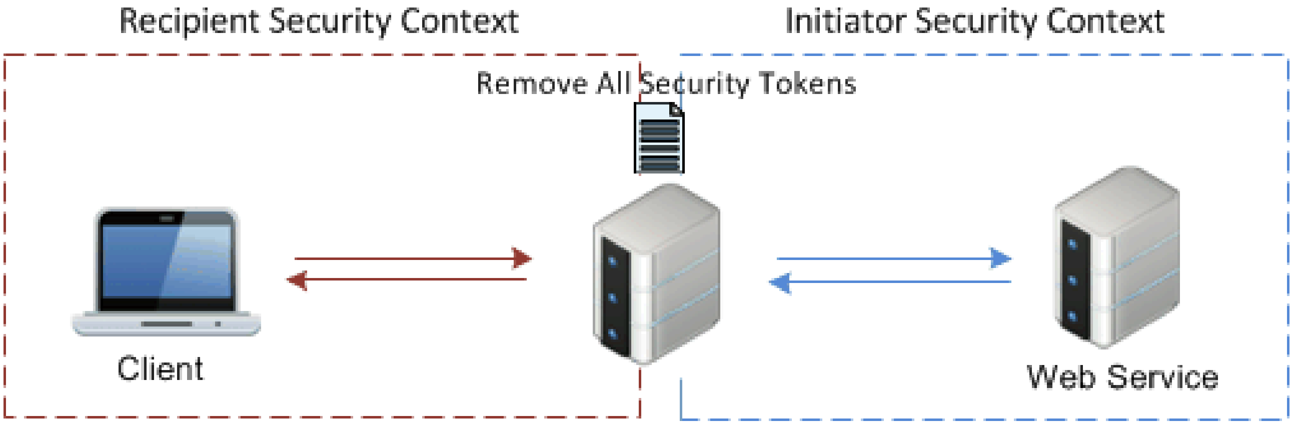
Removing Security Tokens

When you configure a recipient WS-Policy in the **Secure Virtual Service** dialog, the **Remove All Security Tokens** policy is enabled in the **Service Handler** for the imported Web Service. You can view the configured policy by double clicking the **Service Handler**, and selecting the **Message Interception Points** -> **2. User-defined Request Hooks** tab.

The **Remove All Security Tokens** policy ensures that the following security contexts are kept separate:

- *Recipient security context*: This is between the client and the API Gateway, and is determined by the WS-Policy selected in the **Secure Virtual Service** dialog.
- *Initiator security context*: This is between the API Gateway and the back-end Web Service, and is determined by the WS-Policy contained in the imported WSDL for the back-end Web Service.

The **Remove All Security Tokens** policy prevents tokens from one context passing over into the other context, which could breach the security contract governing that context. This ensures that each security context receives a clean SOAP message, on which it can then act to enforce the security requirements of the relevant WS-Policy. The following diagram shows both security contexts and the **Remove All Security Tokens** policy:


Recipient Security Context
Initiator Security Context
Remove All Security Tokens
Client
Web Service

Recipient-side WS-Policy only
In this case, a recipient WS-Policy is configured in the **Secure Virtual Service** dialog to protect a virtual service exposed by the API Gateway. The recipient WS-Policy defines the security contract between the client and the API Gateway. Any security tokens sent by the client are intended for consumption by the API Gateway. They are *not* intended for the back-end Web Service.

For example, the Web Service may not understand SAML, WS-Security, XML Signature, and so on, which may result in a serialization error if these tokens are propagated to it. In addition, it would add unnecessary overhead to the message to propagate security tokens to it. On the response side, the response that the API Gateway returns to the client must adhere to the selected recipient WS-Policy. For example, if the Web Service has returned a SAML Token (out of scope of any WS-Policy requirements), this must not be returned to the client because it would breach the recipient WS-Policy.

Initiator-side and Recipient-side WS-Policy
This occurs when you import a WSDL file that includes a WS-Policy (initiator case), and you also select a WS-Policy in the **Secure Virtual Service** dialog (recipient case). This scenario includes both the *recipient security context* between the client and the API Gateway, and the *initiator security context* between the API Gateway and the Web Service.

It is vital that these security contexts are kept separate because if tokens from one context pass over into the other context, it is highly likely that the security contract for that context will be breached. For example, if the recipient contract between the client and the API Gateway requires a `UsernameToken`, but the initiator contract between the API Gateway and the Web Service requires a SAML token, the `UsernameToken` must not pass over into the initiator context and be sent to the Web Service.

For more details on importing WSDL files that include WS-Policies (initiator case), see *Configuring Security Policies from WSDL Files*.

Important

The **Remove All Security Tokens** policy only applies when a WS-Policy is configured, and is *not* configured when a WS-Policy is not used. In addition, any non-standard behavior that requires a security token to be propagated over to another security context can be handled by disabling the **Remove All Security Tokens** policy in the **Service Handler** for the imported WSDL.

Further Information

For more details on configuring policies to protect your Web Services, see the following:

- *Configuring Security Policies from WSDL Files*
- *Configuring Policies Manually*
- *Web Service Filter*

The **Web Services** filter is the main filter generated when a WSDL file is imported into the Web Services Repository. It contains all the routing information and links all the policies that are to be run on the request and response messages into a logical flow.

Configuring Policies Manually

Overview

In cases where a Web Services definition is not available in a Web Services Description Language (WSDL) file, the policy used to protect a Web Service must be configured manually. The steps outlined in this tutorial describe how to do this.

However, the recommended way to configure a policy to protect a Web Service is to import the WSDL file for that service. If WS-Policy information is contained in the WSDL file, the policy assertions can also be used to produce a complex policy with minimum effort for administrators.

If your Web Service has WSDL-based definitions, see the following:

- *Securing a Virtual Service using Policies*
- *Configuring Security Policies from WSDL Files*

Configuration

The following steps outline how to manually create a policy to protect a Web Service and then test it.

Step 1: Create the Policy
To create a policy manually, complete the following steps:

1. Right-click the **Policies** node in the tree view of the Policy Studio, and select the **Add Policy** menu option.
2. Enter a suitable name (for example `TestPolicy`) for the new policy in the **Name** field, and click the **OK** button. The new policy is now visible in the tree view.
3. Click the new policy in the tree view to start configuring the filters for the policy. You can easily configure the policy by dragging the required filters from the filter palette on the right of the Policy Studio, and dropping them on to the policy canvas.
4. Most policies attempt to check characteristics of the message, such as message size and format, and attempt to authenticate and/or authorize the sender of the message. When the message successfully passes all configured filters, it is usually routed on to the protected Web Service.
5. For demonstration purposes, this topic creates a simple policy consisting of two filters. The first filter checks the size of the message, and the second echoes the request message back to the client if it is below a certain size.
6. Expand the **Content Filtering** category of filters from the filter palette, and drag and drop the **Message Size** filter on to the canvas.
7. Enter `10` in the **At least** field and `1000` in the **At most** field to make sure that only messages between 10 bytes and 1000 bytes are reflected back to the client. Select all other defaults, and click the **Finish** button.
8. Right-click the newly added filter, and select the **Set as Start** menu option to indicate that this is the first filter to be executed in this policy. The icon for the filter changes to indicate that it is the start of the policy.
9. Open the **Utilities** category of filters, and drag the **Reflect** filter onto the canvas. Drop it on to the previously configured **Message Size** filter. Select the defaults for the **Reflect** filter, and click the **Finish** button.
10. Because you dropped the **Reflect** filter on to the **Message Size** filter, both filters are automatically linked with a *success path*. This means that if the first filter runs successfully, the next filter on the success path is executed. To link in more filters, add the filters to the canvas, and click the **Success Path** button at the top of the palette. Click the first filter followed by the second filter in the success path to link both filters.
11. You can also configure *failure paths* for filters in the same way. Failure paths are followed when the checks configured in the filter fail.

This completes the configuration of the simple policy.

Step 2: Create a New Relative Path
You must now create a **Relative Path** on the Oracle API Gateway Process, which maps incoming requests on a particular URI to the new policy. Complete the following steps to do this:

1. In the tree view of the Policy Studio, right-click the **Default Services** node, which can be found under the **Oracle API Gateway** node under the **Listeners** node. Select the **Add Relative Path** menu option.
2. On the **Configure Relative Path** dialog, enter a suitable URI (for example, `TestPolicy`) on which you want to receive requests that are to be processed by the new policy.
3. To map requests received on this URI to our new policy, select the `/TestPolicy` policy from the list of policies in the tree. Click the **OK** button when finished.

Step 3: Deploy to the API Gateway
Before the new configuration changes can take effect, you must deploy them to the API Gateway. You can do this by clicking the **Deploy** button on the right of the toolbar. Alternatively, press the F6 key.

Important

You *must* deploy to the server after making configuration changes. When deploying to the API Gateway, the Policy Studio sends a deployment request to the server. If necessary, you can configure the socket timeout value for this connection in the Policy Studio **Preferences** dialog. Enter the timeout value in milliseconds in the **Server Socket Connection Timeout** field. For more details, see *Policy Studio Preferences*.

Step 4: Test the Policy
You can use the tool of your choice (for example, Oracle API Gateway Explorer) to send SOAP requests to the new policy. You should send requests of different sizes to the following URL, assuming a default installation of the API Gateway running on the local machine:
`http://localhost:8080/TestPolicy`
Request messages that fall between the configured size are reflected to the client. Those fall outside of the configured are blocked, and a SOAP Fault is returned to the client.

Step 5: Next Steps
Try running more complicated checks on request messages by adding new filters to the `TestPolicy`. Try also adding failure paths to the original **Message Size** filter to handle messages that fall outside of the 10-1000 byte range.

Use the **Help** button on each filter screen to find out more about the configuration fields that are available on each screen.

Configuring Global Policies

Overview

Global policies enable you to label policies with specific roles in the API Gateway configuration. For example, you can label a specific policy such as **XML Threat Policy** as a **Global Request policy**. This policy can be executed globally on the request path for all messages passing through the API Gateway. Using a global policy in this way enables you to use the same policy on all requests, and for multiple services. It also means that you can change the labeled global policy to a different policy without needing to rewire any existing policies.

For example, using a **Policy Shortcut Chain** filter in a policy enables you to delegate to one or more policies to perform specific tasks, before continuing execution of the remaining filters in the current policy. Using this approach to encapsulate specific functionality in a policy facilitates modularity and reusability when designing API Gateway policies. This enables you to build up a policy library of reusable routines over time.

Each shortcut in a **Policy Shortcut Chain** points to a specific policy, which is called at each point in the execution chain. However, consider a policy whose role is to be called first in all message handling contexts before any context-specific policies are run, and call this the run-first role. To realize this, you must create a **Policy Shortcut Chain** with a link to the run-first policy as its first entry, the context-specific policy as its second link, and so on.

One of the shortcomings of this approach is that if you have set up a large number of **Policy Shortcut Chain** filters, each calling the run-first policy, and you need to change the run-first policy globally, you must update each **Policy Shortcut Chain** filter individually to point to the newly designated run-first policy. Similarly, if you wish to ignore the run-first Policy globally, you must remove the first entry in each filter.

Global policies enable you to label a specific policy in terms of its role. You can delegate to the policy using its label instead of a specific link to the policy. This indirection using a label makes it very easy to globally change which policy is delegated to, merely by moving the label from one policy to another. Each filter that refers to the policy using its label now resolves the label to the new policy without needing to change the filter configuration. Similarly, if the label is not applied to a specific policy, nothing is executed for this link.

Global Policy Roles

The following global policy roles have a reserved label and a specific meaning in the API Gateway policy framework:

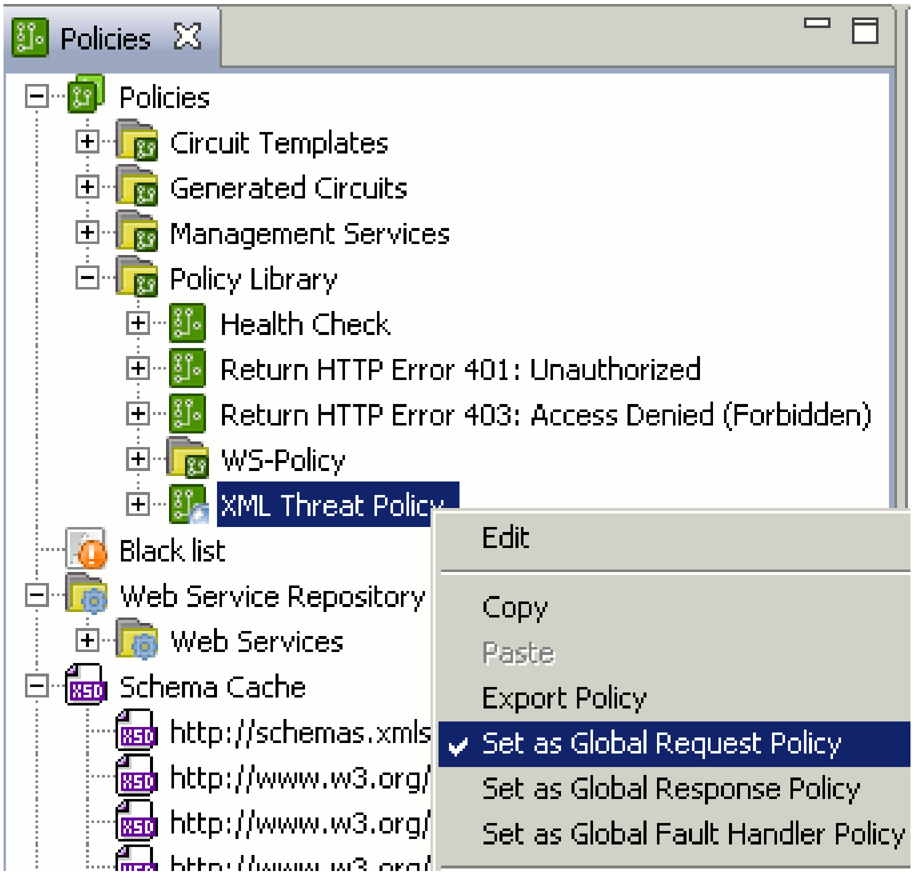
Role	*Label*	*Description*
Global Request Policy	`system.policy.request`	Executed globally for all messages passing through the API Gateway on the request path.
Global Response Policy	`system.policy.response`	Executed globally for all messages passing through the API Gateway on the response path.
Global Fault Handler Policy	`system.policy.faulthandler`	If any policy aborts during execution, or a top-level policy fails and has not specified a Fault Handler filter, this policy is executed instead of the internal **SOAP Fault** filter.

You can select specific policies with these roles under the **Policies** node in the Policy Studio tree. You can then create links to these roles when creating a **Policy Shortcut Chain**. These steps are explained in the next sections.

Selecting a Global Policy

To select a global policy, right-click a policy under the **Policies** node, and select one or more global policies (for example, **Set as Global Request Policy**, **Set as Global Response Policy**, or **Set as Global Fault Handler Policy**). These policies are executed globally for all messages passing through the API Gateway.

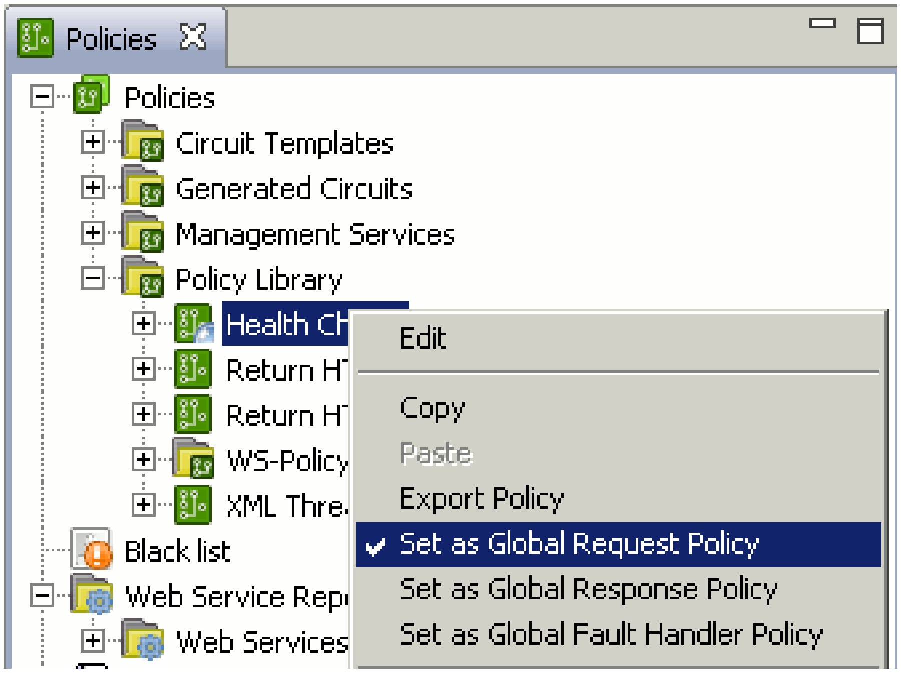
The following example shows the **XML Threat Policy** set as the **Global Request Policy**. The policy node labeled for the specific role is displayed with a globe icon:



When you have selected the policy for a specific role, you can then reference the labeled policy in a **Policy Shortcut Chain** filter, or at the service level in a Relative Path or Web Service Resolver. Referencing a labeled policy is different from referencing a specific policy directly. Referencing a policy directly involves selecting a specific policy to execute in the chain. Referencing a labeled policy means selecting a filter by its label only.

The main advantage of this approach is that you can configure a policy to run in a policy shortcut chain in a specific role, and then select a different policy as the global policy for that role. All references to the global policy label in the various shortcut chain filters are changed to use the newly selected policy, without requiring you to modify each policy shortcut chain filter individually to explicitly point to a different policy.

Selecting another policy in a global role deselects the previously selected policy. The following example shows the **Health Check** set in the global role, and the **XML Threat Policy** policy is no longer selected:

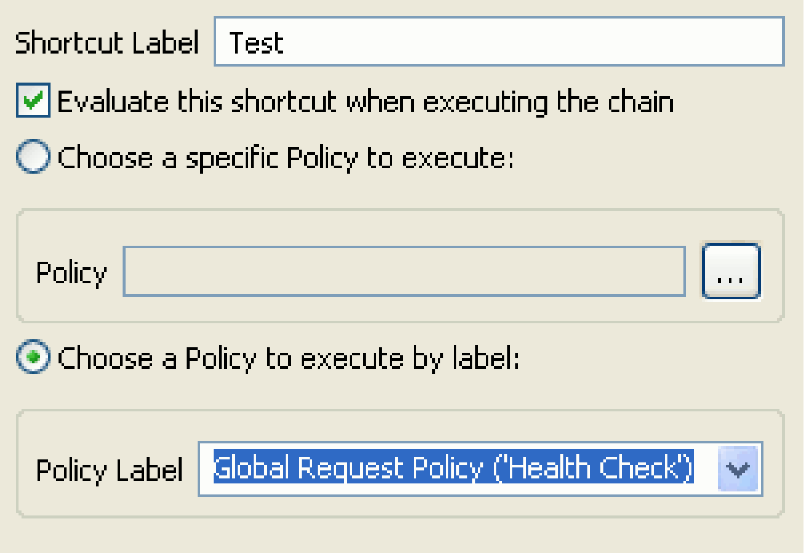


Important

You can not select a policy for a specific role if, in doing so, you create a loop in the policies. For example, if a **Policy Shortcut Chain** filter has a reference to a labeled policy, and the filter's parent policy is marked as the labeled policy, the filter would call back to itself in a loop. This error is caught, and a trace line is output to the Policy Studio **Console** view.

Configuring Global Policies in a Policy Shortcut Chain

When adding a policy shortcut in a **Policy Shortcut Chain** filter, you can select to call a labeled policy instead of selecting a specific policy. The following example from the **Add a new Shortcut to a Policy** dialog shows adding a shortcut to the **Global Request Policy (Health Check)** policy label:



Shortcut Label: Test

☑ Evaluate this shortcut when executing the chain

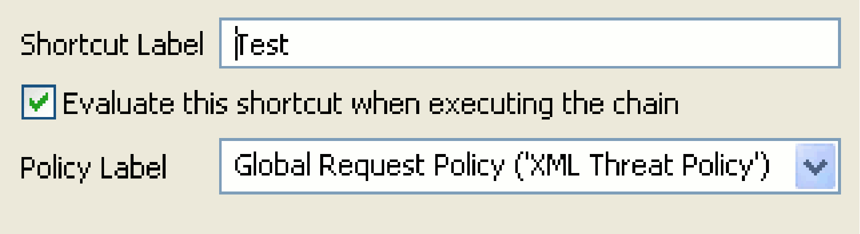
○ Choose a specific Policy to execute:

Policy:

● Choose a Policy to execute by label:

Policy Label: Global Request Policy ('Health Check')

Then if you select a different policy as the request policy in the Policy Studio tree, when you subsequently view this shortcut in the chain filter, you see that the details for the shortcut have changed. The following example from the **Edit the Shortcut to the Policy** dialog shows the policy label changed to **Global Request Policy (XML Threat Policy)**.



Shortcut Label: Test

☑ Evaluate this shortcut when executing the chain

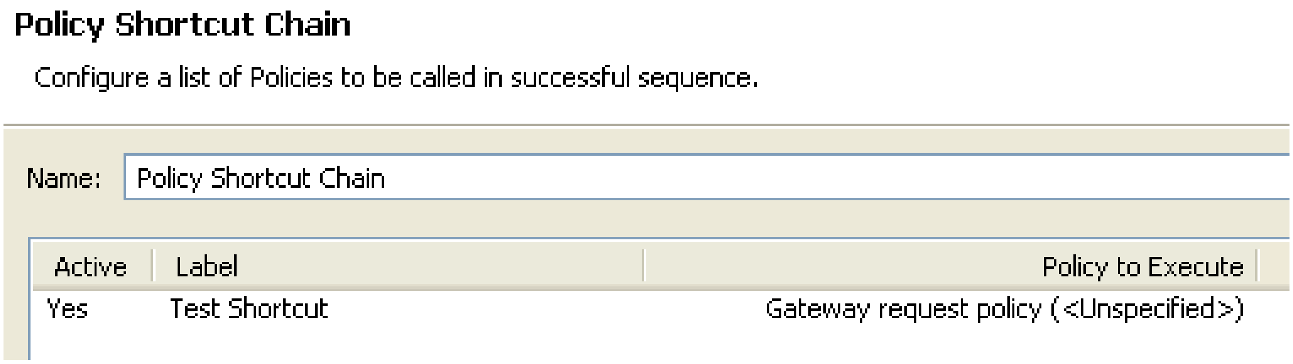
Policy Label: Global Request Policy ('XML Threat Policy')

For more details on configuring these screens, see the *Policy Shortcut Chain* filter.

Important

If you remove a label from a policy by deselecting it in the Policy Studio tree, any reference to that labeled policy is not called when evaluating the shortcut in the chain, irrespective of whether the **Evaluate this shortcut when executing the chain** checkbox is selected (the **Active** status column in the table view). This corresponds with the behavior for a specific policy in the chain. If a link to a policy is not set for a short-cut, the link is not evaluated.

In this example, the table shows that the shortcut is configured to point to the labeled policy, but the label does not resolve to a policy (for example, it is unspecified because there is no policy in the specified role):



Policy Shortcut Chain

Configure a list of Policies to be called in successful sequence.

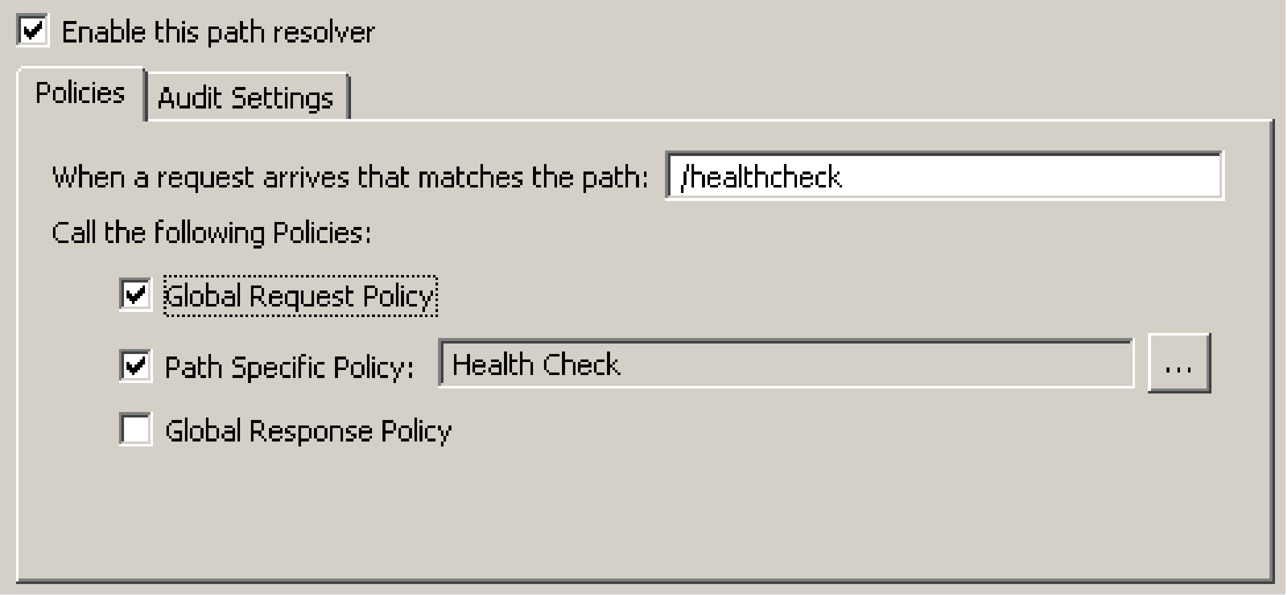
Name: Policy Shortcut Chain

Active	Label	Policy to Execute
Yes	Test Shortcut	Gateway request policy (<Unspecified>)

Configuring Global Policies for a Service

Under the **Listeners** node, you can also configure global policies at the service level to run on a specific Relative Path or Web Service Resolver when messages are received by the API Gateway. A Relative Path binds a policy to a specific relative path location (for example `/healthcheck`). A Web Service Resolver maps messages destined for a specific Web Service to a **Service Handler** or **Web Service Filter**.

You can configure a global policy at the service level to run as part of a policy chain invoked when incoming messages are received by the API Gateway. The following example shows the **Global Request Policy** configured to run first on the `/healthcheck` relative path:



☑ Enable this path resolver

When a request arrives that matches the path: /healthcheck

Call the following Policies:

☑ Global Request Policy

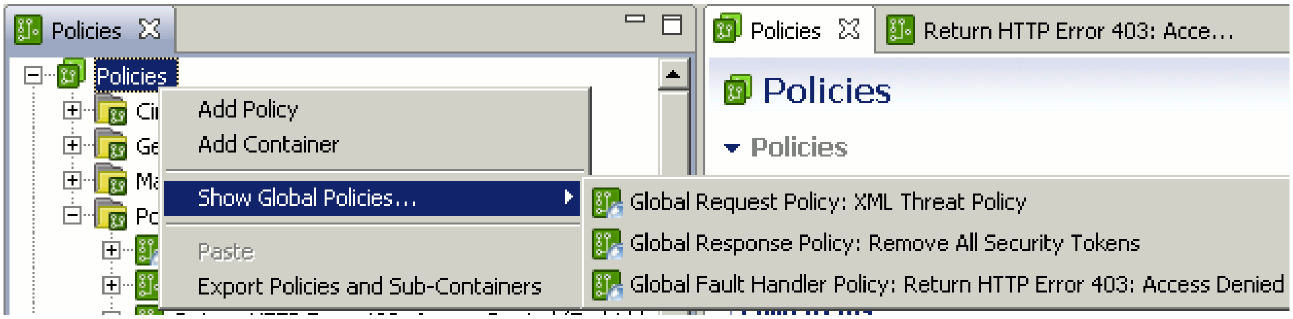
☑ Path Specific Policy: Health Check

☐ Global Response Policy

For more details on how to configure a global policy for a service, see the section called “Relative Paths” and the section called “Web Service Resolvers” in the *Configuring HTTP Services* topic.

Showing Global Policies

To view the currently configured global policies, right-click the **Policies** root tree node, and select **Show Global Policies**. This displays all currently configured global policies in the context menu, for example:


Policies
Policies
Ci
Ge
Ma
Po
Add Policy
Add Container
Show Global Policies...
Paste
Export Policies and Sub-Containers
Policies
Return HTTP Error 403: Acce...
Policies
Policies
Global Request Policy: XML Threat Policy
Global Response Policy: Remove All Security Tokens
Global Fault Handler Policy: Return HTTP Error 403: Access Denied

Note

If there are no global policies configured, the **Show Global Policies** menu item is not available.

Configuring Policy Packages

Overview

In some cases, you may need to convert a policy into a modular, reusable piece of functionality that can be called from other policies. For example, if you have created a complicated policy that creates a WS-Security Username, inserts it into the message, and subsequently creates an XML Signature over the token and SOAP body. Depending on the ultimate recipient for this message, the content may need to be signed using slightly different settings. One service may require one set of algorithms to be used (for example, a `<sp:Basic128/>` Algorithm Suite in WS-SecurityPolicy). While another policy may require a different set of algorithms (for example, `<sp:Basic256/>`).

Similarly, subtle differences in the security requirements of services may require the token and signature to be generated using different configurations. For example:

- Use a basic or digest password for the `UsernameToken`
- Insert a `<dsig:CarriedKeyName>` into the XML-Signature
- Create an enveloped or enveloping signature
- Include a `<wsse:BinarySecurityToken>`
- Use one signing key over another
- Sign different parts of the message

If you need to create separate policies to implement such nuances, the task of interoperating with different vendor services can become arduous, and involves creating several complicated policies where each may only differ in one field in a given filter. To avoid this duplication, you can create a *policy package* that inserts the WS-Security `UsernameToken` into the message and generates the XML-Signature. However, instead of explicitly configuring fields mentioned above (for example, enveloped or enveloping signature, include a `<wsse:BinarySecurityToken>`, and the signing key to use), the policy package can use selectors for these fields, which are configured dynamically at runtime instead of statically at design time. For more details, see *Selecting Configuration Values at Runtime*.

The policy package advertises that it requires certain configuration details to be called generically from other policies. For example, it would typically require the key to sign the message. By templating the signing policy as a policy package, and making it available to call from other policies like any other filter, the caller must set the signing key for the policy package to use. In this way, two distinct policies that require a signed `UsernameToken` can call the same policy package. By using selectors to pass in different signing keys, messages are signed using the appropriate key for each calling policy.

When a policy has been configured as a policy package, it is displayed in the Policy Studio filter palette, and you can drag and drop it into any policy that requires the functionality encapsulated in the package. You must configure any fields required by the policy package when it is dragged and dropped on to the canvas of another policy.

Configuring a Policy Package

To configure a policy as a policy package in the Policy Studio, perform the following steps:

1. Right-click the policy in the Policy Studio tree on the left, and select **Policy Package** -> **Create**.
2. Specify the following settings on the **General** tab:
   - **Palette Category**
     Enter the filter palette category in which to display the policy package (for example, `Monitoring`).
   - **Palette Icon**
     Enter the path to the palette icon to display the policy package (for example, `C:\Oracle\apigateway\icons\monitor.ico`).
3. The **Input** tab lists all required message attributes for the policy package. You can enter user-friendly names for each attribute to be displayed in the **Policy Activation** filter for the policy package (for example, `HTTP Headers` for the `http.headers` attribute).

4. The **Output** tab lists the generated message attributes for the policy package. To add a generated attribute, click **Add**, and enter the following details:
   - **Expression**
     Enter the selector expression for the attribute (for example, `${content.body}`).
   - **Attribute Type**
     Enter the message attribute type (for example, `com.vordel.mime.Body`).
   - **Output Attribute Name**
     Enter the message attribute generated by the policy package (for example, `content.body`).
5. When finished, click **OK**.
6. Click the **Deploy** button in the toolbar to deploy the newly created policy package to the API Gateway.

Applying a Policy Package

When a policy is configured as a policy package, it is available for reuse in the Policy Studio filter palette. Draging and droping the policy package on to the policy canvas displays the **Policy Activation Filter** screen for that policy package. This enables you to specify any required message attributes and filter-level monitoring settings.

Specifying Required Attributes

The **Required Attributes** tab enables you to set the configuration fields required by the policy package (for example, those configured with selectors for dynamic configuration). Click **Add** to specify the following fields:

- **Required Attribute**
  Enter the name of the required attribute for display (for example, `HTTP Header`).
- **Raw Attribute Name**
  Enter the message attribute name (for example, `http.headers`).
- **Attribute Type**
  Enter the message attribute type (for example, `com.vordel.mime.HeaderSet`).
- **Value/Selector**
  Enter a message attribue value or selector (for example, `${http.headers}`).

Specifying Monitoring Settings

The **Traffic Monitor** tab enables you to set filter-level monitoring settings. You can configure the following fields:

- **Record outbound transactions**
  Select whether to record outbound message transactions sent from the API Gateway to remote hosts. This is enabled by default.
- **Record policy path**
  Select whether to record the policy path for the message transaction, which shows the filters that the message passes through. This is enabled by default.
- **Trace level**
  Select the filter trace level from the list. Defaults to `INFO`.

Applying a Policy Package to a Service

The reusabe policy packages created in Policy Studio can also be applied to services in API Service Manager. When you deploy the newly created policy package to the API Gateway, it becomes available for selection in the **New API Service** wizard in the API Service Manager. For more details, see the topic on *Managing API Services*.

Getting Started with Managing Deployments

Overview

When connected to the Admin Node Manager server, you can deploy configurations to API Gateway instances running in groups in a domain. In the Policy Studio, the **Topology** view enables you to edit the configuration of currently running API Gateway instances. You can update the downloaded configuration, and deploy it to the server, where it can be reloaded later. You can deploy modified configuration to multiple API Gateway instances managed by Policy Studio. You can also create groups and API Gateway instances.

The web-based API Gateway Manager enables you to deploy configurations to API Gateway instances running in groups in a domain, to create groups and API Gateway instances, and to manage Admin Users. In this way, Policy Studio and the API Gateway Manager play a key role in an operational Service Oriented Architecture (SOA). These tools enable policy developers and administrators to centrally manage the policies that are enforced at all nodes throughout the network.

Connecting to a Server in the Policy Studio

Before starting Policy Studio, you should first ensure that the Admin Node Manager and the server instance that you wish to connect to have been started. For more details, see *Startup Instructions*.

When the Policy Studio starts up, click a link to a server to display the **Open Connection** dialog. You can use this dialog to specify **Connection Details** (for example, host, port, user name, and password) or to specify **Saved Sessions**. If you wish to connect to the server using a non-default URL, click **Advanced**, and enter the **URL**. The default Admin Node Manager URL is:

```
https://localhost:8090/api
```

Alternatively, you can connect to a server configuration file by clicking the **Open File** button. For more details on connecting using a server URL, configuration file, or deployment archive, see *Connection Details*.

Note

You must connect to the Admin Node Manager server to deploy API Gateway configuration or manage multiple API Gateway instances in your network.

When the connection to the server has been made, the **Topology** view is displayed. This displays the list of server instances currently managed by the Admin Node Manager in the Policy Studio, and enables you to manage the configuration for server instances.

Editing Server Configuration in the Policy Studio

The **Topology** view lists all available instances in each group. Double-click an instance name in the list to load its active configuration. Alternatively, right-click an instance name, and select **Edit Configuration**. The active server configuration is loaded and displayed in the following format: **InstanceName [HostName:Port]** (for example, **test_server [roadrunner.acme.com:8085]**).

When an active server configuration is loaded, its services are displayed under the **Listeners** node in the Policy Studio tree on the left. Expand one of the top-level nodes in the tree to display additional details (for example, **Business Services**, **External Connections**, **Resources**, **Libraries**, or **Settings**).

When editing an active server configuration, you can deploy updates using the **Deploy** button in the toolbar (alternatively, press **F6**).

Managing Deployments in the API Gateway Manager

In the web-based API Gateway Manager tool, the **TOPOLOGY** section on the **Dashboard** tab enables you to create groups and API Gateway instances, and to deploy configuration. For details on how to access the API Gateway Manager, see the section called “Launching API Gateway Manager”.

The **API Service Manager** tab in the API Gateway Manager enables you to manage your services and policies online. You must first create your API Service Manager workspace. For more details, see *Introduction to API Service Manager*. You can then use the API Service Manager to virtualize API services. For more details, see *Managing API Services*.

Managing Admin Users in the API Gateway Manager

You can add new Admin Users to enable role-based access to the API Gateway configuration managed by the Policy Studio and API Gateway Manager. The default `admin` user has access to all API Gateway features in the Policy Studio and API Gateway Manager, and can view and modify all API Gateway configurations.

To add or remove Admin Users, click the **Settings** -> **Admin Users** tab in the API Gateway Manager. For more details, see *Managing Admin Users*.

For more details on role-based access, see *Configuring Role-Based Access Control (RBAC)*.

Configuring Policies in the Policy Studio

You can use Policy Studio to manage the configuration of your policies, which can then be deployed to running instances of Oracle API Gateways.

For details on configuring the full range of message filters (for example, for Authentication, Authorization, or Content Filtering), see the Table of Contents for this guide. For details on general configuration topics, see Chapter 6, General Configuration.

Deploying Configuration

Overview

A *deployment archive* is a `.fed` file that contains API Gateway configuration. For example, this includes a Certificate Store, User Store, Core Configuration, External Connections and Listeners. Such a collection of configuration is also known as a *configuration assembly*. Using a deployment archive to deploy an assembly enables you to edit configuration offline, and then deploy elsewhere later on a specified process. For more details, see the section called “Creating a Deployment Archive in the Policy Studio”.

A *configuration property* is a name-value pair that identifies a specific configuration assembly. Specifying a property associates metadata with the corresponding configuration profiles in that assembly. For example, the **Name** property with a value of `Default Factory Configuration` associated with a default installation. Assigning properties to an configuration also enables you to edit configuration offline, and deploy later on a specified process.

You can use the Policy Studio to create deployment archives. You can also use the Policy Studio to deploy an existing deployment archive, factory configuration, or working configuration on selected API Gateway instances. You can also use the API Gateway Manager to deploy a deployment archive in your Web browser. Alternatively you can use the `managedomain` script to create and deploy deployment archives on the command line.

Creating a Deployment Archive in the Policy Studio

Currently Loaded Configuration
To create a deployment archive (`.fed` file) for a currently loaded API Gateway configuration, perform the following steps:

1. In the main menu, select **File** -> **Save Deployment Archive**.
2. Enter values for the appropriate configuration properties (**Name**, **Description**, and **Version**).
3. If you wish to create any additional properties (for example, **Department**), click the green plus button on the right, and enter a property value (for example, `Engineering`).
4. Click **OK**.
5. Enter a filename for the `.fed` file, and click **Save**.

API Gateway Group View
To create a deployment archive in the API Gateway **Group** view, perform the following steps:

1. Right-click a server in the tree, and select **Save Deployment Archive**.
2. Browse to a directory in the dialog.
3. Click **OK**. The file is saved to the specified directory (for example, `c:\temp\5c3b2a3c-23a5-4261-87cb-eca150f0a037.fed`.
4. Click **OK**.

To view and modify configuration properties in the API Gateway **Group** view, perform the following steps:

1. Right-click a server in the tree, and select **View/Modify Properties**.
2. Enter values for the appropriate configuration properties (**Name**, **Description**, and **Version**).
3. If you wish to create additional properties (for example, **Department**), click the green plus button on the right, and enter a property value (for example, `Engineering`).
4. Click **Update Configuration Properties**.
5. Click **OK**.

Deploying a Deployment Archive in the Policy Studio

To deploy an existing deployment archive (`.fed` file) in the API Gateway **Group** view, perform the following steps:

1. Click the **Deploy** button in the toolbar.
2. In the **Select the servers(s) you wish to deploy to** section, select a server group from the list, and select the server instance(s) in the box below.
3. In the **Select the configuration you wish to deploy** section, select **I wish to deploy an existing archive**.
4. In the **Location of Archive** field, click **Browse**, and select the `.fed` file.
5. Click **Deploy** to upload the archive to the Admin Node Manager and deploy to the selected server(s).
6. When the archive has deployed, click **Finish**.

Deploying a Factory Configuration in the Policy Studio

To deploy a default factory configuration in the API Gateway **Group** view, perform the following steps:

1. Click the **Deploy** button in the toolbar.
2. In the **Select the servers(s) you wish to deploy to** section, select a server group from the list, and select the server instance(s) in the box below.
3. In the **Select the configuration you wish to deploy** section, select **I wish to deploy a factory configuration**.
4. Click **Deploy** to deploy the configuration to the selected server(s).

Deploying a Currently Loaded Configuration in the Policy Studio

To deploy a currently loaded configuration in the Policy Studio, perform the following steps:

1. Click the **Deploy** button on the right in the toolbar.
2. In the **Select the servers(s) you wish to deploy to** section, select a server group from the list, and select the server instance(s) in the box below.
3. Enter values for the appropriate configuration properties (**Name**, **Description** and/or **Version**).
4. If you wish to create any additional properties (for example, **Department**), click the green plus button on the right, and enter a property value (for example, `Engineering`).
5. Click **Deploy**.

Deployment Summary in the Policy Studio

When the **Deployment Results** screen is displayed, the deployment starts, and deployment to each server occurs sequentially. Feedback is provided using icons in the **Task** column, and text in the **Status** column. When the configuration has deployed, click **Finish**.

Canceling Deployments
You can cancel deployments by clicking the **Cancel** button. Feedback is provided in the **Status** column. You cannot cancel a deployment when it has started. The wizard performs the cancellation at the end of the current deployment, with all remaining deployments being cancelled.

Deployment Errors
Client-side and server-side errors can occur. Client-side errors are displayed in the **System Trace** in the **Console** view. If any server-side deployment errors occur during the deployment process, you can review these in the **Deployment Error Log** view. This is displayed at the bottom of the screen when you click **Finish**, and lists any errors that occur for each process. The corresponding Console **Deployment Log** is also available in the **Console** view.

Redeploying
When you have deployed a configuration to one or more processes, you can click back through the wizard to change your selections and redeploy, without needing to exit and relaunch the wizard.

Deploying an Archive in API Gateway Manager

You can use the API Gateway Manager in a Web browser to deploy an existing deployment archive to a group of API Gateways. Perform the following steps:

1. In the **Topology** view, right-click the API Gateway, and select **Deploy**.
2. Browse to the deployment archive to be deployed.
3. Choose the Group and API Gateways to which you wish to deploy the archive.
4. Select **Deploy** in the wizard, and the deployment archive is deployed to the selected API Gateways.

Deploying on the Command Line

You can create and deploy a deployment archive using the `managedomain` script in the following directory:

Windows	`INSTALL_DIR\Win32\bin`
UNIX/Linux	`INSTALL_DIR/posix/bin`

The deployment options in the `managedomain` script are as follows:

```
18) Deploy to a group
  19) List deployment information
  20) Create deployment archive
  21) Download deployment archive
  22) Update deployment archive properties
```

Deploying the API Gateway in Multiple Environments

Overview

You can specify configuration values in the API Gateway on a per-environment basis using environment variables in the `envSettings.props` file. For example, you can specify the port on which the API Gateway listens for HTTP traffic with different values depending on the environment in which the API Gateway is deployed.

The environment variable settings in the `envSettings.props` file are external to the API Gateway core configuration. The API Gateway runtime settings are determined by a combination of external environment variable settings and core configuration polices. This mechanism provides a simple and powerful approach for configuring the API Gateway to work across multiple environments.

The `envSettings.props` file is located in the `conf` directory of your API Gateway installation, and is read each time the API Gateway starts up. Environment variable values specified in the `envSettings.props` file are displayed as environment variable selectors in the Policy Studio (for example, `${env.PORT.TRAFFIC}`) For more details on selectors, see *Selecting Configuration Values at Runtime*.

Configuring Environment Variables

The `envSettings.props` file enables you to externalize configuration values and set them on a per-environment basis. This section shows the configuration syntax used, and shows some example values in this file.

Environment Variable Syntax
If the API Gateway configuration contains a selector with a format of `${env.`*X*`}`, where *X* is any string (for example, `MyCustomSetting`), the `envSettings.props` file must contain an equivalent name-value pair with the following format:

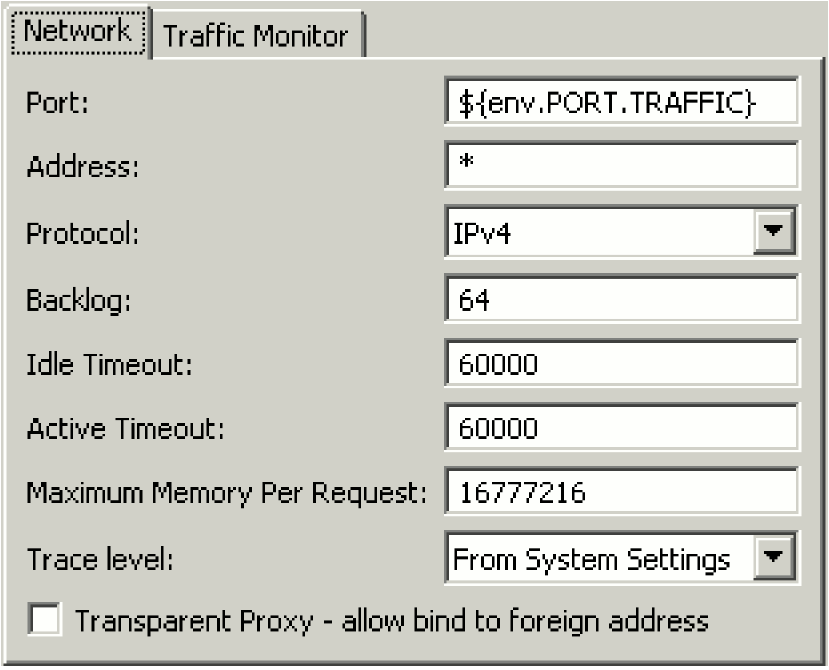
```
env.MyCustomSetting=MyCustomValue
```

When the API Gateway starts up, every occurrence of the `${env.MyCustomSetting}` selector is expanded to the value of `MyCustomValue`. For example, by default, the HTTP port in the server configuration is set to `${env.PORT.TRAFFIC}`. Specifying a name-value pair of `env.PORT.TRAFFIC=8080` in the `envSettings.props` file results in the server opening up port 8080 at start up.

Example Settings
The following simple example shows some environment variables set in the `envSettings.props` file:

```
# default port the API Gateway listens on for HTTP traffic
env.PORT.TRAFFIC=8080

# default port the API Gateway listens on for management/configuration HTTP traffic
env.PORT.MANAGEMENT=8090
```

The following example screenshot shows the corresponding `${env.PORT.TRAFFIC}` selector displayed in the **Configure HTTP Interface** dialog. At runtime, this is expanded to the value of the `env.PORT.TRAFFIC` environment variable specified in the `envSettings.props` file:



Port: ${env.PORT.TRAFFIC}

Address: *

Protocol: IPv4

Backlog: 64

Idle Timeout: 60000

Active Timeout: 60000

Maximum Memory Per Request: 16777216

Trace level: From System Settings

□ Transparent Proxy - allow bind to foreign address

Important

All entries in the `envSettings.props` file use the `env.` prefix, and the corresponding selectors specified in the Policy Studio use the `${env.*}` syntax. If you update the `envSettings.props` file, you must restart or deploy the API Gateway for updates to be applied to the currently running API Gateway configuration.

Configuring Certificates as Environment Variables

You can also use the `envSettings.props` file to bind a reference to a server host-specific SSL certificate to a specific deployment.

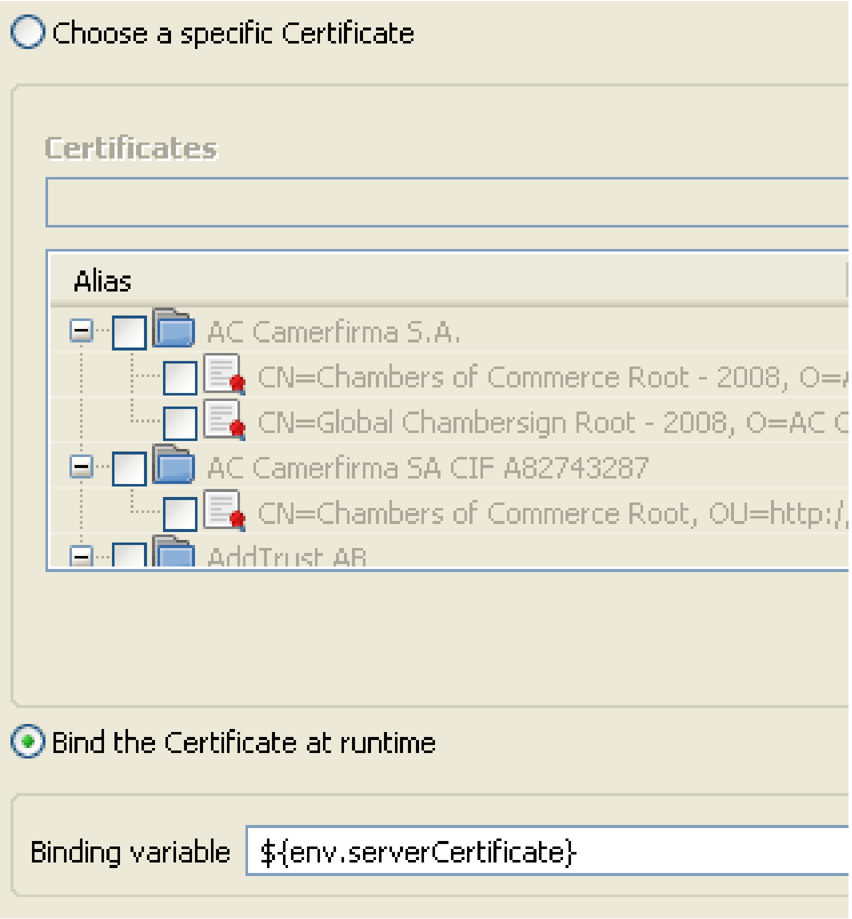
Example Syntax:
The following entry shows an example of the environment variable syntax used to specify a server host-specific certificate:

```
env.serverCertificate=${system.prefix.cert}MY_ALIASED_CERT_NAME
```

Alternatively, the following entry shows the syntax when the alias is the same as the Distinguished Name:

```
env.serverCertificate=${system.prefix.cert}CN=MY_HOST
```

Example Settings
When the `env.serverCertificate` variable is specified in the `envSettings.props` file, the **X.509 Certificate** field in the **Configure HTTPS Interface** dialog can then reference its value using the `${env.serverCertificate}` selector. The following example screenshot shows the corresponding `${env.serverCertificate}` selector specified at the bottom of the **Select Certificate** dialog, which is displayed by pressing the **X.509 Certificate** button:



○ Choose a specific Certificate

Certificates

Alias

□ AC Camerfirma S.A.

□ CN=Chambers of Commerce Root - 2008, O=A

□ CN=Global Chambersign Root - 2008, O=AC C

□ AC Camerfirma SA CIF A82743287

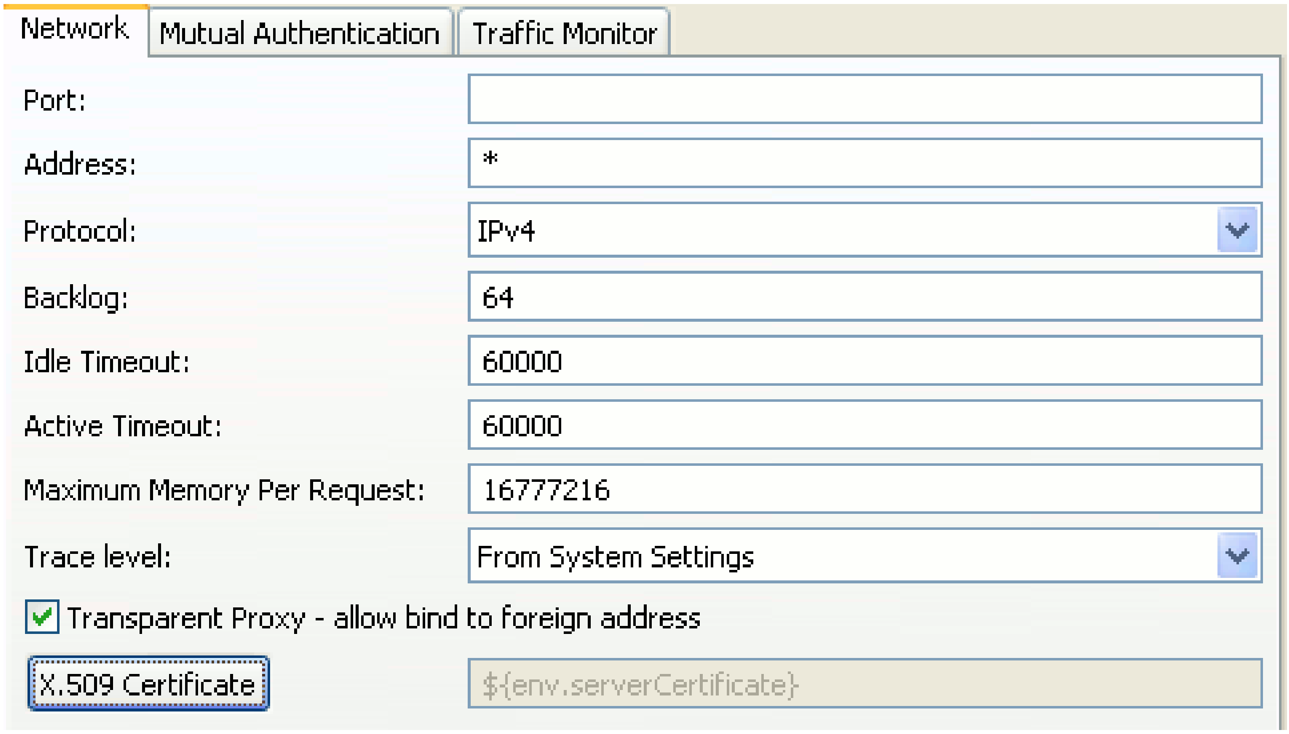
□ CN=Chambers of Commerce Root, OU=http:/,

□ AddTrust AB

● Bind the Certificate at runtime

Binding variable ${env.serverCertificate}

The following example screenshot then shows the `${env.serverCertificate}` selector referenced in **X.509 Certificate** field:



Port:

Address: *

Protocol: IPv4

Backlog: 64

Idle Timeout: 60000

Active Timeout: 60000

Maximum Memory Per Request: 16777216

Trace level: From System Settings

☑ Transparent Proxy - allow bind to foreign address

${env.serverCertificate}

Important

In the `envSettings.props` file, you must specify commas using `\\` escape characters. For example:

```
env.serverCertificate=${system.prefix.cert}CN=
linux-test-desktop\\,OU=QA\\,O=Saturn Inc.\\,L=Dublin\\,ST=Dublin\\,C=IE
```

Managing Admin Users

Overview

When logging into the Policy Studio or API Gateway Manager, you must enter the user credentials stored in the local Admin User store to connect to the API Gateway server instance. Admin Users are responsible for managing API Gateway instances using the API Gateway management APIs. You can manage Admin Users by clicking the **Settings** -> **Admin Users** tab in the API Gateway Manager.

Note

Admin Users provide access to the API Gateway configuration management features available in the Policy Studio and API Gateway Manager. Whereas API Gateway Users provide access to the messages and services protected by the API Gateway. For more details, see the *API Gateway Users* topic.

Admin User Privileges

After installation, a single Admin User is defined in the API Gateway Manager with a username of `admin`. Admin User rights in the system include the following:

- Add another Admin User.
- Delete another Admin User.
- Update an Admin User.
- Reset Admin User passwords.

Important

An Admin User *cannot* delete itself.

Removing the Default Admin User
If you wish to remove the default Admin User, perform the following steps:

1. Add another Admin User.
2. Log in as the new Admin User.
3. Delete the default Admin User.

The **Admin Users** tab displays all existing Admin Users. You can use this tab to add, update, and delete Admin Users. These tasks are explained in the sections that follow.

Admin User Roles

The API Gateway uses Role-Based Access Control (RBAC) to restrict access to authorized users based on their assigned roles in a domain. Using this model, permissions to perform specific system operations are assigned to specific roles only. This simplifies system administration because users do not need to be assigned permissions directly, but instead acquire them through their assigned roles.

For example, the default Admin User (`admin`) has the following user roles:

- `Policy Developer`
- `API Server Administrator`

- `API Service Developer`
- `KPS Administrator`

User Roles and Privileges
User roles have specific tools and privileges assigned to them. These define who can use which tools to perform what tasks. The user roles provided with the API Gateway assign the following privileges to Admin Users with these roles:

Role	Tool	Privileges
`Policy Developer`	Policy Studio	Download, edit, deploy, version, and tag a configuration.
`API Service Developer`	API Service Manager	Perform create, read, update, delete (CRUD) operations, and deploy API services. No access to other API Gateway Manager tabs.
`API Service Administrator`	API Service Manager	Read-only list of API services. No access to other API Gateway Manager tabs.
`API Server Administrator`	API Gateway Manager	Read/write access to API Gateway Manager. No access to API Service Manager tab.
`API Server Operator`	API Gateway Manager	Read-only access to API Gateway Manager. No access to API Service Manager tab.
`Deployer`	Deployment scripts	Deploy a new configuration.
`KPS Administrator`	KPS Web UI	Perform CRUD operations on data in a Key Property Store (KPS).

Note

A single Admin User typically has multiple roles. For example, in a development environment, a policy developer Admin User would typically have the following roles:

- `Policy Developer`
- `API Service Developer`
- `API Server Administrator`

Adding a New Admin User

Complete the following steps to add a new Admin User to the system:

1. Click the **Settings** -> **Admin Users** tab in the API Gateway Manager.
2. Click the **Create** button.
3. In the **Create New Admin User** dialog, enter a name for the User in the **Username** field.
4. Enter a user password in the **Password** field.
5. Re-enter the user password in the **Confirm Password** field.
6. Select roles for the user from the list of available roles (for example, `Policy Developer` and/or `API Server Administrator`).

7. Click **Create**.

Removing an Admin User

To remove an Admin User, select it in the **Username** list, click the **Delete** button. The Admin User is removed from the list and from the local Admin User store.

Resetting an Admin User Password

You can reset an Admin User password as follows:

1. Select the Admin User in the **Username** list.
2. Click the **Edit** button.
3. Enter and confirm the new password in the **Password** and **Confirm Password** fields.
4. Click **OK**.

Managing Admin User Roles

You can manage the roles that are assigned to specific Admin Users as follows:

1. Select the Admin User in the **Username** list.
2. Click the **Edit** button.
3. Select the user roles that you wish to enable for this Admin User in the dialog (for example, `Policy Developer` and/or `API Server Administrator`).
4. Click **OK**.

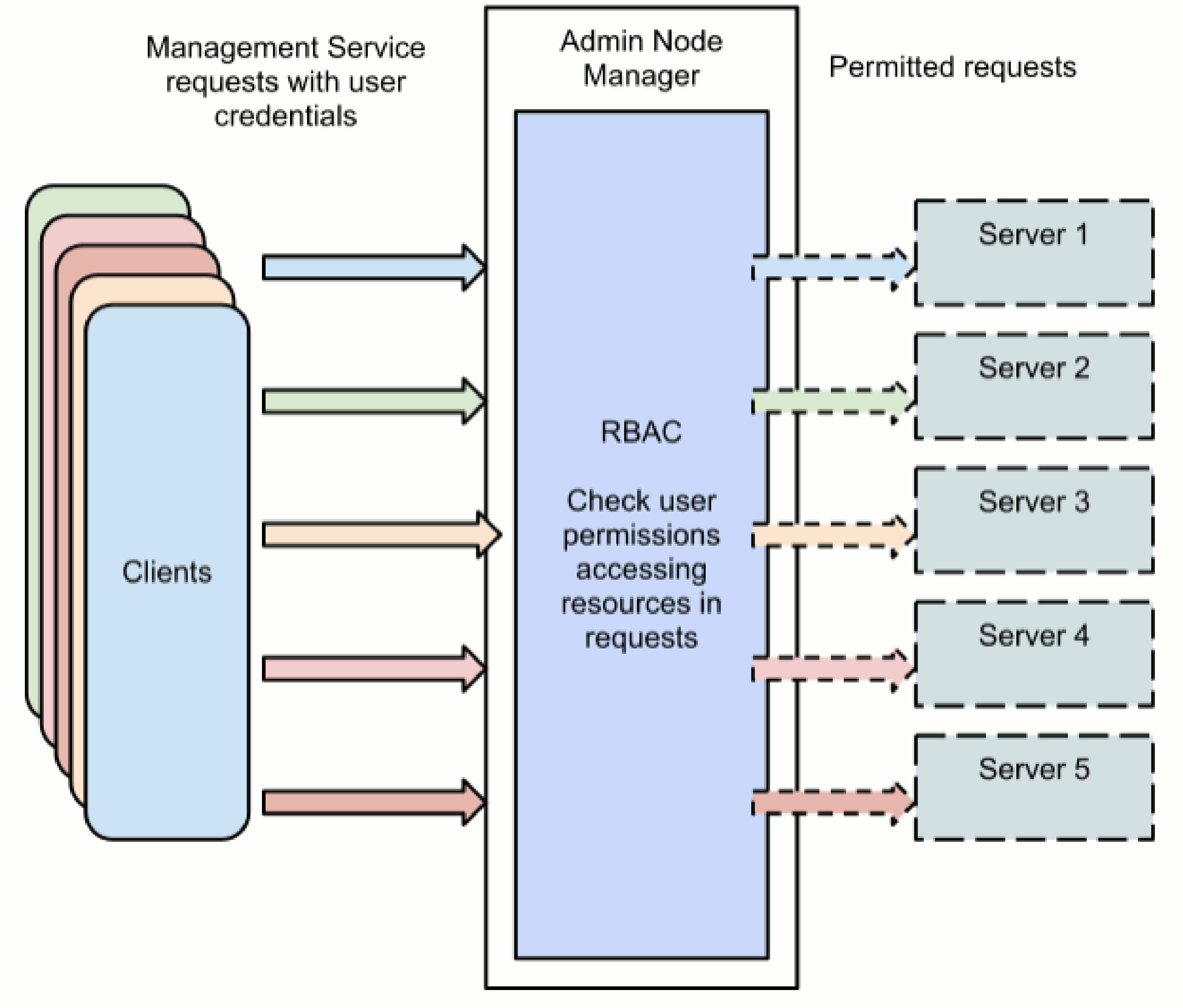
Editing Roles
To add or delete specific roles, you must edit the available roles in the `adminUsers.json` and `acl.json` files in the `conf` directory of your API Gateway installation. For full details on managing roles, see the topic on *Configuring Role-Based Access Control (RBAC)*.

Configuring Role-Based Access Control (RBAC)

Overview

Role-Based Access Control (RBAC) enables you to restrict system access to authorized users based on their assigned roles. Using the RBAC model, permissions to perform specific system operations are assigned to specific roles, and system users are granted permission to perform specific operations only through their assigned roles. This simplifies system administration because users do not need to be assigned permissions directly, and instead acquire them through their assigned roles.

The API Gateway uses the RBAC permissions model to ensure that only users with the assigned role can access parts of the Management Services exposed by the Admin Node Manager. For example, this includes access to traffic monitoring data or making a configuration change by deploying to a group of API Gateways. The following diagram shows an overview of the RBAC model in the API Gateway:


Management Service requests with user credentials
Admin Node Manager
Permitted requests
Clients
RBAC
Check user permissions accessing resources in requests
Server 1
Server 2
Server 3
Server 4
Server 5

API Gateway Manager

The web-based API Gateway Manager tool (`https://localhost:8090`) is a centralized dashboard for managing and monitoring the API Gateway, and is controlled by RBAC. Users connecting to this URL with different roles results in different features being displayed.

For example, a user with the `API Service Administrator` or `API Service Developer` role can access the API Service Manager tool. However, users in the `Policy Developer`, `API Gateway Operator`, or `Deployer` roles cannot access the API Service Manager tool.

For more details on the tools and privileges assigned to specific user roles, see the topic on *Managing Admin Users*.

Protected Management Services

The Admin Node Manager exposes a number of REST Management Services, which are all protected by RBAC. For example, the exposed services and the associated tools that use them include the following:

Protected Service	Tool	Description
Traffic Monitoring Service	API Gateway Manager	Displays HTTP, HTTPS, JMS, and FTP message traffic processed by the API Gateway.
API Service Manager API	API Gateway Manager	Enables users to virtualize REST APIs and SOAP Services on the API Gateway.
Configuration Service	API Gateway Manager	Adds and removes tags on the API Gateway.
Topology API	API Gateway Manager	Accesses and configures API Gateway domains.
Static Content Resources	API Gateway Manager	Manages UI elements in a browser.
Deployment API	Policy Studio	Deploys configurations to the API Gateway.
KPS Service	Policy Studio	Manages a Key Property Store.

User Roles

User access to Management Services is determined by their role(s). Each role has a defined set of Management Services that it can access. A Management Service is defined by the URI used to access it, for example:

Role Name	Service Name	API Type	Example URI
`API Gateway Operator`	Topology API	REST	`/api/topology/hosts`
`API Gateway Administrator`	Deployment API	REST	`/api/router/service/instance-1/deployment/domain/deployments`
`API Service Administrator`	Static Content Resource	Static	`/`

For full details on the default roles that have access to each Management Service, see the section called “Management

Service Roles and Permissions”.

Local Admin User Store

By default, all the user credentials are stored in a local Admin User store in the following file:

```
INSTALL_DIR/conf/adminUsers.json
```

`INSTALL_DIR` is the directory where the API Gateway is installed as Admin Node Manager.

The following shows an example file:

```
{
  "version" : 1,
  "adminUserPasswords" : {
    "user-1" : "Y2hhbmdlbWU="
  },
  "productVersion" : "7.1.0",
  "adminUsers" : [ {
    "name" : "admin",
    "id" : "user-1",
    "roles" : [ "role-1", "role-4", "role-6", "role-7" ]
  } ],
  "adminUserRoles" : [ {
    "name" : "API Server Administrator",
    "id" : "role-1"
  }, {
    "name" : "API Server Operator",
    "id" : "role-2"
  }, {
    "name" : "API Service Administrator",
    "id" : "role-3"
  }, {
    "name" : "API Service Developer",
    "id" : "role-4"
  }, {
    "name" : "Deployer",
    "id" : "role-5"
  }, {
    "name" : "KPS Administrator",
    "id" : "role-6"
  }, {
    "name" : "Policy Developer",
    "id" : "role-7"
  } ],
  "uniqueIdCounters" : {
    "User" : 2,
    "Role" : 8
  }
}
```

The credentials from this file are used to authenticate and perform RBAC on all accesses to the Management Services. This store holds the user credentials, so their passwords can be verified, and also holds their roles. Credentials and associated roles can also be retrieved from an LDAP Directory Server (for example, Microsoft Active Directory or OpenLDAP).

For details on configuring an LDAP repository, see the following topics:

- *Using Active Directory for Authentication and RBAC of Management Services*
- *Using OpenLDAP for Authentication and RBAC of Management Services*

Access Control List

The Access Control List file (`acl.json`) is located in the `conf` directory of your API Gateway installation. This file lists each role and the Management Services that each role may access. By default, this file defines the following roles:

- `API Service Developer`
- `API Service Administrator`
- `API Gateway Administrator`
- `API Gateway Operator`
- `KPS Administrator`
- `Policy Developer`
- `Deployer`

The default `admin` user is assigned the `API Service Developer`, `API Gateway Administrator`, `KPS Administrator`, and `Policy Developer` roles by default, which together allow access to everything. For full details on the Management Services that each role has access to, and the URIs that must be listed in the `acl.json` file to have access to them, see the table in the section called “Management Service Roles and Permissions”.

Important

The roles defined in the `acl.json` file should exist in the user store used to authenticate the users and load their roles and/or groups. The default roles are defined in the local Admin User store, which is used to control access to the Management Services using the **Protect Management Interfaces** policy. If a different user store is used (for example, an LDAP repository), the LDAP groups should be listed in the `acl.json` and `adminUsers.json` files .

Access Control List File Format

Each role entry in the `acl.json` file has the following format:

```
"role-name" : [ <list_of_permission_names> ]
```

The permissions consist of operations that are defined by HTTP methods and URIs:

```
“permission-name” : { <list_of_operation_names> }
“operation-name” : {
        "methods" : [ <list of HTTP Methods> ],
        "paths" : [ <list of path-names> ]
}

“path-name” : {
        "path" : <URI>
}
```

This file entry format is described as follows:

- The permissions line is repeated for each permission the role has. To determine which permissions should be listed for each Management Service, see the table in the section called “Management Service Roles and Permissions”.
- You can place a wildcard (`*`) at the end of the `path` field. For example, see the path for `dojo resources` in the example that follows. This means the role has access to all URIs that start with the URI content that precedes the `*`.
- In some cases, you must protect a Management Service by specifying a query string after the URI. Exact matches only are supported for query strings.

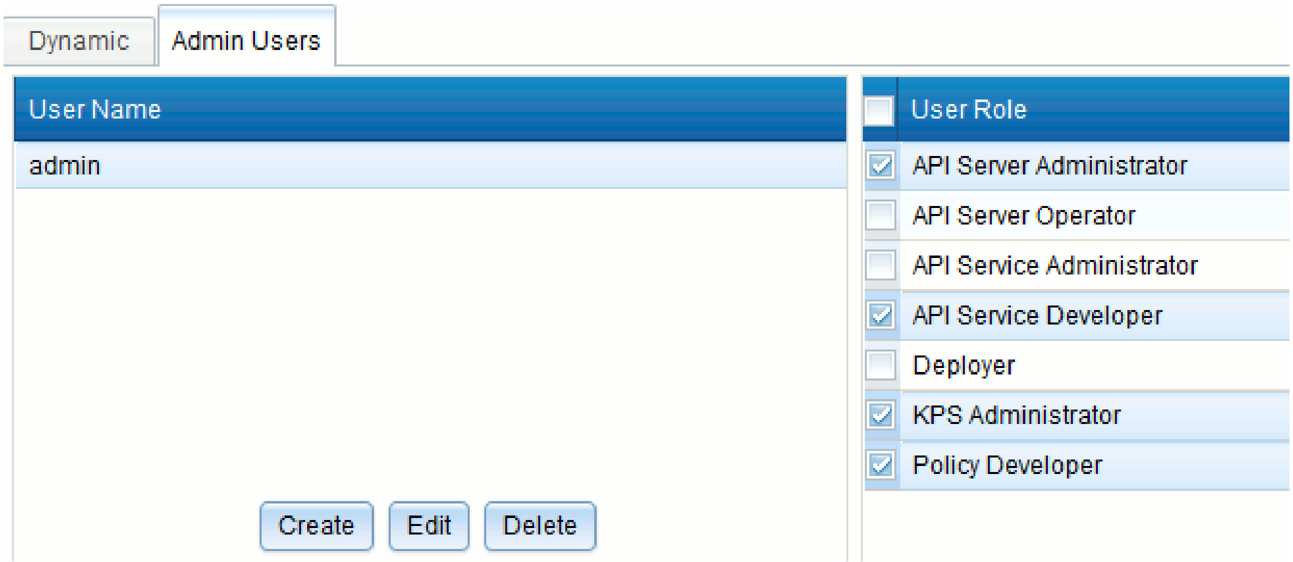
Example Access Control List File

The following example shows the roles and permissions to URIs:

```
"paths" : {
        "root" : { "path" : "/" },
        "emc pages" : { "path" : "/emc/*" },
        "site images" : { "path" : "/images/*" },
        "dojo resources" : { "path" : "/dojo/*" },
        ....
        }

},

"operations" : {
        "emc_read_web" : {
        "methods" : [ "GET" ],
        "paths" : [ "emc pages", "dojo resources" ]
        },

        "common_read_web" : {
        "methods" : [ "GET" ],
        "paths" : [ "root", "site images" ]
        },
        ....
},

"permissions" : {
        "emc" : [ "common_read_web", "emc_read_web" ],
        "config" : [ "configuration" ],
        "deploy" : [ "deployment", "management" ],
        "api_service_manager" : [ "servicemanager",
        "servicemanager.read", "management" ],
        "api_service_manager_modify" : [ "servicemanager.modify",
        "configuration" ]
        ...
},

“roles” : {
        "API Service Administrator" : [ "emc", "mgmt",
        "api_service_manager" ],
        "Policy Developer" : [ "deploy", "config", "ps"]
}
```

Configuring Users and Roles

You can use the API Gateway Manager to configure the users and roles in the local Admin User store. Click the **Settings** -> **Admin Users** to view and modify user roles (assuming you have a role that allows this). This screen is displayed as follows:



User Name
admin

☐	User Role
☑	API Server Administrator
☐	API Server Operator
☐	API Service Administrator
☑	API Service Developer
☐	Deployer
☑	KPS Administrator
☑	Policy Developer

Managing User Roles
When you click **Create** to create a new user, you can select the roles to assign to the that new user. New users are not assigned a default role. While users that are replicated from an LDAP repository do not require a role to be assigned to them. You can click **Edit** to changed the roles assigned to a selected user.

Adding a New Role to the User Store
When you add a new role to the Admin User store, you must modify the available roles in the `adminUsers.json` and `acl.json` files in the `conf` directory of your Admin Node Manager installation. You must add the new role to the `roles` section of the `acl.json` file, which lists all the permissions that the new role may have.

Important

You must update the `acl.json` before you add the roles to the Admin User store. The RBAC policy object automatically reloads the `acl.json` file each time you add or remove a role in the Policy Studio.

When you update the `acl.json` file, you must restart the Admin Node Manager to reload the `acl.json` file. However, the Admin Node Manager does not need to be rebooted or refreshed if a user's roles change.

For more details on managing user roles, see the topic on *Managing Admin Users*.

Management Service Roles and Permissions

You can use the following table for reference purposes when making changes to the `acl.json` file. It defines each Management Service, and the default roles that have access to them. It also lists the URIs that must be listed in the `acl.json` to have access to the Management Service.

Management Service	Default Roles	Permissions
API Gateway Manager (`https://localhost:8090`)	• API Gateway Administrator • API Gateway Operator • API Service Developer • API Service Administrator	• `emc` • `mgmt`

Management Service	Default Roles	Permissions
API Service Manager (read-only access)	• API Gateway Administrator	• `emc` • `mgmt` • `api_service_manager`
API Service Manager (write access)	• API Gateway Developer	• `emc` • `mgmt` • `api_service_manager` • `api_service_manager_modify`
API Gateway Manager Dashboard	• API Gateway Administrator	• `emc` • `mgmt` • `mgmt_modify` • `dashboard` • `dashboard_modify` • `deploy` • `config`
API Gateway Manager Dashboard (read-only access)	• API Gateway Operator	• `emc` • `mgmt` • `dashboard` • `dashboard_modify`
API Gateway Manager Monitoring	• API Gateway Administrator • API Gateway Operator	• `emc` • `mgmt` • `monitoring` • `events` • `traffic_monitor` • `settings` • `settings_modify` • `logs`
API Gateway Manager Traffic	• API Gateway Administrator • API Gateway Operator	• `emc` • `mgmt` • `traffic_monitor`
API Gateway Manager Logs	• API Gateway Administrator • API Gateway Operator	• `emc` • `mgmt` • `logs`

Management Service	**Default Roles**	**Permissions**
API Gateway Manager Events	• API Gateway Administrator • API Gateway Operator	• `emc` • `mgmt` • `monitoring` • `events`
API Gateway Manager Settings	• API Gateway Administrator	• `emc` • `mgmt` • `mgmt_modify` • `settings` • `settings_modify`
API Gateway Manager Settings (read-only access)	• API Gateway Operator	• `emc` • `mgmt` • `settings`
Documentation	• API Gateway Administrator • API Gateway Operator • API Service Developer • API Service Administrator	• `emc` • `mgmt`
KPS	• KPS Administrator	• `mgmt` • `kps`
Policy Studio	• Policy Developer	• `mgmt` • `deploy` • `config` • `ps` • `ps_tagging`
API Server Configuration Deployment	• API Gateway Administrator • Policy Developer • Deployer	• `mgmt` • `deploy` • `config`
Configuration API	• Policy Developer	• `ps` • `ps_tagging`

Using Active Directory for Authentication and RBAC of Management Services

Overview

This topic explains how to use Local Directory Access Protocol (LDAP) to authenticate and perform Role-Based Access Control (RBAC) of Management Services. You can use the sample **Protect Management Interfaces (LDAP)** policy instead of the **Protect Management Interfaces** policy. This means that an LDAP repository is used instead of the local Admin User store for authentication and RBAC of users attempting to access Management Services. This topic describes how to configure the server to use an example Microsoft Active Directory LDAP repository.

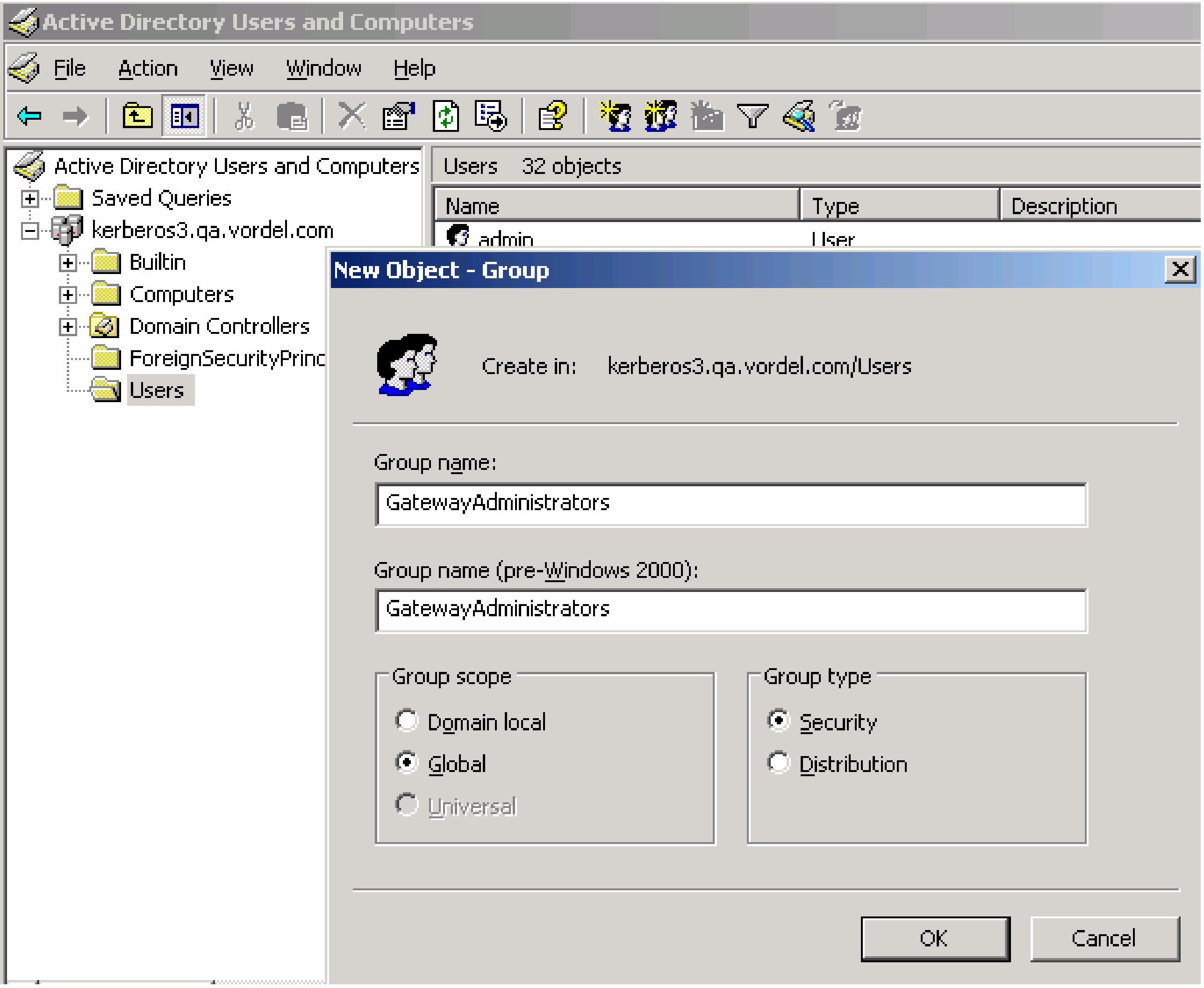
Note

To access the policies and settings described in this topic, you must have the **Show Management Services** setting enabled in the **Preferences** in the Policy Studio.

Step 1: Create an Active Directory Group

To create a new user group in Active Directory, perform the following example steps:

1. Click **Start** -> **Administrative Tools** -> **Active Directory Users and Computers**.
2. On the **Users** directory, right-click, and select **New** -> **Group**.
3. Enter the Group name (for example, `GatewayAdministrators`).



Create in: kerberos3.qa.vordel.com/Users

Group name: GatewayAdministrators

Group name (pre-Windows 2000): GatewayAdministrators

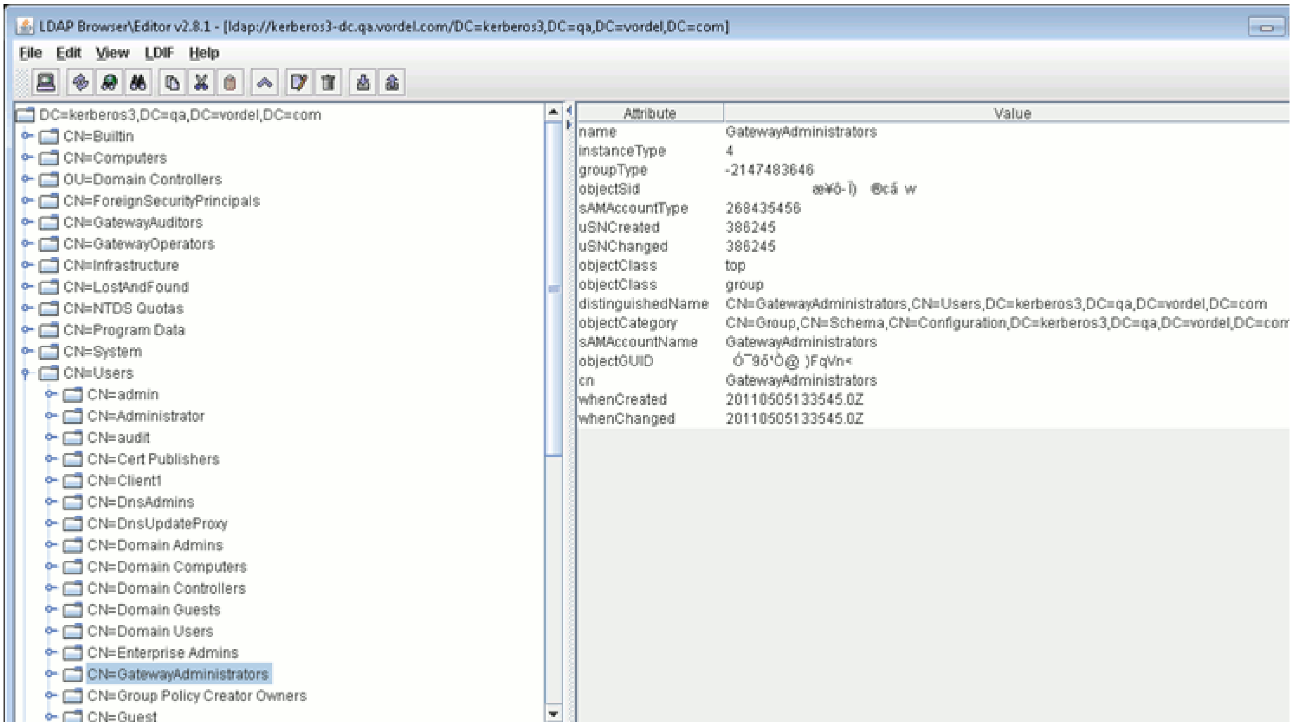
Group scope

○ Domain local
● Global
○ Universal

Group type

● Security
○ Distribution

You can view the new group using an LDAP Browser. For example:


LDAP Browser\Editor v2.8.1 - [ldap://kerberos3-dc.qa.vordel.com/DC=kerberos3,DC=qa,DC=vordel,DC=com]
File
Edit
View
LDIF
Help

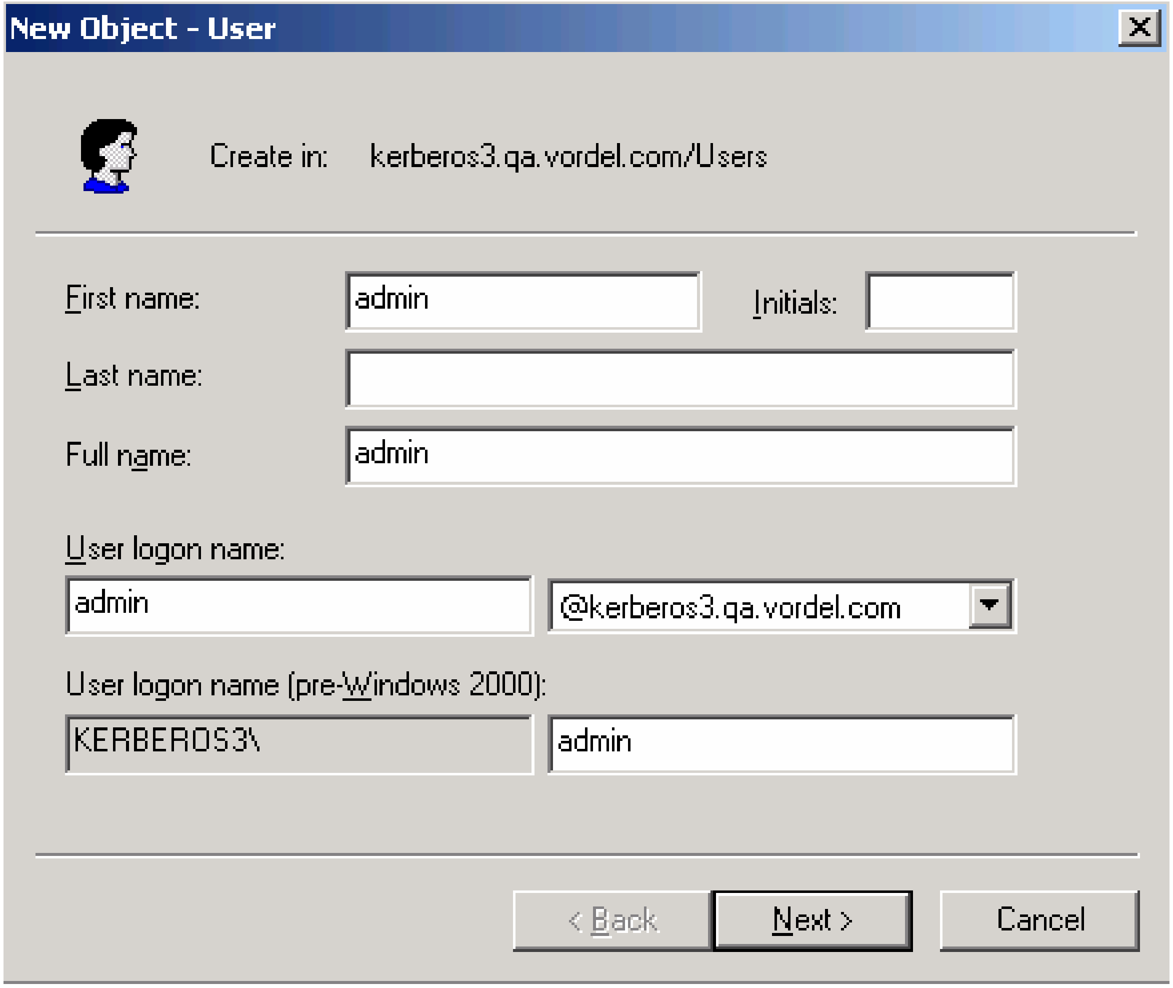
DC=kerberos3,DC=qa,DC=vordel,DC=com
- CN=Builtin
- CN=Computers
- OU=Domain Controllers
- CN=ForeignSecurityPrincipals
- CN=GatewayAuditors
- CN=GatewayOperators
- CN=Infrastructure
- CN=LostAndFound
- CN=NTDS Quotas
- CN=Program Data
- CN=System
- CN=Users
  - CN=admin
  - CN=Administrator
  - CN=audit
  - CN=Cert Publishers
  - CN=Client1
  - CN=DnsAdmins
  - CN=DnsUpdateProxy
  - CN=Domain Admins
  - CN=Domain Computers
  - CN=Domain Controllers
  - CN=Domain Guests
  - CN=Domain Users
  - CN=Enterprise Admins
  - CN=GatewayAdministrators
  - CN=Group Policy Creator Owners
  - CN=Guest

Attribute	Value
name	GatewayAdministrators
instanceType	4
groupType	-2147483646
objectSid	æ¥ő-Ì) ®cå w
sAMAccountType	268435456
uSNCreated	386245
uSNChanged	386245
objectClass	top
objectClass	group
distinguishedName	CN=GatewayAdministrators,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com
objectCategory	CN=Group,CN=Schema,CN=Configuration,DC=kerberos3,DC=qa,DC=vordel,DC=com
sAMAccountName	GatewayAdministrators
objectGUID	Ó¯9õ'Ò@)FqVn<
cn	GatewayAdministrators
whenCreated	20110505133545.0Z
whenChanged	20110505133545.0Z

Step 2: Create an Active Directory User

You will most likely be unable to create an `admin` user with a password of `changeme` because this password is not strong enough to be accepted by Active Directory. Using **Active Directory Users and Computers**, perform the following steps:

1. On the **Users** directory, right-click, and select **New** -> **User**.
2. Enter a user name (for example, `admin`).



New Object - User

Create in: kerberos3.qa.vordel.com/Users

First name: admin

Initials:

Last name:

Full name: admin

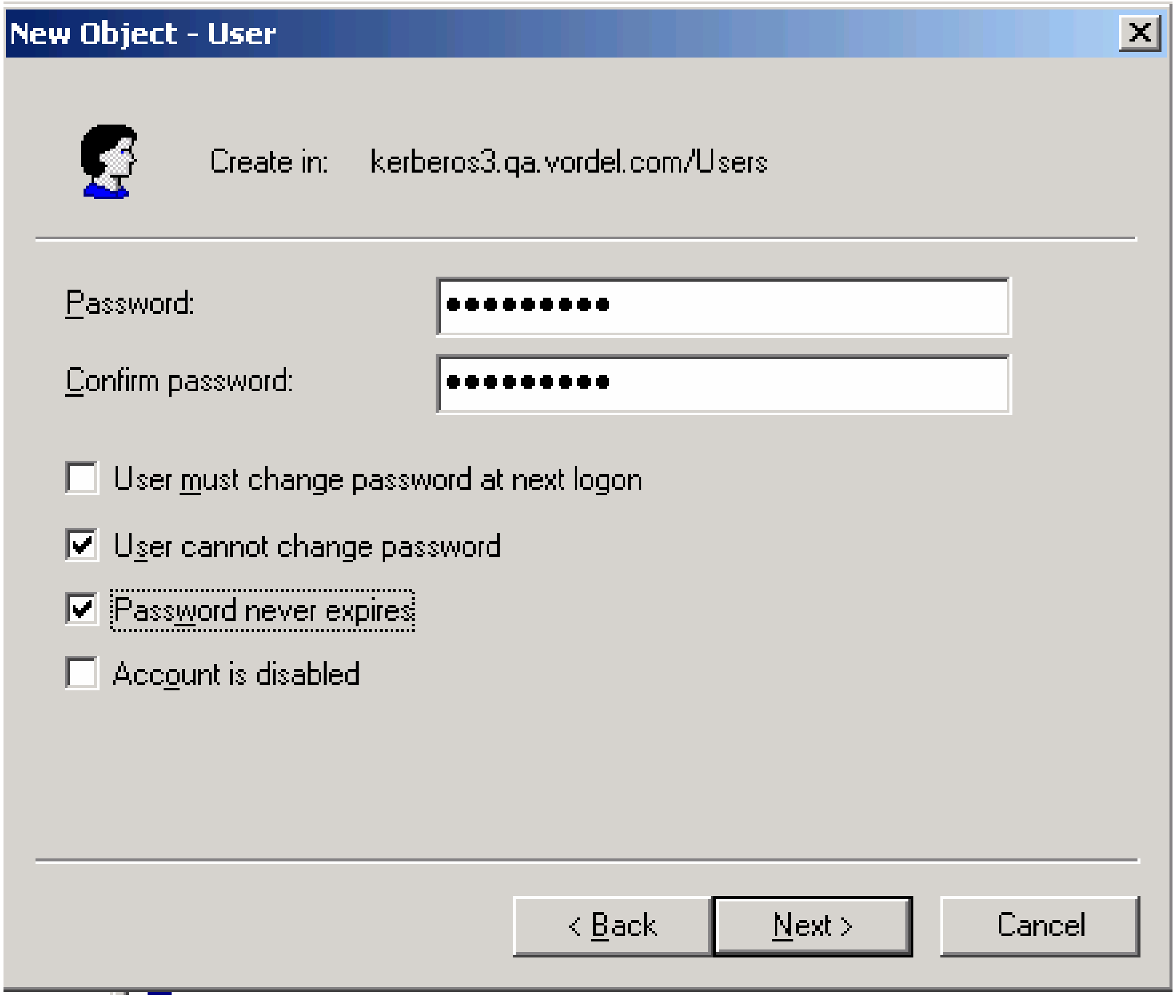
User logon name:

admin @kerberos3.qa.vordel.com

User logon name (pre-Windows 2000):

KERBEROS3\ admin

3. Click **Next**.
4. Enter a password (for example, `Oracle123`).
5. Select **User cannot change password** and **Password never expires**.
6. Ensure **User must change password at next logon** is not selected.
7. Click **Next**.
8. Click **Finish**.



Create in: kerberos3.qa.vordel.com/Users

Password: ●●●●●●●●●●

Confirm password: ●●●●●●●●●●

□ User must change password at next logon

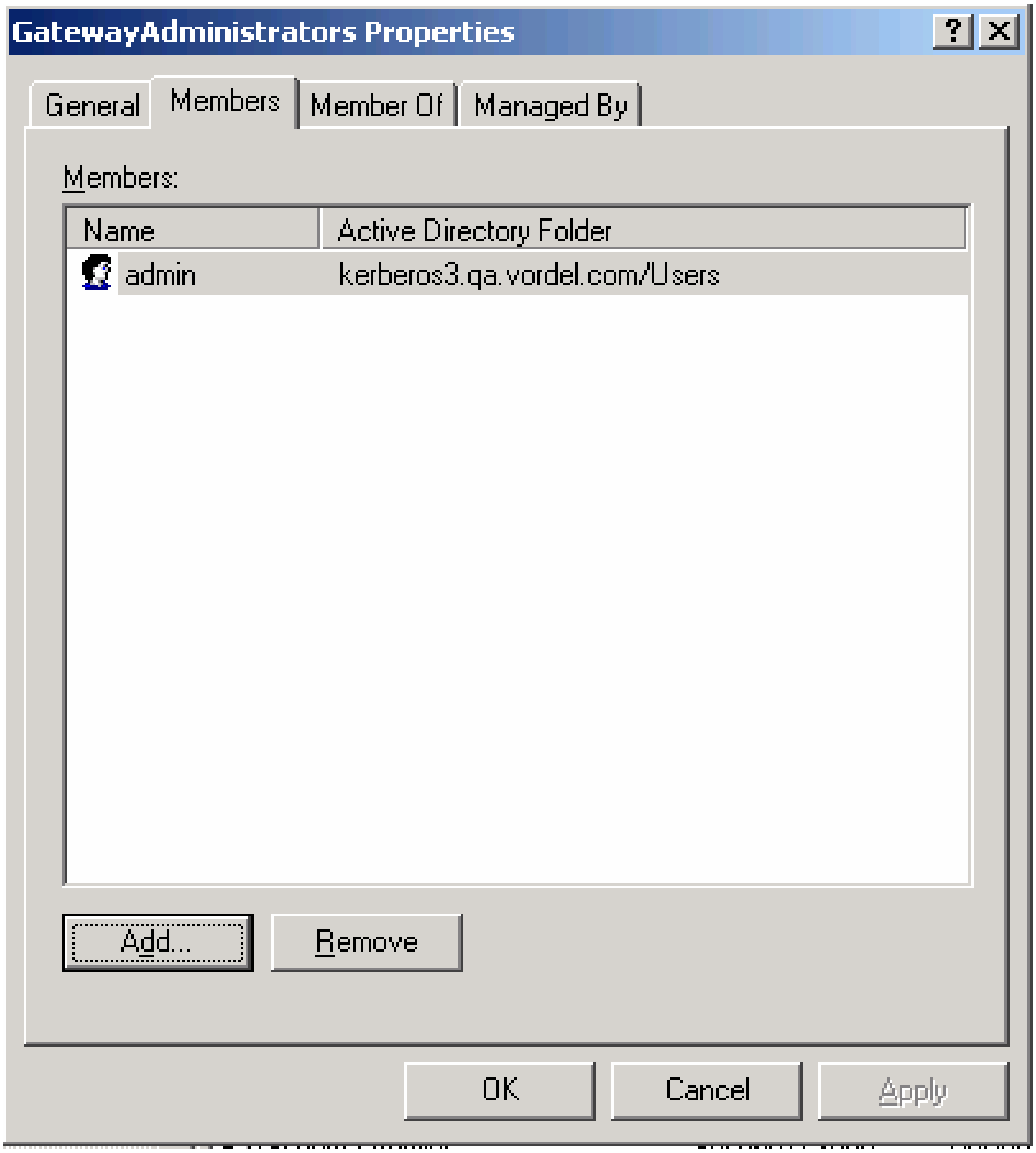
☑ User cannot change password

☑ Password never expires

□ Account is disabled

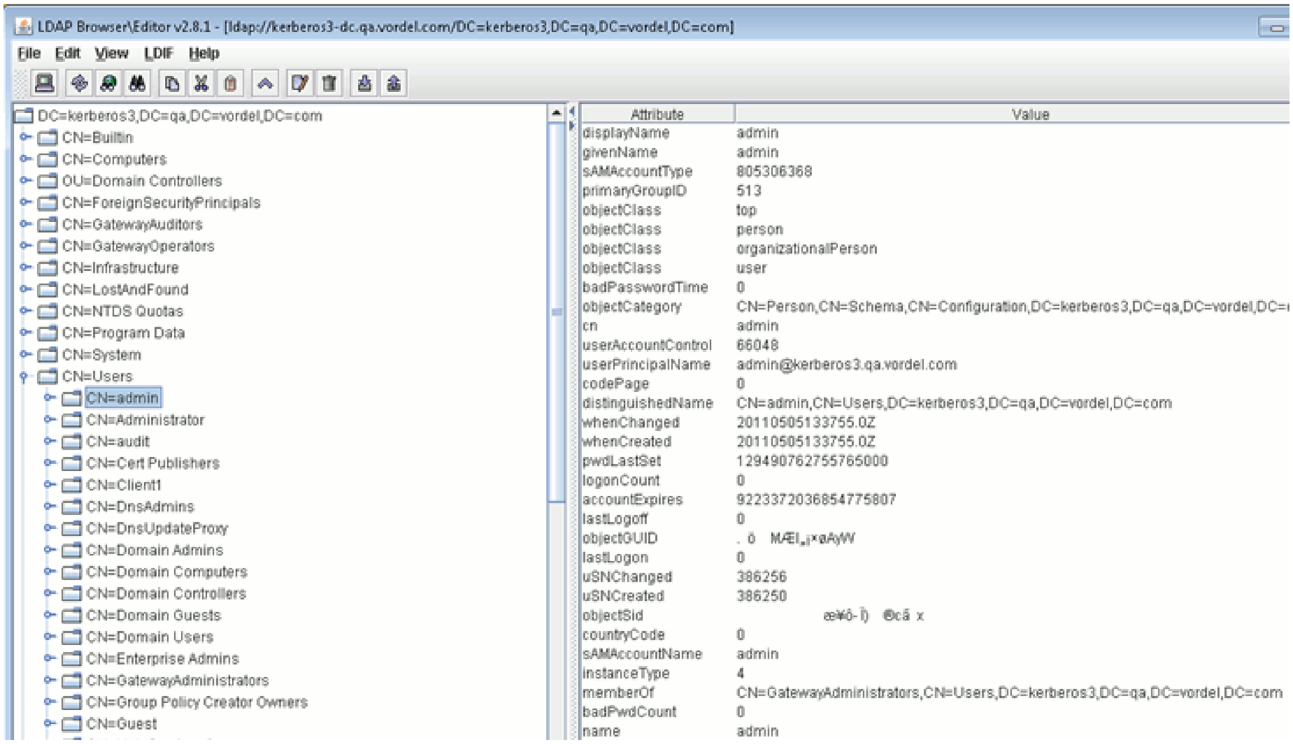
Adding the User to the Group
To make the user a member of the group using **Active Directory Users and Computers**, perform the following steps:

1. Select the **GatewayAdministrators** group, right-click, and select **Properties**.
2. Click the **Members** tab.
3. Click **Add**.
4. Click **Advanced**.
5. Click **Find Now**.
6. Select the `admin` user.
7. Click **OK**.



Name	Active Directory Folder
admin	kerberos3.qa.vordel.com/Users

You can view the new user using an LDAP Browser. For example:


LDAP Browser\Editor v2.8.1 - [ldap://kerberos3-dc.qa.vordel.com/DC=kerberos3,DC=qa,DC=vordel,DC=com]
File
Edit
View
LDIF
Help

DC=kerberos3,DC=qa,DC=vordel,DC=com
- CN=Builtin
- CN=Computers
- OU=Domain Controllers
- CN=ForeignSecurityPrincipals
- CN=GatewayAuditors
- CN=GatewayOperators
- CN=Infrastructure
- CN=LostAndFound
- CN=NTDS Quotas
- CN=Program Data
- CN=System
- CN=Users
  - CN=admin
  - CN=Administrator
  - CN=audit
  - CN=Cert Publishers
  - CN=Client1
  - CN=DnsAdmins
  - CN=DnsUpdateProxy
  - CN=Domain Admins
  - CN=Domain Computers
  - CN=Domain Controllers
  - CN=Domain Guests
  - CN=Domain Users
  - CN=Enterprise Admins
  - CN=GatewayAdministrators
  - CN=Group Policy Creator Owners
  - CN=Guest

Attribute	Value
displayName	admin
givenName	admin
sAMAccountType	805306368
primaryGroupID	513
objectClass	top
objectClass	person
objectClass	organizationalPerson
objectClass	user
badPasswordTime	0
objectCategory	CN=Person,CN=Schema,CN=Configuration,DC=kerberos3,DC=qa,DC=vordel,DC=
cn	admin
userAccountControl	66048
userPrincipalName	admin@kerberos3.qa.vordel.com
codePage	0
distinguishedName	CN=admin,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com
whenChanged	20110505133755.0Z
whenCreated	20110505133755.0Z
pwdLastSet	129490762755765000
logonCount	0
accountExpires	9223372036854775807
lastLogoff	0
objectGUID	. ö MÆI„¡×øAyW
lastLogon	0
uSNChanged	386256
uSNCreated	386250
objectSid	æ¥ô-Ï) ®cå x
countryCode	0
sAMAccountName	admin
instanceType	4
memberOf	CN=GatewayAdministrators,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com
badPwdCount	0
name	admin

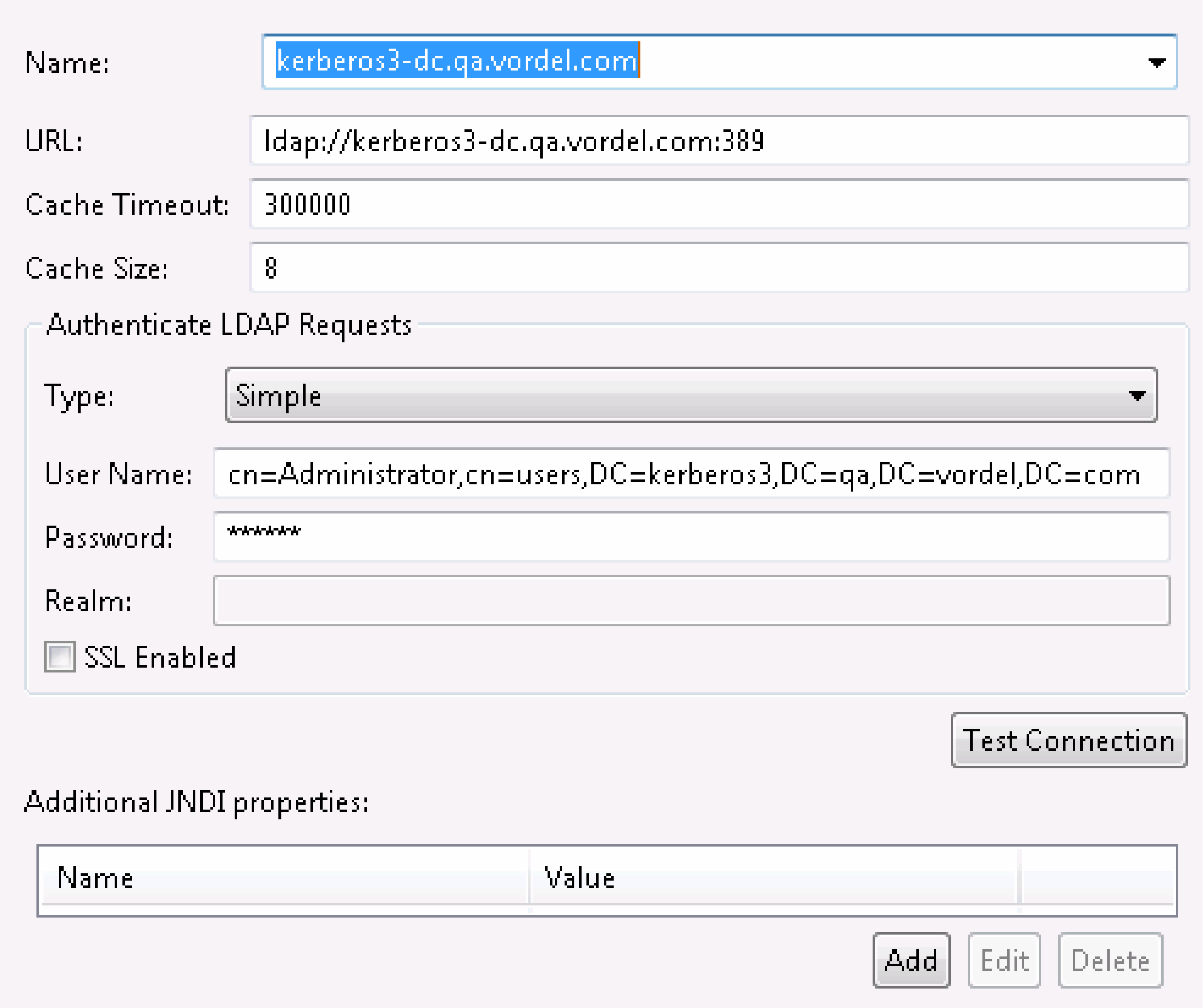
Note

The `memberOf` attribute points to the Active Directory group. The user has an instance of this attribute for each group they are a member of.

Step 3: Create an LDAP Connection

To create an new LDAP Connection, perform the following steps:

1. In the Policy Studio, select **Open File** , and select the Admin Node Manager configuration file (for example, `INSTALL_DIR\apigateway\conf\fed\configs.xml`.
2. In the Policy Studio tree, select **External Connections** -> **LDAP Connections**.
3. Right-click, and select **Create an LDAP Connection**.
4. Complete the fields in the dialog as appropriate. The specified **User Name** should be an LDAP administrator that has access to search the full directory for users. For example:



Name: kerberos3-dc.qa.vordel.com

URL: ldap://kerberos3-dc.qa.vordel.com:389

Cache Timeout: 300000

Cache Size: 8

Authenticate LDAP Requests

Type: Simple

User Name: cn=Administrator,cn=users,DC=kerberos3,DC=qa,DC=vordel,DC=com

Password: *******

Realm:

□ SSL Enabled

5. Click **Test Connection** to ensure the connection details are correct.

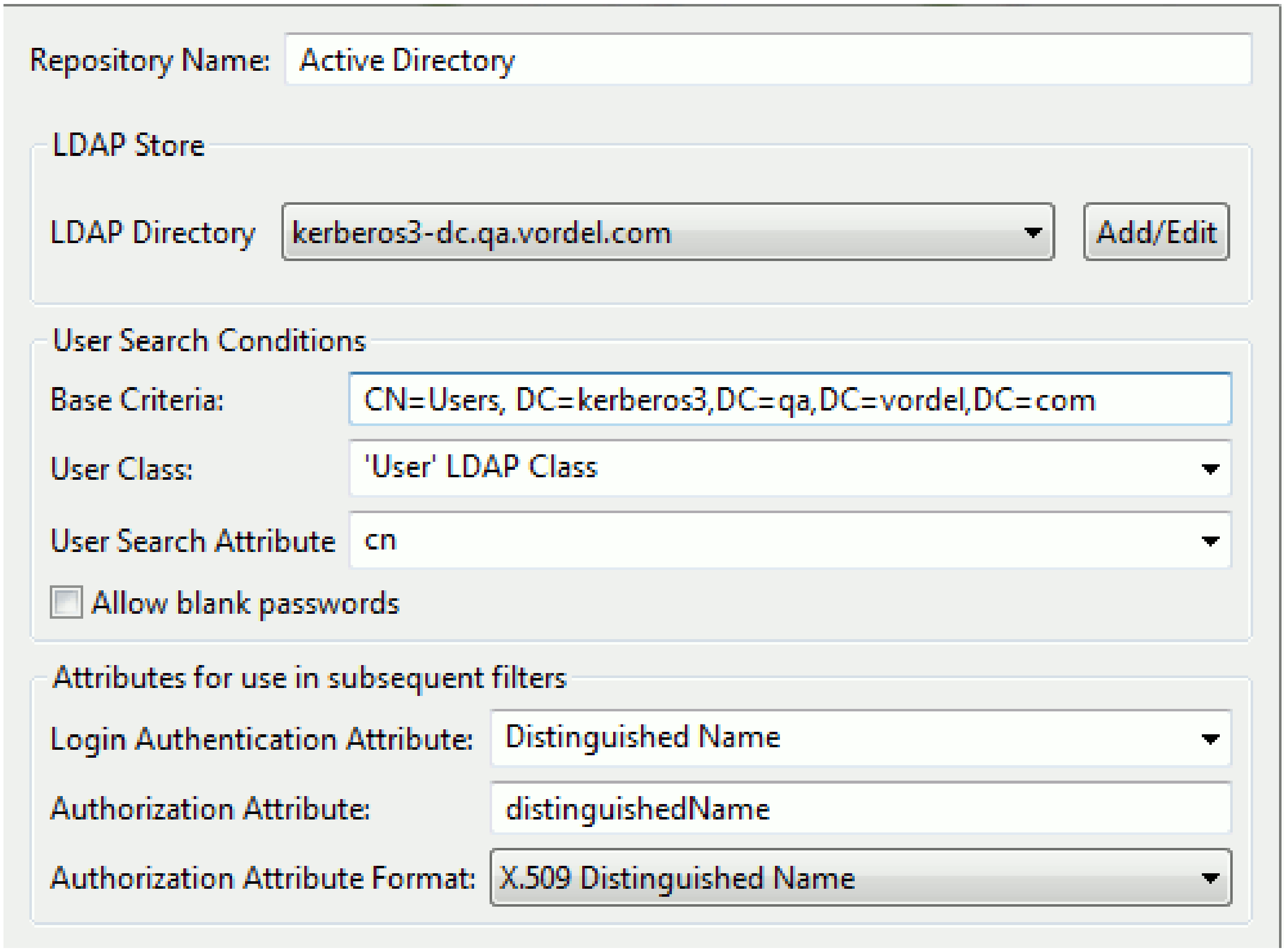
Step 4: Create an LDAP Repository

To create an new LDAP Repository, perform the following steps:

1. In the Policy Studio tree, select **External Connections** -> **Authentication Repository Profiles** -> **LDAP Repositories**.
2. Right-click, and select **Add a new Repository**.
3. Complete the following fields in the dialog:

Repository Name	Enter an appropriate name for the repository.
LDAP Directory	Use the LDAP directory created in the section called “Step 3: Create an LDAP Connection”.
Base Criteria	Enter the LDAP node that contains the users (for example, see the LDAP Browser screen shot in the section called “Step 2: Create an Active Directory User”).

User Search Attribute	Enter cn. This is the username entered at login time (in this case, admin).
Authorization Attribute	Enter distinguishedName. This is the username entered at login time (admin). The authentication.subject.id message attribute is set to the value of this LDAP attribute (for example, CN=admin,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com. The authentication.subject.id is used as the base criteria in the filter that loads the LDAP groups (the user's roles). This enables you to narrow the search to a particular user node in the LDAP tree. For more details, see the **Retrieve Attributes from Directory Server** filter in the section called "Step 5: Create a Test Policy for LDAP Authentication and RBAC".



Repository Name: Active Directory

LDAP Store

LDAP Directory: kerberos3-dc.qa.vordel.com

User Search Conditions

Base Criteria: CN=Users, DC=kerberos3,DC=qa,DC=vordel,DC=com

User Class: 'User' LDAP Class

User Search Attribute: cn

□ Allow blank passwords

Attributes for use in subsequent filters

Login Authentication Attribute: Distinguished Name

Authorization Attribute: distinguishedName

Authorization Attribute Format: X.509 Distinguished Name

Connecting to Other LDAP Repositories

This topic uses Microsoft Active Directory as an example LDAP repository. Other LDAP repositories such as Oracle Directory Server (formerly iPlanet and Sun Directory Server) and OpenLDAP are also supported. For an example of querying an Oracle Directory Server repository, see the **Retrieve Attributes from Directory Server** filter in the section called "Step 5: Create a Test Policy for LDAP Authentication and RBAC". For details on using OpenLDAP, see *Using OpenLDAP for Authentication and RBAC of Management Services*.

Step 5: Create a Test Policy for LDAP Authentication and RBAC

To avoid locking yourself out of the Policy Studio, you can create a test policy for LDAP authentication and RBAC, which is invoked when a test URI is called on the server (and not a Management Services URI). For an example policy, select **Policies** -> **Management Services** -> **Sample LDAP Policies** -> **Protect Management Interfaces (LDAP)** when the Admin Node Manager configuration is loaded.

Create the Test Policy

Perform the following steps:

1. Select **Open File** and load the Admin Node Manager configuration file in the Policy Studio. For example:

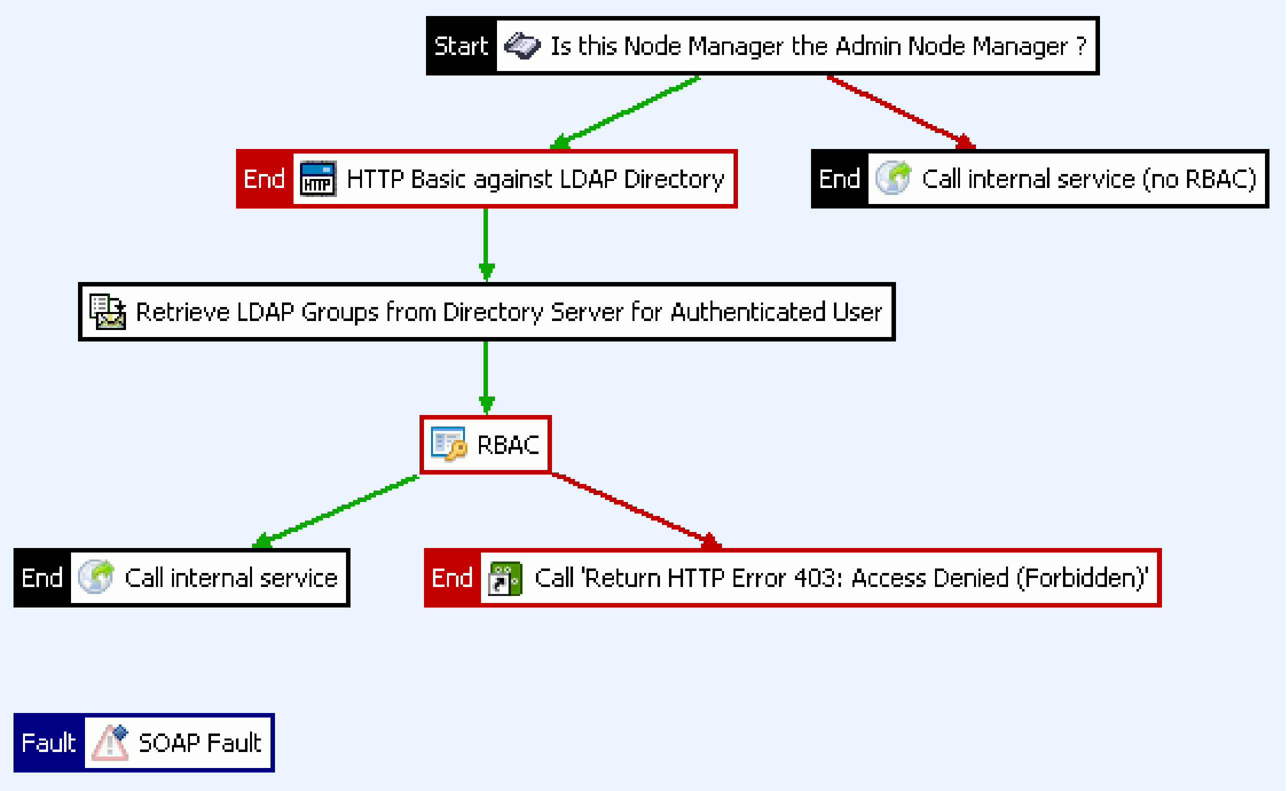
```
INSTALL_DIR/apigateway/conf/fed/configs.xml
```

2. Right-click the **Policies** node in the tree view of the Policy Studio, and select **Add Policy**.
3. Enter a suitable name (for example `Test Policy`) for the new policy in the **Name** field, and click **OK**.
4. Click the new policy in the tree to start configuring the policy filters. You can configure the policy by dragging the required filters from the filter palette on the right, and dropping them on to the policy canvas.

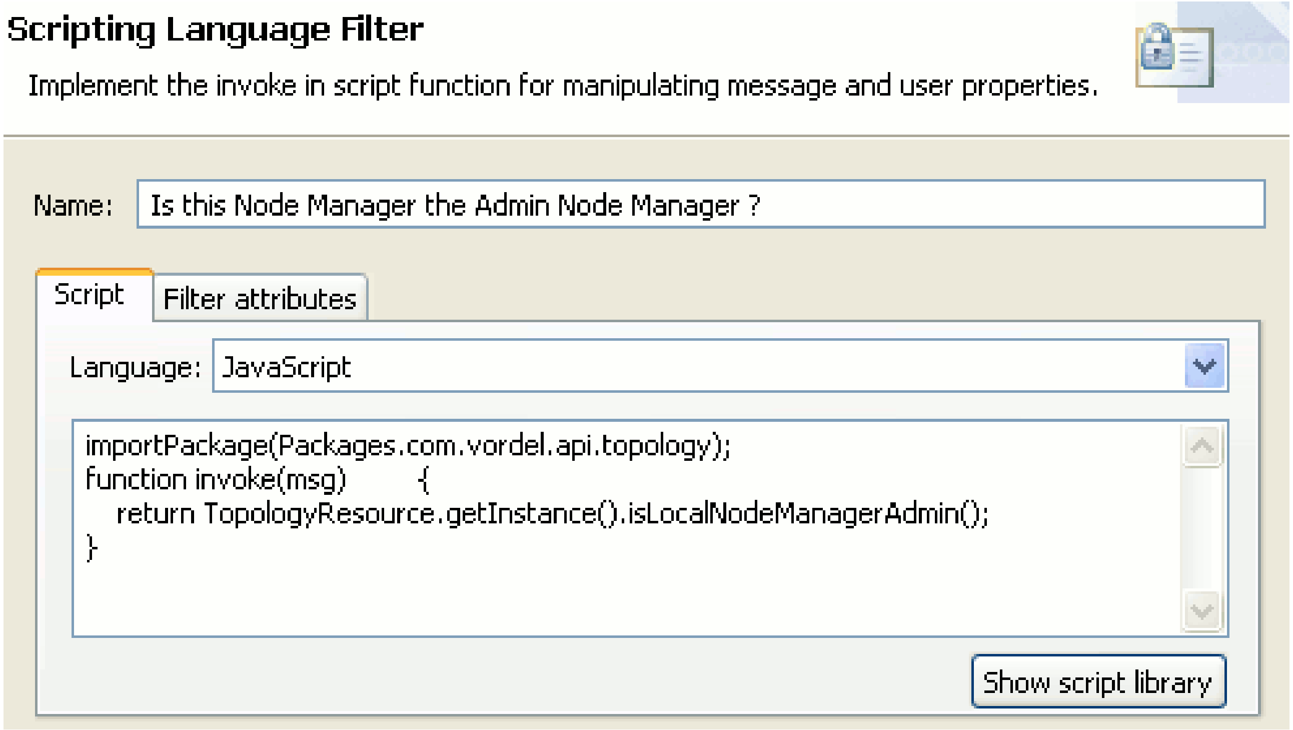
For more details, see the topic on *Configuring Policies Manually*.

Configure the Test Policy

Configure the test policy with the following filters:


Start
Is this Node Manager the Admin Node Manager ?
End
HTTP Basic against LDAP Directory
End
Call internal service (no RBAC)
Retrieve LDAP Groups from Directory Server for Authenticated User
RBAC
End
Call internal service
End
Call 'Return HTTP Error 403: Access Denied (Forbidden)'
Fault
SOAP Fault

Scripting Language Filter
This includes the following settings:



Scripting Language Filter

Implement the invoke in script function for manipulating message and user properties.

Name: Is this Node Manager the Admin Node Manager ?

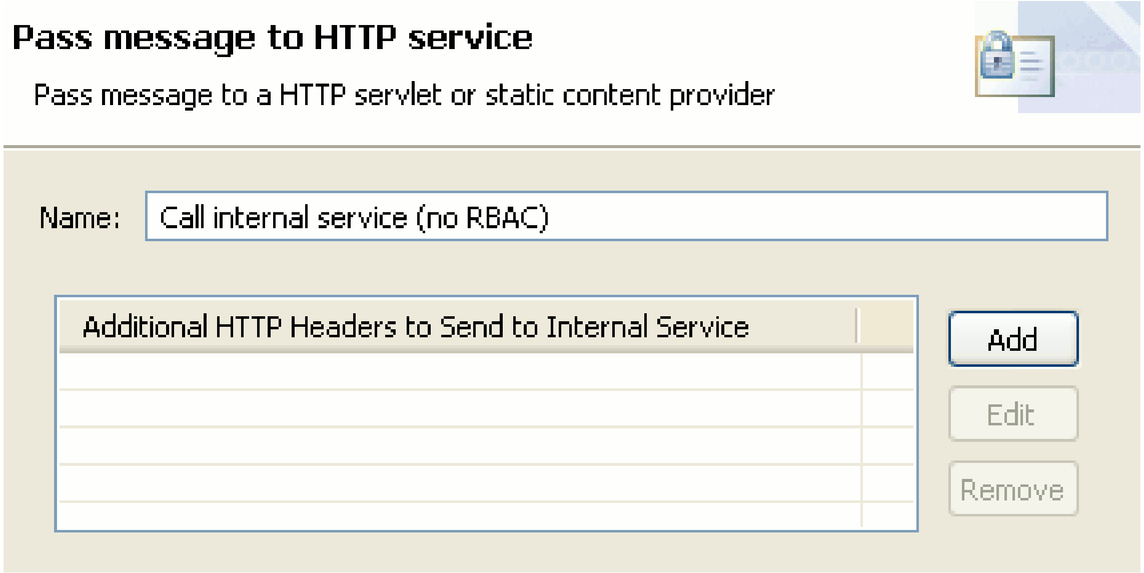
Language: JavaScript

```
importPackage(Packages.com.vordel.api.topology);
function invoke(msg)        {
   return TopologyResource.getInstance().isLocalNodeManagerAdmin();
}
```

The **Scripting Language Filter** performs the following tasks:

- Returns true if the Node Manager is the Admin Node Manager and passes control to the **HTTP Basic Authentication** filter.
- Otherwise, calls the **Call Internal Service (no RBAC)** filter without adding the `authentication.subject.role` and `authentication.subject.id` HTTP headers.

Call Internal Service (no RBAC) Filter
This filter is called without adding any HTTP headers as follows:



Pass message to HTTP service

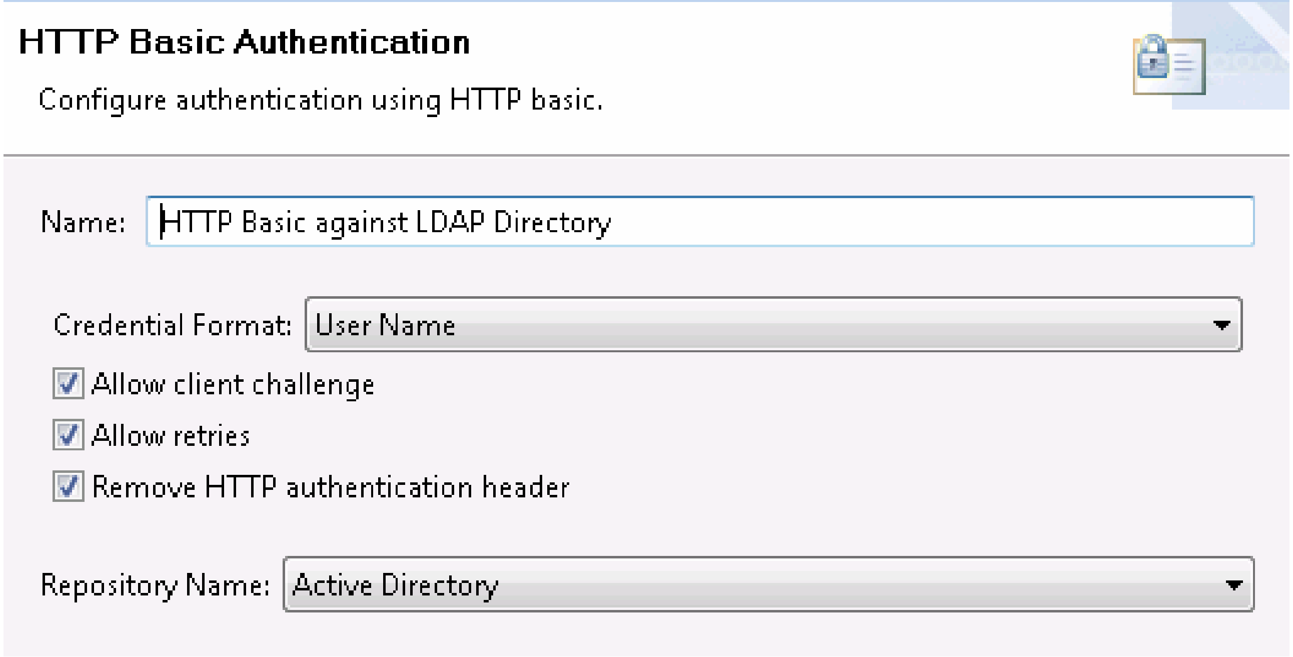
Pass message to a HTTP servlet or static content provider

Name: Call internal service (no RBAC)

Additional HTTP Headers to Send to Internal Service

HTTP Basic Authentication Filter

This filter uses the LDAP repository configured in the section called “Step 4: Create an LDAP Repository”, and includes the following settings:



HTTP Basic Authentication

Configure authentication using HTTP basic.

Name: HTTP Basic against LDAP Directory

Credential Format: User Name

☑ Allow client challenge

☑ Allow retries

☑ Remove HTTP authentication header

Repository Name: Active Directory

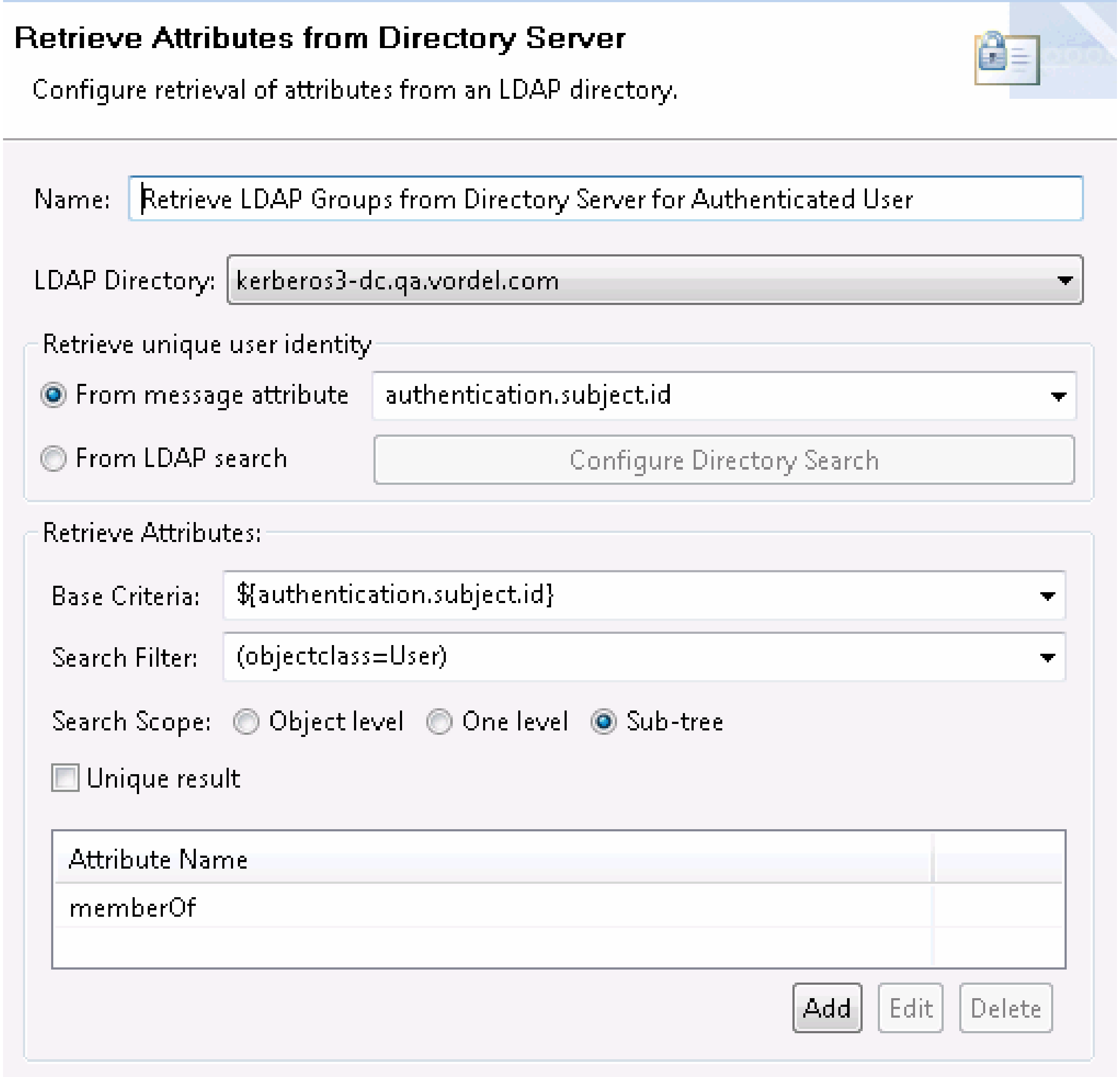
The **HTTP Basic Authentication** filter performs the following tasks:

- Connects to the LDAP directory using the connection details specified in the LDAP directory.
- Finds the user using the specified base criteria and search filter.
- If the user is found, verifies the user's name and password against the LDAP repository by performing a bind.
- If authentication fails, always throws a 401. This allows retry for browser users.

- The `distinguishedName` (DName) is held in the `authentication.subject.id` message attribute. This is specified by the **Authorization Attribute** field in the LDAP repository configuration.
- The user's roles (LDAP groups) are not available yet.

Retrieve Attributes from Directory Server Filter

This filter uses the LDAP directory configured in the section called "Step 3: Create an LDAP Connection", and includes the following settings:



Retrieve Attributes from Directory Server

Configure retrieval of attributes from an LDAP directory.

Name: Retrieve LDAP Groups from Directory Server for Authenticated User

LDAP Directory: kerberos3-dc.qa.vordel.com

Retrieve unique user identity

● From message attribute: authentication.subject.id

○ From LDAP search

Retrieve Attributes:

Base Criteria: ${authentication.subject.id}

Search Filter: (objectclass=User)

Search Scope: ○ Object level ○ One level ● Sub-tree

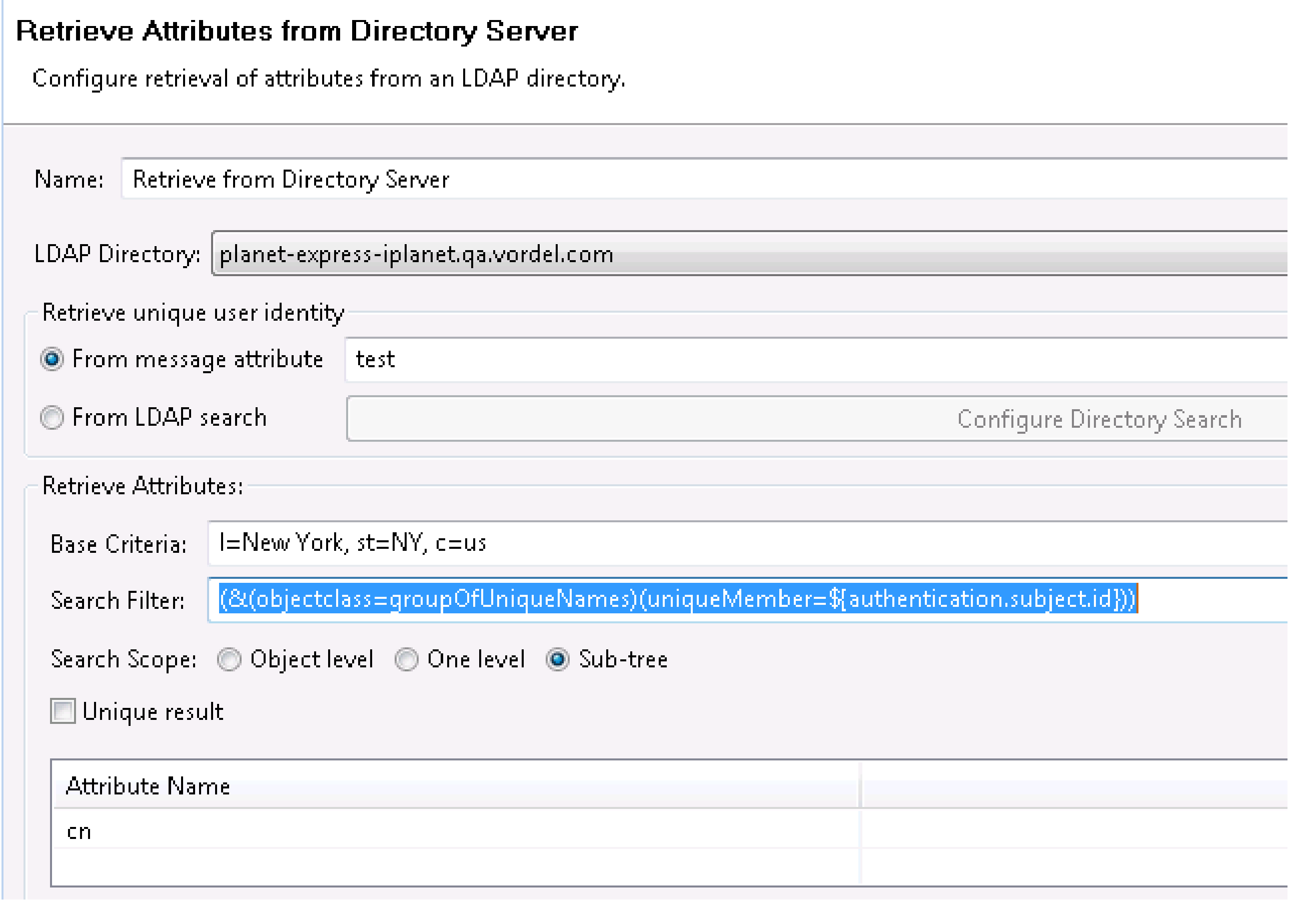
□ Unique result

Attribute Name
memberOf

The **Retrieve Attributes from Directory Server** filter performs the following tasks:

- Using the user's DName as the search start point, find the user's `memberOf` attribute, and load the LDAP groups for the user.
- If the user is in one group, the group name is contained in the `user.memberOf` message attribute. If the user is in multiple (n) LDAP groups, the group names are held in `user.memberOf.1`...`user.memberOf.n` message attributes.

Alternatively, the following screen shows an example of querying an Oracle Directory Server repository, where the query returns the authenticated user's groups instead of the user object:



Retrieve Attributes from Directory Server

Configure retrieval of attributes from an LDAP directory.

Name: Retrieve from Directory Server

LDAP Directory: planet-express-iplanet.qa.vordel.com

Retrieve unique user identity

● From message attribute: test

○ From LDAP search

Retrieve Attributes:

Base Criteria: l=New York, st=NY, c=us

Search Filter: (&(objectclass=groupOfUniqueNames)(uniqueMember=${authentication.subject.id}))

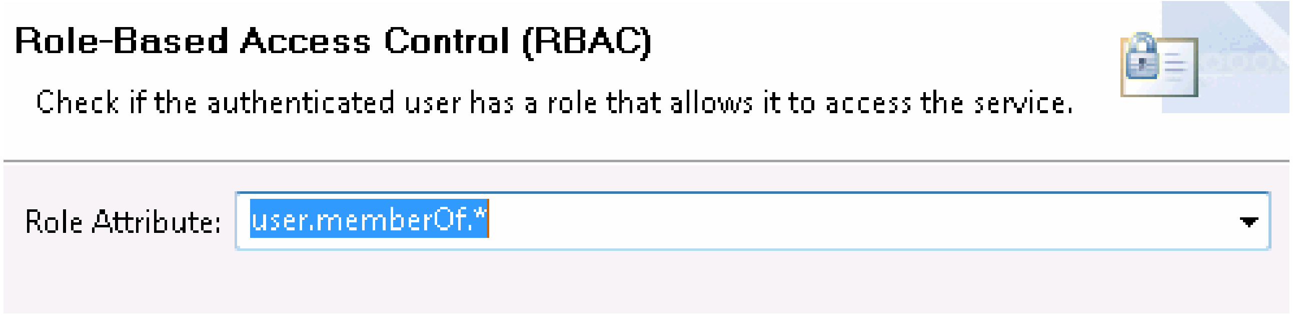
Search Scope: ○ Object level ○ One level ● Sub-tree

□ Unique result

Attribute Name	
cn	

You should be able to query any LDAP directory in this way. Assuming that the user's groups or roles can be retrieved as attributes of an object, the query does not need to be for the user object.

RBAC Filter
This filter includes a the following setting:



Role-Based Access Control (RBAC)

Check if the authenticated user has a role that allows it to access the service.

Role Attribute: user.memberOf.*

The **RBAC** filter performs the following tasks:

- Reads the roles from the `user.memberOf.*` message attribute. It understands the meaning of the wildcard, and loads the roles as required. It creates a string version of the roles, and places it in the `authentication.subject.role` message attribute for consumption by the **Call Internal Service** filter, which receives the roles as an HTTP header value.
- Determines which Management Service URI is currently being invoked.
- Returns true if one of the roles has access to the Management Service currently being invoked, as defined in the `acl.json file`.
- Otherwise, returns false and the **Return HTTP Error 403: Access Denied (Forbidden)** policy is called. This message content of this filter is shown when a valid user has logged into the browser, but their role(s) does not give them access to the URI they have invoked. For example, this occurs if a new user is created and they have not yet been assigned any roles.

Test the Policy Configuration

To test this policy configuration, perform the following steps:

1. Update the `acl.json` file with the new LDAP group as follows:

```
"CN=GatewayAdministrators,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com" : [
  "emc", "mgmt", "mgmt_modify", "dashboard", "dashboard_modify", "deploy",
  "config", "monitoring", "events", "traffic_monitor", "settings",
  settings_modify", "logs" "api_service_manager", "api_service_manager_modify",
  "ps", "ps_tagging" ]
```

2. Update the `adminUsers.json` file with the new role as follows:

```
{
  "name" : "CN=GatewayAdministrators,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com",
  "id" : "role-8"
}
```

   And increase the number of roles, for example:

```
"uniqueIdCounters" : {
    "Role" : 9,
    "User" : 2
},
```

3. In the Policy Studio tree, select **Listeners** -> **Node Manager** -> **Add HTTP Services**, and enter a service name (for example, `LDAP Test`).
4. Right-click the HTTP service, and select **Add Interface** -> **HTTP**.
5. Enter an available port to test the created policy (for example, `8888`), and click **OK**.
6. Right-click the HTTP service, and select **Add Relative Path**.
7. Enter a relative path (for example, `/test`).

8. Set the **Path Specify Policy** to the **Protect Management Interfaces (LDAP)** policy, and click **OK**.
9. Close the connection to the Admin Node Manager file, and restart the Admin Node Manager so it loads the updated configuration.
10. Use API Gateway Explorer to call `http://localhost:8080/test`.
11. Enter the HTTP Basic credentials (for example, username `admin` and password `Oracle123`). If authentication is passed, the Admin Node Manager should return an HTTP 404 code (not found).

Important

Do not use the **Admin Users** tab in the API Gateway Manager to manage user roles because these are managed in LDAP.

Step 6: Use the LDAP Policy to Protect Management Services

If the authentication and RBAC filters pass, you can now use this policy to protect the management interfaces. To ensure that you do not lock yourself out of the server, perform the following steps:

1. Make a copy of the `conf/fed` directory contents from the server installation, and put it into a directory accessible from the Policy Studio.
2. Make another backup copy of the `conf/fed` directory, which will remain unmodified.
3. In the Policy Studio, select **File** -> **Open**, and browse to `configs.xml` in the first copy of the `fed` directory.
4. Under the **Listeners** -> **Management Services** node, select the `/` and the `/configuration/deployments` relative paths, and set the **Path Specify Policy** to the **Protect Management Interfaces (LDAP)** policy.
5. Remove the previously created `LDAP Test` HTTP Services.
6. Close the connection to the file.
7. Copy the `fed` directory back to the Admin Node Manager's `conf` directory.
8. Reboot the Admin Node Manager.
9. Start the Policy Studio, and connect to the Admin Node Manager using `admin` and password `Oracle123` (the LDAP user credentials). You should now be able to edit API Gateway configurations as usual.

Adding an LDAP User with Limited Access to Management Services

You can add an LDAP user with limited access to Management Services. For example, assume there is already a user named `Fred` defined in Active Directory. `Fred` has the following DName:

```
CN=Fred,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com
```

`Fred` belongs to an existing LDAP group called `TraceAnalyzers`. He can also belong to other LDAP groups that have no meaning for RBAC in the API Gateway. The `TraceAnalyzers` LDAP group has the following DName:

```
CN=TraceAnalyzers,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com
```

The user `Fred` should be able to read server trace files in a browser. No other access to Management Services should be given to `Fred`.

Adding Limited Access Rights
You must perform the following steps to allow `Fred` to view the trace files:

1. Add the following entry in the `roles` section in the `acl.json` file:

```
"CN=TraceAnalyzers,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com" :
        [ "emc", "mgmt", "logs" ]
```

2. Update the `adminUsers.json` file with the new role as follows:

   ```
   {
     "name" : "CN=TraceAnalyzers,CN=Users,DC=kerberos3,DC=qa,DC=vordel,DC=com",
     "id" : "role-8"
   }]
   ```

   And increase the number of roles, for example:

   ```
   "uniqueIdCounters" : {
       "Role" : 9,
       "User" : 2
   },
   ```

3. Restart the Admin Node Manager so that the `acl.json` and `adminUsers.json`file updates are picked up.
4. Enter the following URL in your browser:
   `http://localhost:8090/`
5. Enter user credentials for `Fred` when prompted in the browser.
6. The API Gateway Manager displays a **Logs** tab enabling access to the trace files that `Fred` can view.

Note

`Fred` is not allowed to access the server APIs used by the Policy Studio. If an attempt is made to connect to the server using the Policy Studio with his credentials, an `Access denied` error is displayed. No other configuration is required to give user `Fred` the above access to the Management Services. Other users in the same LDAP group can also view trace files without further configuration changes because the LDAP group is already defined in the `acl.json` file.

Using OpenLDAP for Authentication and RBAC of Management Services

Overview

This topic explains how to use Local Directory Access Protocol (LDAP) to authenticate and perform Role-Based Access Control (RBAC) of Management Services. You can use the sample **Protect Management and Interfaces (LDAP)** policy instead of the **Protect Management Interfaces** policy. This means that an LDAP repository is used instead of the local Admin User store for authentication and role-based access control (RBAC) of users attempting to access the Management Services. This topic describes how to reconfigure the server to use OpenLDAP as the LDAP repository, and to use the Apache Directory Studio as an LDAP browser.

Prerequisites
This example assumes that you already have configured connection to the OpenLDAP server and setup your organization groups and users that you wish to use to perform RBAC. For example:

- **LDAP URL**: `ldap://openldap.qa.oracle.com:389`
- **User**: `cn=admin,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE`
- **Password**: `oracle`

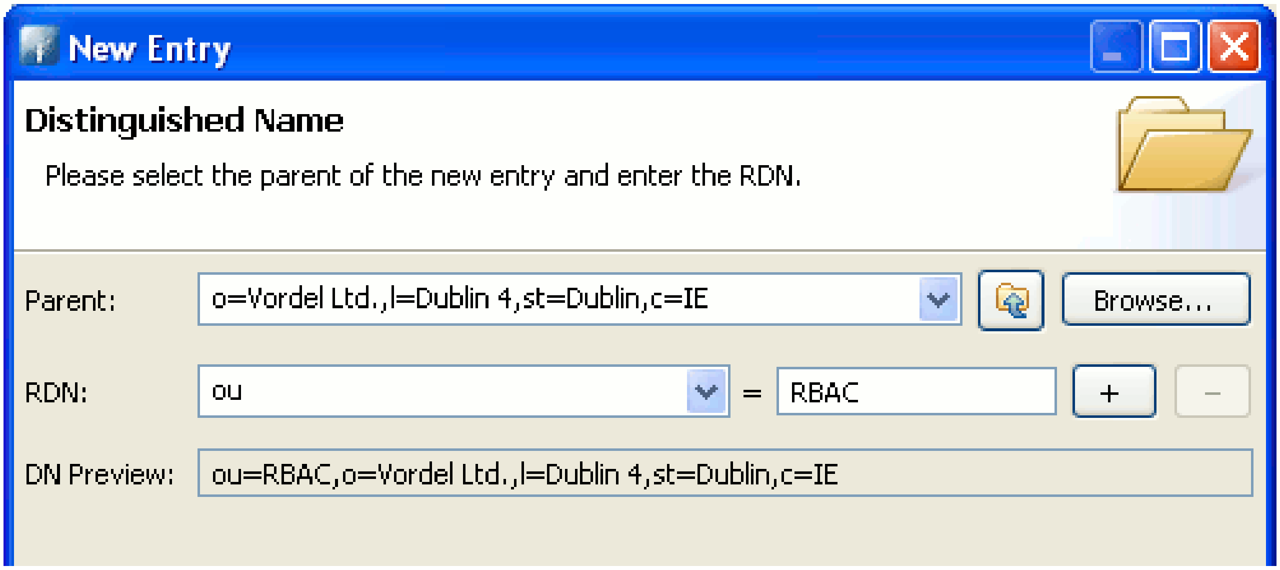
Important

To access the policies and settings described in this topic, you must have the **Show Management Services** setting enabled in the **Preferences** in the Policy Studio.

Step 1: Create an OpenLDAP Group for RBAC Roles

To create a new user group in OpenLDAP, perform the following steps:

1. Select the `cn=admin,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE` directory.
2. Right-click, and select **New** -> **New Entry**.
3. Select **Create entry from scratch**.
4. Click **Next**.
5. Add an `organizationalUnit` object class.
6. Click **Next**.
7. Set the **Parent** to `o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE`.
8. Set the **RDN** to `ou = RBAC`.
9. Click **Next**.
10. Click **Finish**.



Distinguished Name

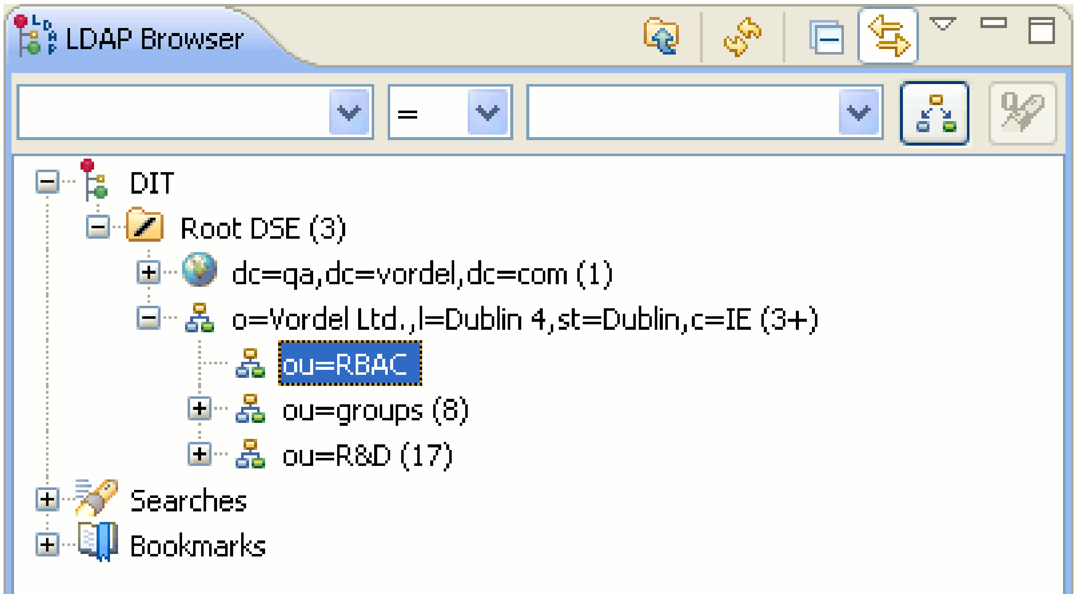
Please select the parent of the new entry and enter the RDN.

Parent: o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

RDN: ou = RBAC

DN Preview: ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

You can view the new group using an LDAP Browser. For example:



Step 2: Add RBAC Roles to the OpenLDAP RBAC Group

You must add the following default RBAC roles to the `ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE` group to give the LDAP users appropriate access to the API Gateway Management Services:

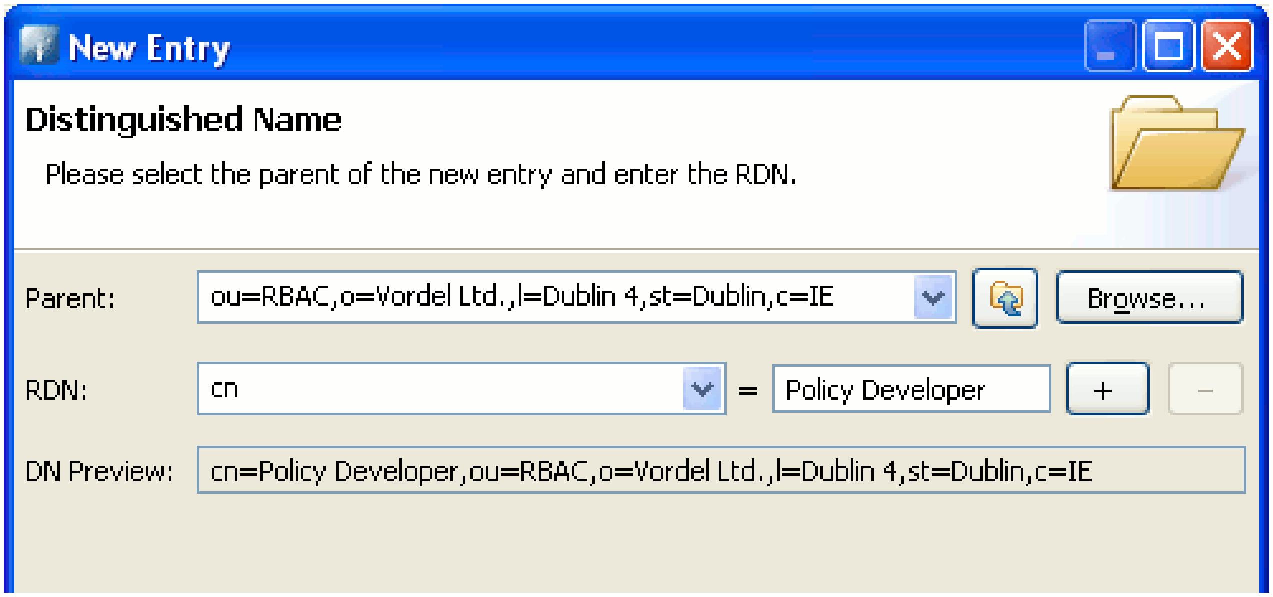
- `API Service Developer`
- `API Service Administrator`
- `API Gateway Administrator`
- `API Gateway Operator`
- `KPS Administrator`

- Policy Developer
- Deployer

These RBAC roles are located in the roles section of the acl.json file.

Adding Roles to the RBAC Directory
To add these RBAC roles to the OpenLDAP RBAC group, perform the following steps:

1. Select the cn=admin,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE directory.
2. Right-click, and select **New** -> **New Entry**.
3. Select **Create entry from scratch**.
4. Click **Next**.
5. Add a groupOfNames object class.
6. Click **Next**.
7. Set the **Parent** to ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE.
8. Set the **RDN** to ou = Policy Developer.
9. Click **Next**.
10. In the **DN Editor** dialog, set admin as first member of the following group: cn=admin,ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE. You can change the member Distinguished Name at any time.
11. Click **OK**.
12. Click **Finish**.

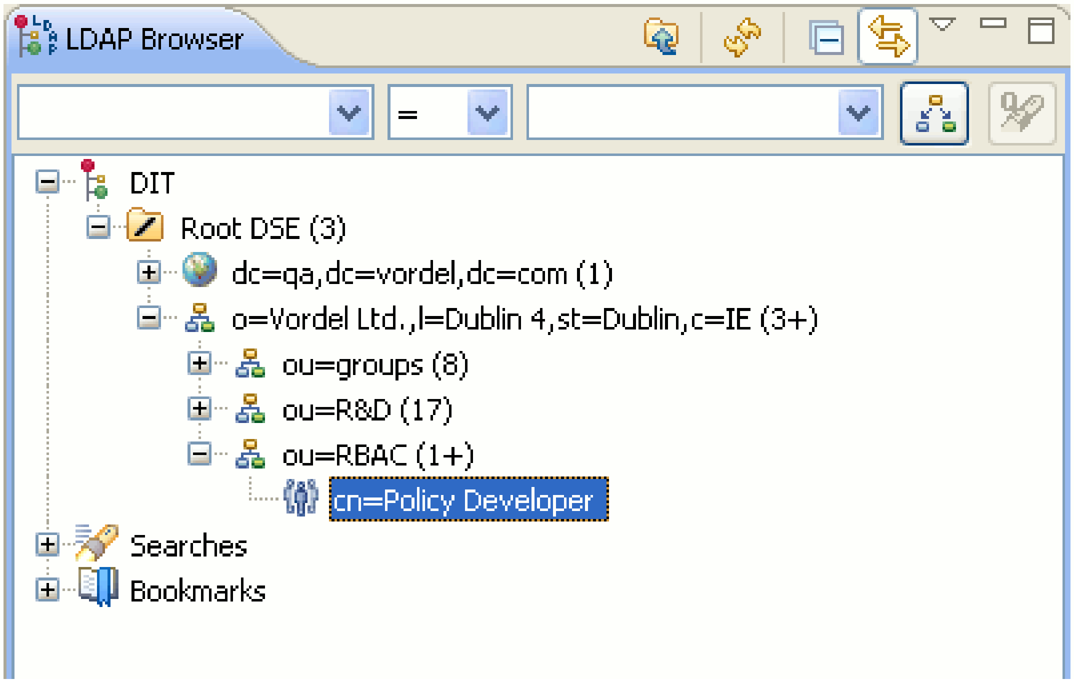

New Entry
Distinguished Name
Please select the parent of the new entry and enter the RDN.
Browse...
+
−

Parent: ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

RDN: cn = Policy Developer

DN Preview: cn=Policy Developer,ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

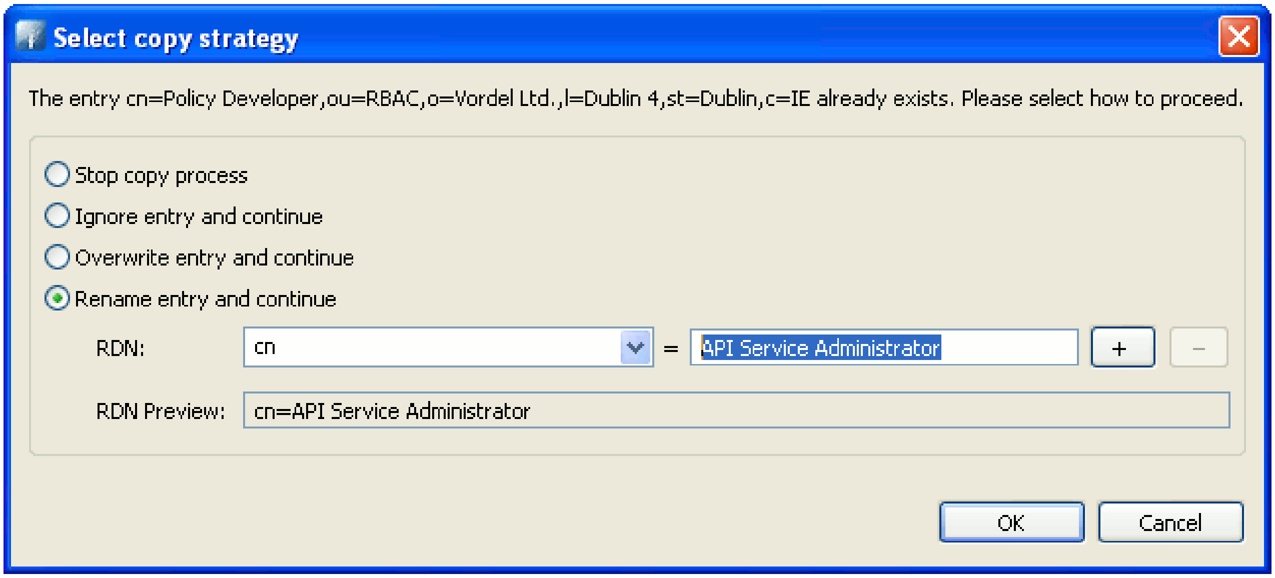
You can view the role in the OpenLDAP group in an LDAP Browser. For example:



- DIT
  - Root DSE (3)
    - dc=qa,dc=vordel,dc=com (1)
    - o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE (3+)
      - ou=groups (8)
      - ou=R&D (17)
      - ou=RBAC (1+)
        - cn=Policy Developer
- Searches
- Bookmarks

Adding Other Roles to the RBAC Directory

You can repeat these steps to add other roles to the RBAC directory. Alternatively, you can copy the `Policy Developer` entry, and paste it into the RBAC directory, renaming the entry with required RBAC role name. For example:



Select copy strategy

The entry cn=Policy Developer,ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE already exists. Please select how to proceed.

○ Stop copy process
○ Ignore entry and continue
○ Overwrite entry and continue
● Rename entry and continue

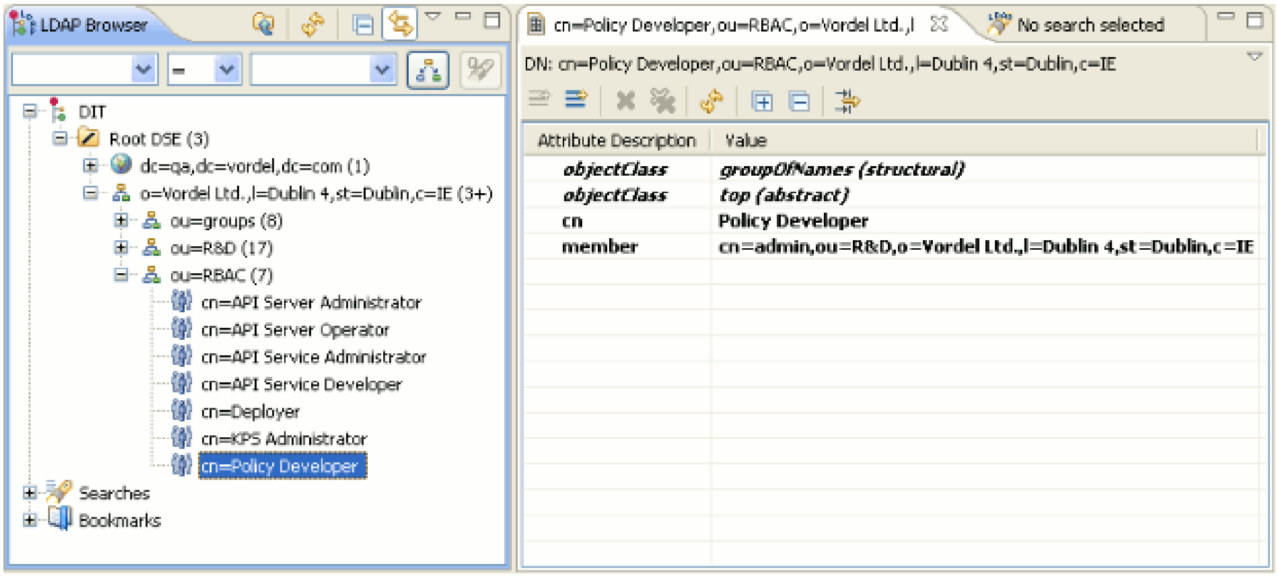
RDN: cn = API Service Administrator

RDN Preview: cn=API Service Administrator

Note

You should have the RBAC directory ready to add members to the role entries. By default, the `admin` user (“`cn=admin,ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE`”) is already a member of the role entries.

The following example shows the added roles:



- DIT
  - Root DSE (3)
    - dc=qa,dc=vordel,dc=com (1)
    - o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE (3+)
      - ou=groups (6)
      - ou=R&D (17)
      - ou=RBAC (7)
        - cn=API Server Administrator
        - cn=API Server Operator
        - cn=API Service Administrator
        - cn=API Service Developer
        - cn=Deployer
        - cn=KPS Administrator
        - cn=Policy Developer
- Searches
- Bookmarks

cn=Policy Developer,ou=RBAC,o=Vordel Ltd.,l

DN: cn=Policy Developer,ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

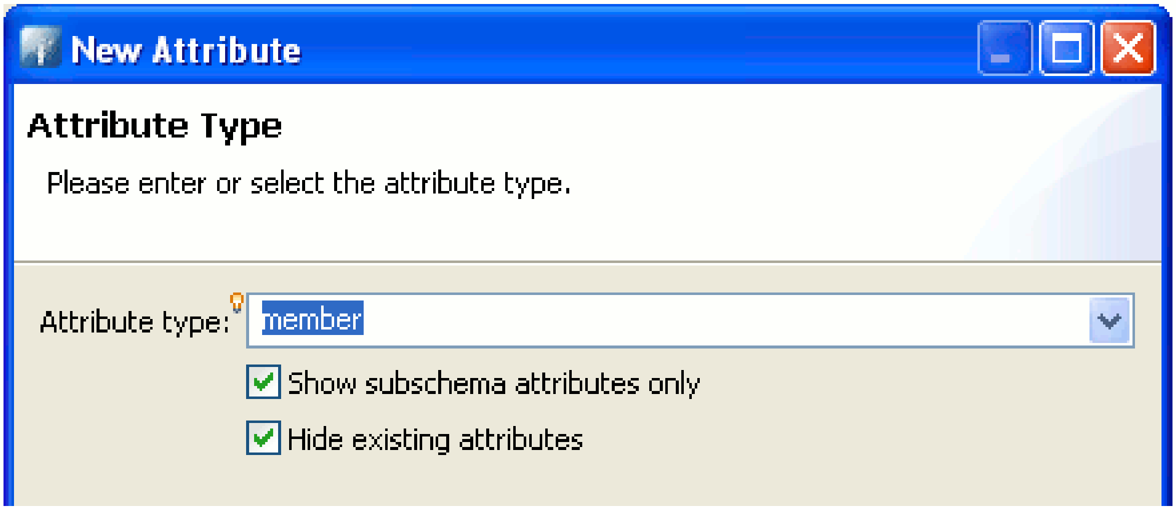
Attribute Description	Value
objectClass	*groupOfNames (structural)*
objectClass	*top (abstract)*
cn	Policy Developer
member	cn=admin,ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

Now you can add new users to the RBAC role entries. The member attribute value should contain the user Distinguished Name. This is explained in the next section.

Step 3: Add Users to the OpenLDAP RBAC Group

To add a user to the OpenLDAP RBAC group, perform the following steps:

1. Select the required RBAC group (for example, `cn=API Server Administrator`) to view the group details.
2. Right-click the list of group attributes, and select **New Attribute**.
3. Enter `member` in the attribute type.



New Attribute

Attribute Type

Please enter or select the attribute type.

Attribute type: member

☑ Show subschema attributes only

☑ Hide existing attributes

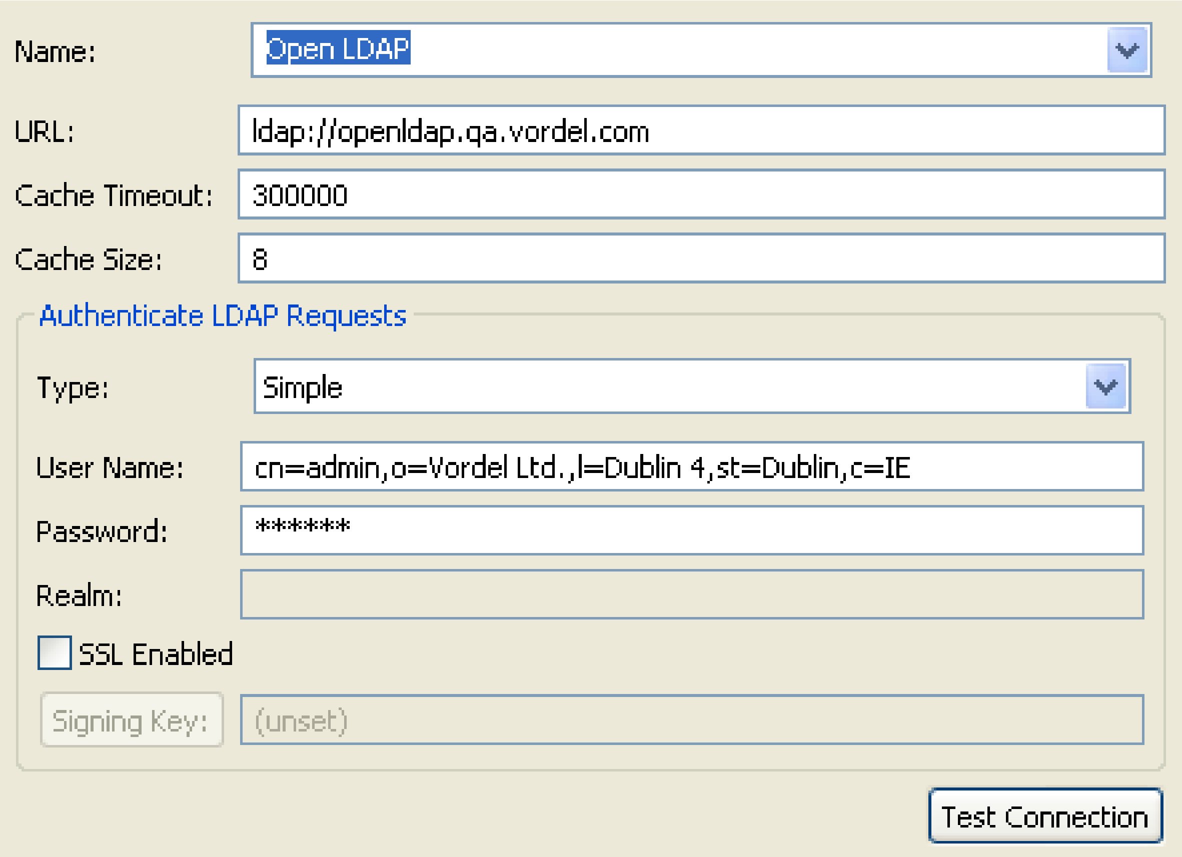
4. Click **Finish**.
5. In the **DN Editor** dialog, enter the user Distinguished Name (for example, `cn=joe.bloggs,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE`).
6. Click **OK**.

The `cn=joe.bloggs,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE` new user has been added to the RBAC `API Server Administrator` role.

Step 4: Create an LDAP Connection

To create an new LDAP Connection, perform the following steps:

1. In the Policy Studio, select **Open File** , and select the Admin Node Manager configuration file (for example, `INSTALL_DIR\apigateway\conf\fed\configs.xml`).
2. In the Policy Studio tree, select **External Connections** -> **LDAP Connections**.
3. Right-click, and select **Create an LDAP Connection**.
4. Complete the fields in the dialog as appropriate. For example:



Name: Open LDAP

URL: ldap://openldap.qa.vordel.com

Cache Timeout: 300000

Cache Size: 8

Authenticate LDAP Requests

Type: Simple

User Name: cn=admin,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE

Password: ******

Realm:

□ SSL Enabled

Signing Key: (unset)

Note

The specified **User Name** should be an LDAP administrator that has access to search the full directory for users.

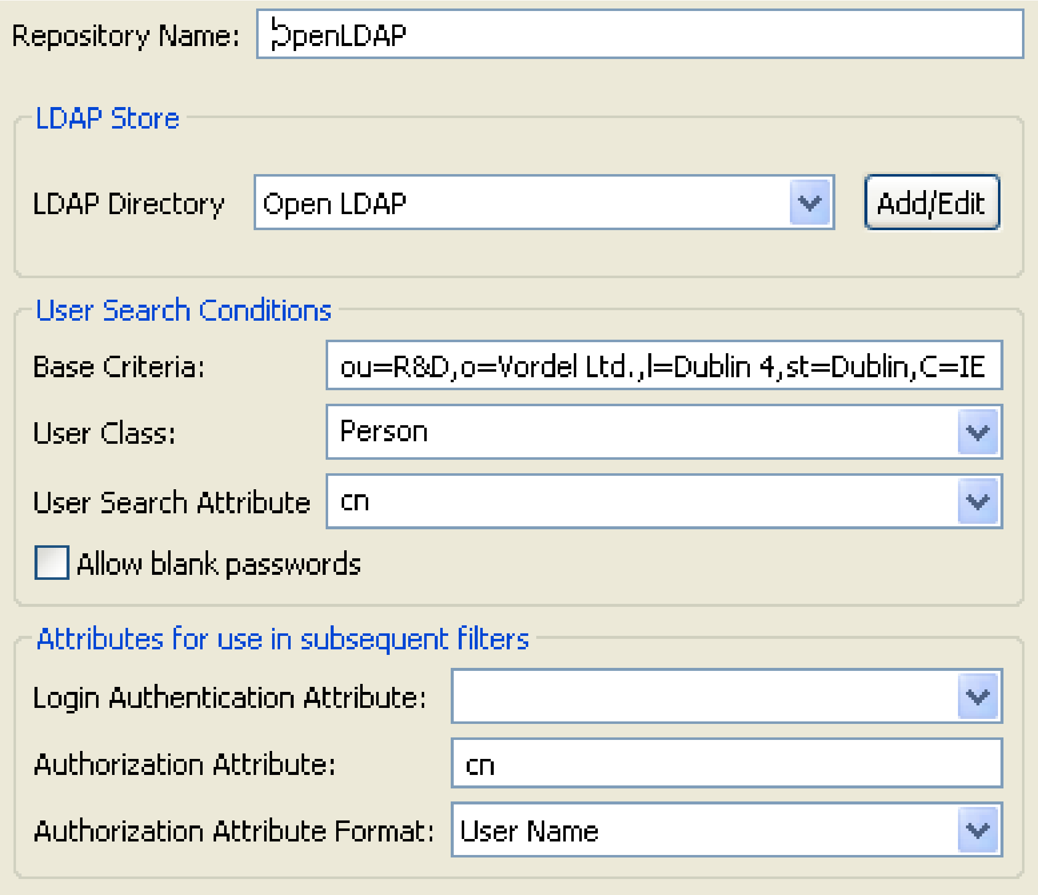
5. Click **Test Connection** to ensure the connection details are correct.

Step 5: Create an OpenLDAP Repository

To create an new OpenLDAP Repository, perform the following steps:

1. In the Policy Studio tree, select **External Connections** -> **Authentication Repository Profiles** -> **LDAP Repositories**.
2. Right-click, and select **Add a new Repository**.
3. Complete the following fields in the dialog:

Repository Name	Enter an appropriate name for the repository.
LDAP Directory	Use the LDAP directory created in the section called “Step 4: Create an LDAP Connection”.
Base Criteria	Enter the LDAP node that contains the users (for example, see the LDAP Browser screen in the section called “Step 3: Add Users to the OpenLDAP RBAC Group”).
User Search Attribute	Enter `cn`. This is the username entered at login time (in this case, `admin`).
Authorization Attribute	Enter `cn`. The `authentication.subject.id` message attribute is set to the value of this LDAP attribute (for example, `cn=admin,ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,c=IE`. The `authentication.subject.id` is used as the base criteria in the filter used to load the LDAP groups (the user’s roles). This allows you to narrow the search to a particular user node in the LDAP tree. For more details, see the **Retrieve Attributes from Directory Server** filter in the section called “Step 6: Create a Test Policy for LDAP Authentication and RBAC”.



Repository Name: OpenLDAP

LDAP Directory: Open LDAP

Base Criteria: ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE

User Class: Person

User Search Attribute: cn

□ Allow blank passwords

Login Authentication Attribute:

Authorization Attribute: cn

Authorization Attribute Format: User Name

Connecting to Other LDAP Repositories
This topic uses OpenLDAP as an example LDAP repository. Other LDAP repositories such as Oracle Directory Server (formerly iPlanet and Sun Directory Server) and Microsoft Active Directory are also supported. For details on using a Microsoft Active Directory repository, see *Using Active Directory for Authentication and RBAC of Management Services*. For an example of querying an Oracle Directory Server repository, see the **Retrieve Attributes from Directory Server** filter in the section called “Step 5: Create a Test Policy for LDAP Authentication and RBAC”.

Step 6: Create a Test Policy for LDAP Authentication and RBAC

To avoid locking yourself out of the Policy Studio, you can create a test policy for LDAP authentication and RBAC, which is invoked when a test URI is called on the server (and not a Management Services URI). For an example policy, select **Policies** -> **Management Services** -> **Sample LDAP Policies** -> **Protect Management Interfaces (LDAP)** when the Admin Node Manager configuration is loaded.

Create the Test Policy
Perform the following steps:

1. Select **Open File** and load the Admin Node Manager configuration file in the Policy Studio. For example:

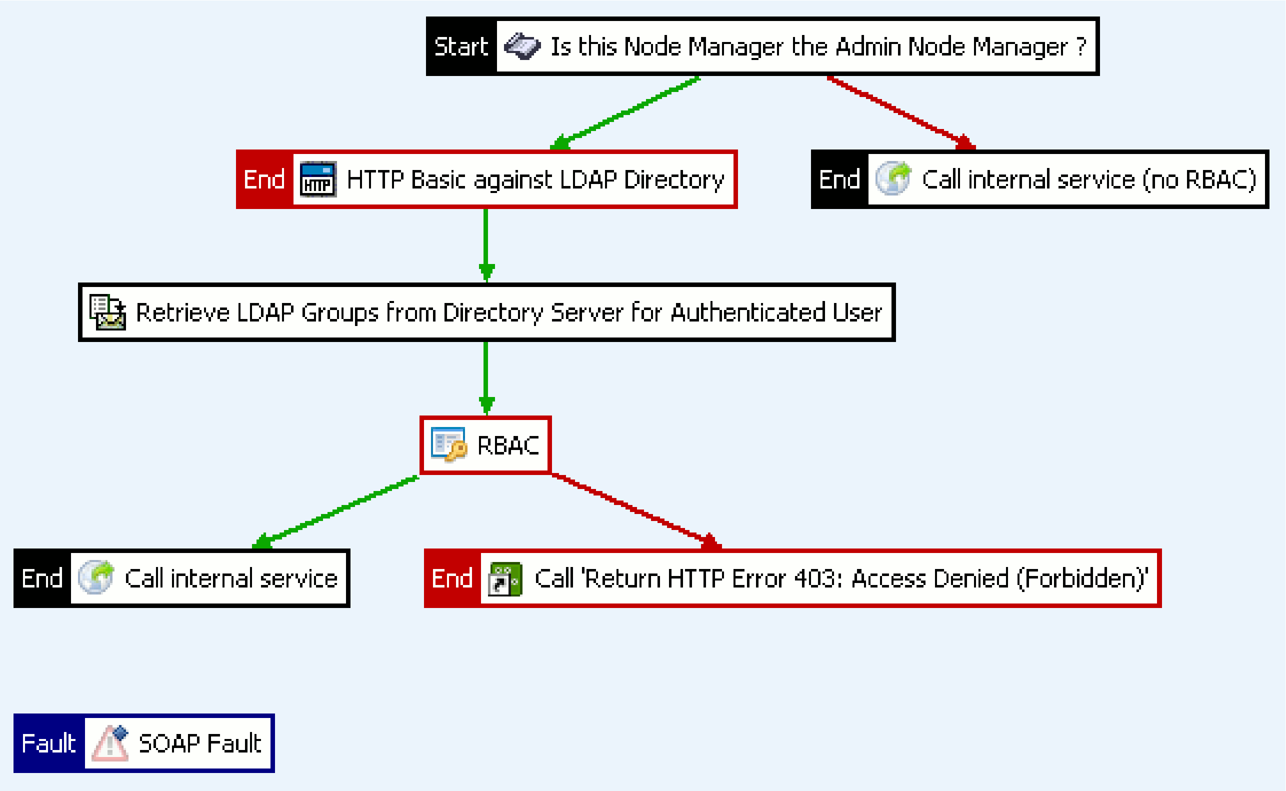
```
INSTALL_DIR/apigateway/conf/fed/configs.xml
```

2. Right-click the **Policies** node in the tree view of the Policy Studio, and select **Add Policy** .

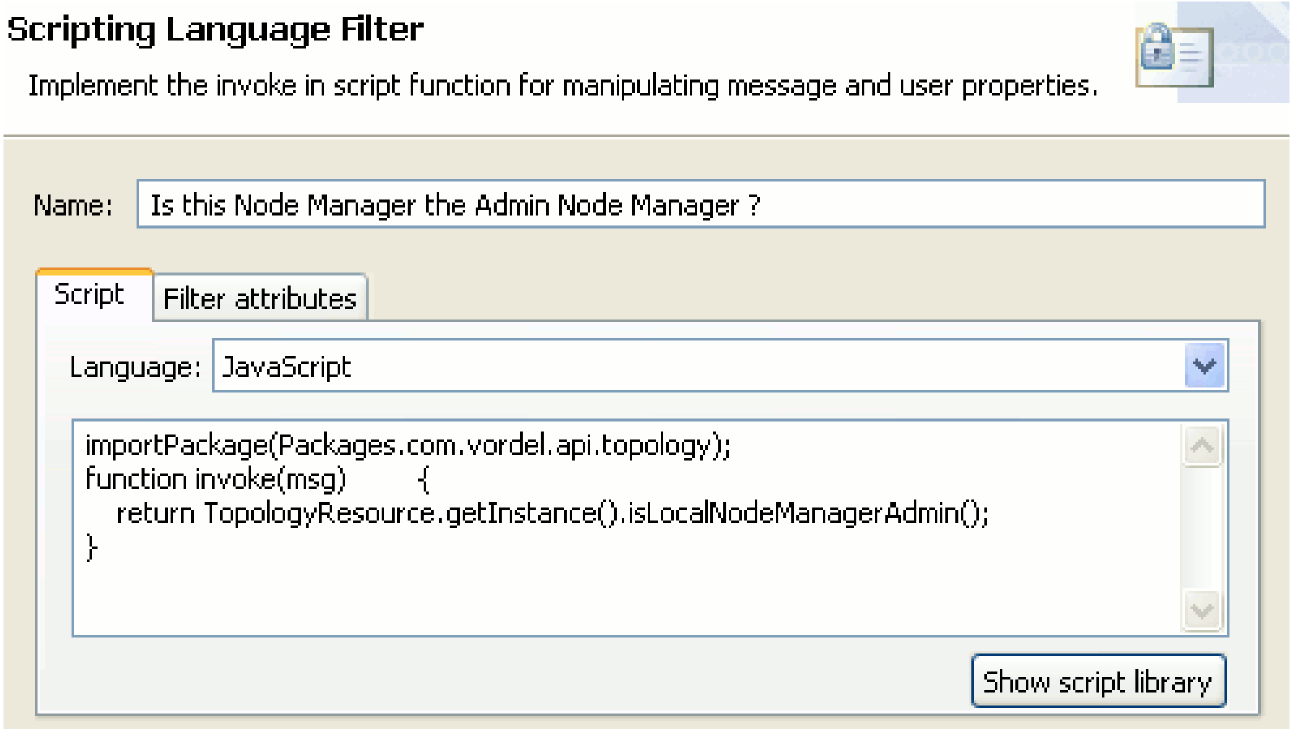
3. Enter a suitable name (for example `Test Policy`) for the new policy in the **Name** field, and click the **OK** button. The new policy is now visible in the tree view.
4. Click the new policy in the tree view to start configuring the filters for the policy. You can easily configure the policy by dragging the required filters from the filter palette on the right of the Policy Studio, and dropping them on to the policy canvas.

For more details, see the topic on *Configuring Policies Manually*.

Configure the Test Policy
Configure the test policy with the following filters:


Start
Is this Node Manager the Admin Node Manager ?
End
HTTP Basic against LDAP Directory
End
Call internal service (no RBAC)
Retrieve LDAP Groups from Directory Server for Authenticated User
RBAC
End
Call internal service
End
Call 'Return HTTP Error 403: Access Denied (Forbidden)'
Fault
SOAP Fault

Scripting Language Filter
This includes the following settings:



Scripting Language Filter
Implement the invoke in script function for manipulating message and user properties.
Script
Filter attributes
Show script library

Name: Is this Node Manager the Admin Node Manager ?

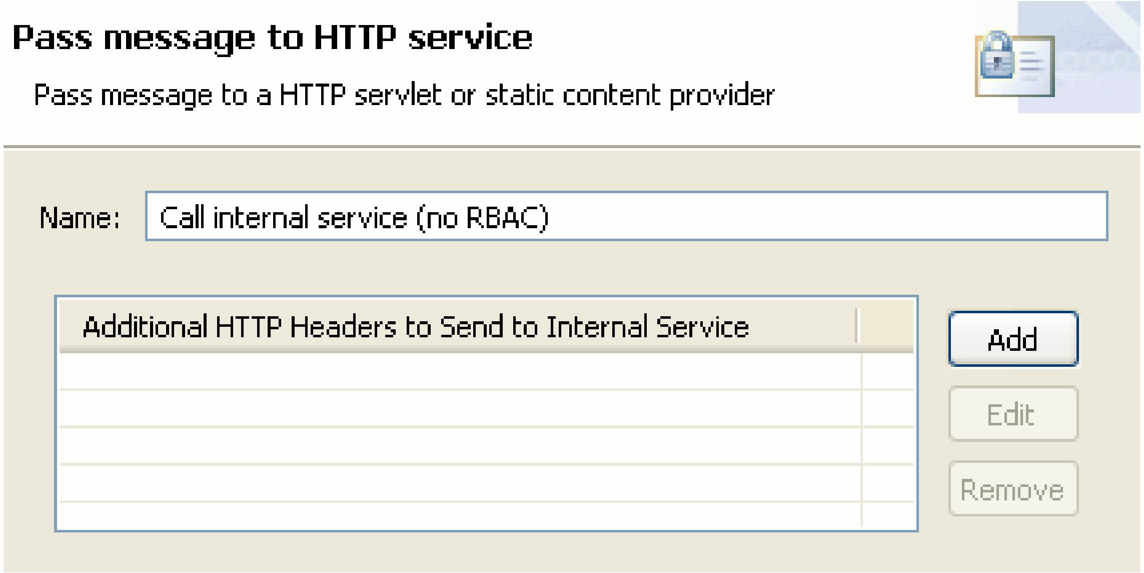
Language: JavaScript

```
importPackage(Packages.com.vordel.api.topology);
function invoke(msg)        {
   return TopologyResource.getInstance().isLocalNodeManagerAdmin();
}
```

The **Scripting Language Filter** performs the following tasks:

- Returns true if the Node Manager is the Admin Node Manager and passes control to the **HTTP Basic Authentication** filter.
- Otherwise, calls the **Call Internal Service (no RBAC)** filter without adding the `authentication.subject.role` and `authentication.subject.id` HTTP headers.

Call Internal Service (no RBAC) Filter
This filter is called without adding any HTTP headers as follows:



Pass message to HTTP service

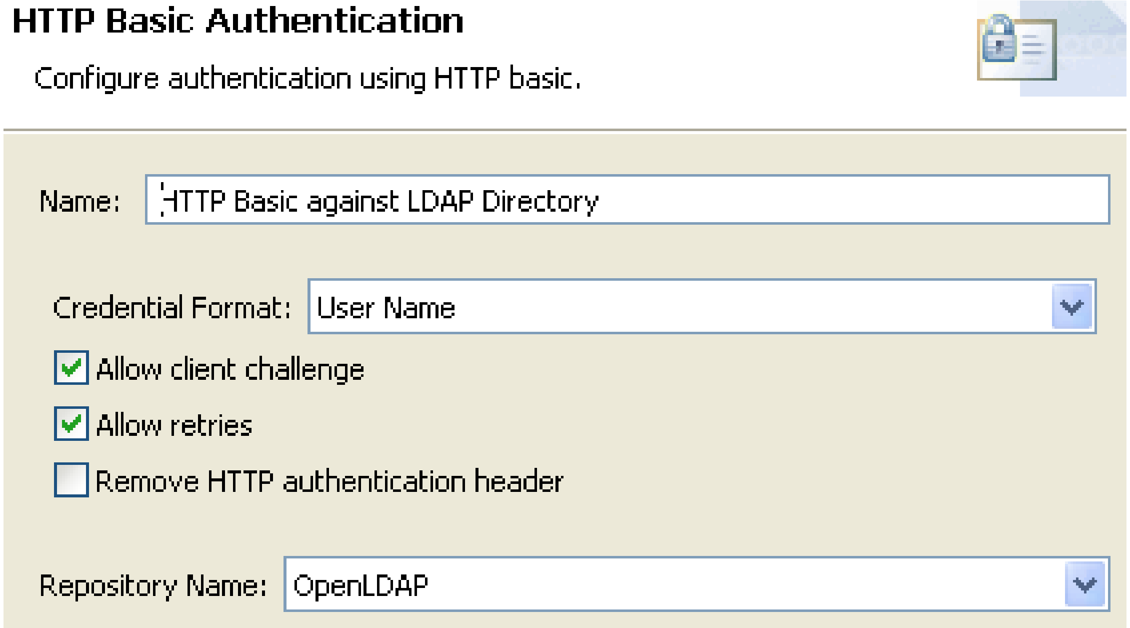
Pass message to a HTTP servlet or static content provider

Name: Call internal service (no RBAC)

Additional HTTP Headers to Send to Internal Service

HTTP Basic Authentication Filter

This filter uses the LDAP repository configured in the section called “Step 5: Create an OpenLDAP Repository”, and includes the following settings:



HTTP Basic Authentication

Configure authentication using HTTP basic.

Name: HTTP Basic against LDAP Directory

Credential Format: User Name

☑ Allow client challenge

☑ Allow retries

☐ Remove HTTP authentication header

Repository Name: OpenLDAP

The **HTTP Basic Authentication** filter performs the following tasks:

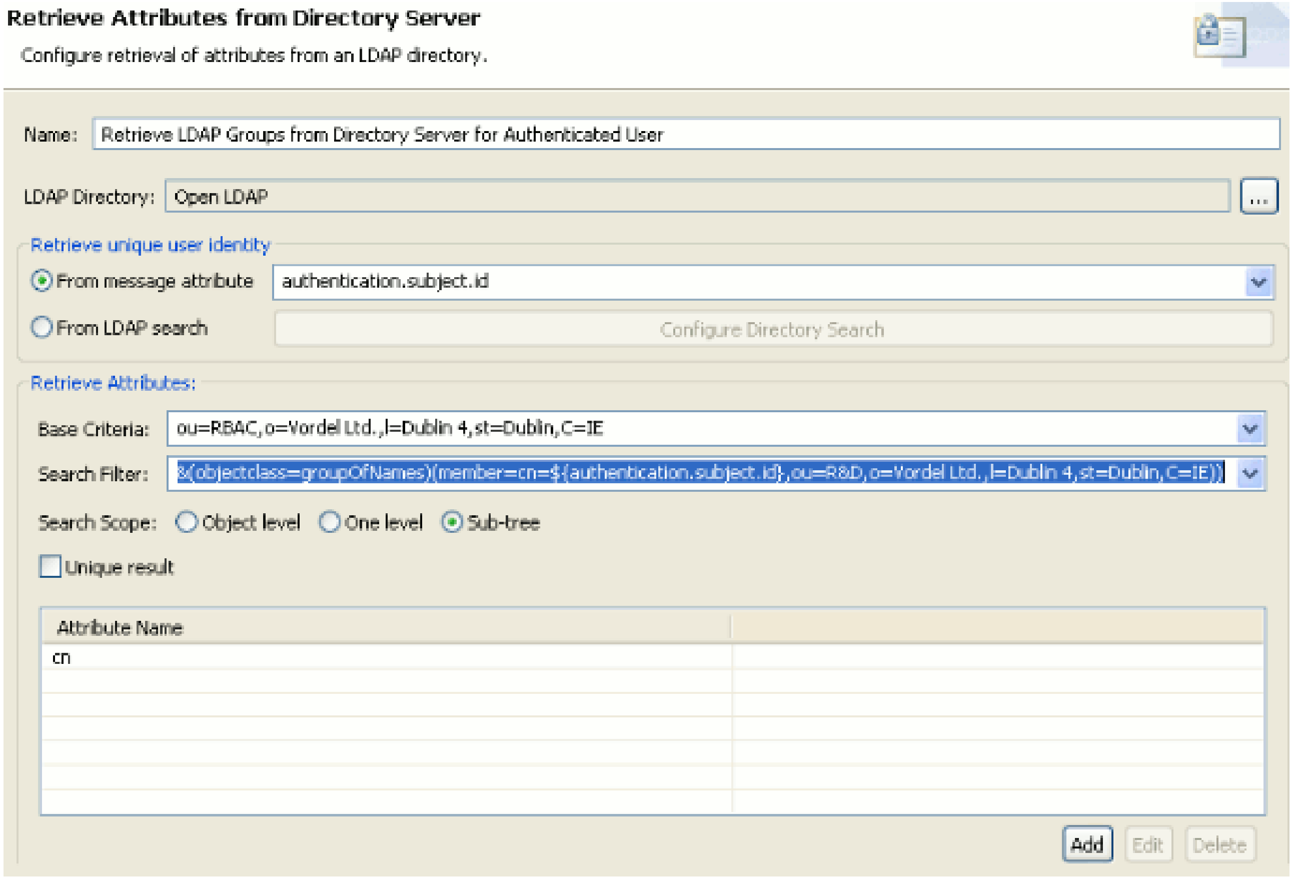
- Connects to the LDAP directory using the connection details specified in the LDAP directory.
- Finds the user using the specified base criteria and search filter.
- If the user is found, verifies the user's name and password against the LDAP repository by performing a bind.
- If authentication fails, always throws a 401. This allows retry for browser users.
- The `distinguishedName` (DName) is held in the `authentication.subject.id` message attribute. This is

specified by the **Authorization Attribute** field in the LDAP repository configuration.

- The user's roles (LDAP groups) are not available yet.

Retrieve Attributes from Directory Server Filter

This filter uses the LDAP directory configured in the section called "Step 4: Create an LDAP Connection", and includes the following settings:



Retrieve Attributes from Directory Server

Configure retrieval of attributes from an LDAP directory.

Name: Retrieve LDAP Groups from Directory Server for Authenticated User

LDAP Directory: Open LDAP

Retrieve unique user identity

● From message attribute: authentication.subject.id

○ From LDAP search

Retrieve Attributes:

Base Criteria: ou=RBAC,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE

Search Filter: &(objectclass=groupOfNames)(member=cn=${authentication.subject.id},ou=R&D,o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE))

Search Scope: ○ Object level ○ One level ● Sub-tree

□ Unique result

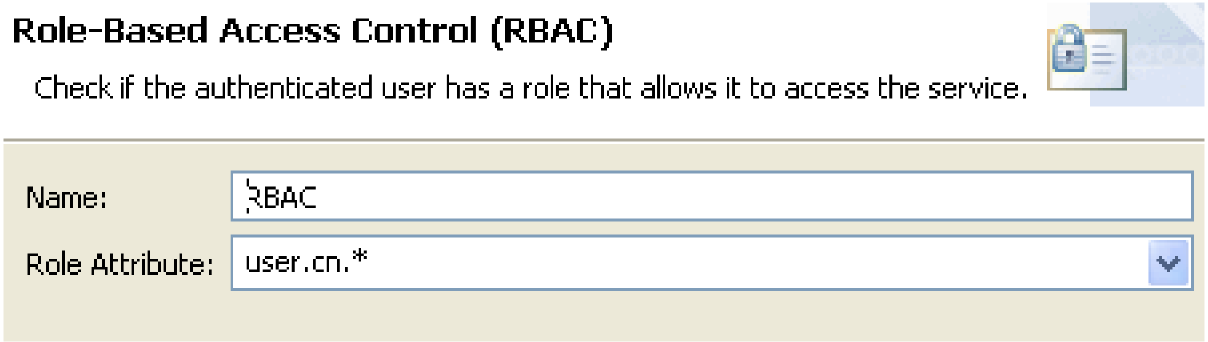
Attribute Name	
cn	

The **Retrieve Attributes from Directory Server** filter performs the following tasks:

- Using the user's DName as the search start point, finds the user's `cn` attribute, and loads the LDAP groups for the user.
- If the user is in one group, the group name is contained in the `user.cn` message attribute. If the user is in multiple (n) LDAP groups, the group names are contained in `user.cn.1`...`user.cn.n` message attributes.

RBAC Filter

This filter includes a the following setting:



Role-Based Access Control (RBAC)

Check if the authenticated user has a role that allows it to access the service.

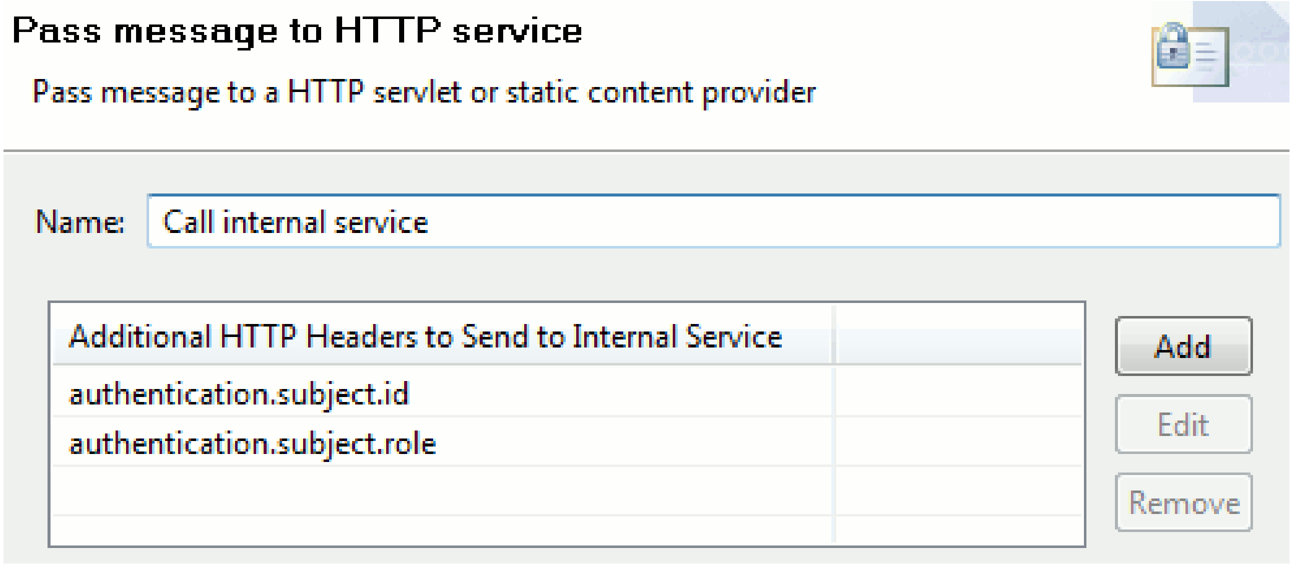
Name: RBAC

Role Attribute: user.cn.*

The **RBAC** filter performs the following tasks:

1. Reads the roles from the `user.cn.*` message attribute. It understands the meaning of the wildcard, and loads the roles as required. It creates a string version of the roles, and places it in the `authentication.subject.role` message attribute for consumption by the **Call Internal Service** filter, which receives the roles as an HTTP header value.
2. Determines which Management Service URI is currently being invoked.
3. Returns true if one of the roles has access to the Management Service currently being invoked, as defined in the `acl.json` file.
4. Otherwise, returns false and calls the **Return HTTP Error 403: Access Denied (Forbidden)** policy. The message content of this filter is shown when a valid user has logged into the browser, but their role(s) does not give them access to the URI they have invoked. For example, this occurs if a new user is created and they have not yet been assigned any roles.

Call Internal Service Filter
This filter includes a the following settings:



Pass message to HTTP service

Pass message to a HTTP servlet or static content provider

Name: Call internal service

Additional HTTP Headers to Send to Internal Service	
authentication.subject.id	
authentication.subject.role	

The **Call Internal Service** filter sends the message to the internal service with the following additional HTTP headers:

- `authentication.subject.role`
- `authentication.subject.role`

Note

The `authentication.subject.id` message attribute is specified using `${authentication.subject.orig.id}` because `authentication.subject.id` holds the full DName, and the `cn` only needs to be passed to the services.

Test the Policy Configuration
To test this policy configuration, perform the following steps:

1. In the Policy Studio tree, select **Listeners** -> **Node Manager** -> **Add HTTP Services**, and enter a service name (for example, `LDAP Test`).
2. Right-click the HTTP service, and select **Add Interface** -> **HTTP**.
3. Enter an available port to test the created policy (for example, `8888`), and click **OK**.
4. Right-click the HTTP service, and select **Add Relative Path**.
5. Enter a relative path (for example, `/test`).
6. Set the **Path Specify Policy** to the **Protect Management Interfaces (LDAP)** policy, and click **OK**.
7. Close the connection to the Admin Node Manager file and reboot the Admin Node Manager so it loads the updated configuration.
8. Use API Gateway Explorer to call `http://localhost:8080/test`.
9. Enter the HTTP Basic credentials (for example, username `admin` and password `Oracle123`). If authentication is passed, the Admin Node Manager should return an HTTP 404 code (not found).

Step 7: Use the OpenLDAP Policy to Protect Management Services

If the authentication and RBAC filters pass, you can now use this policy to protect the management interfaces. To ensure that you do not lock yourself out of the server, perform the following steps:

1. Make a copy of the `conf/fed` directory contents from the server installation, and put it into a directory accessible from the Policy Studio.
2. Make another backup copy of the `conf/fed` directory, which will remain unmodified.
3. In the Policy Studio, select **File** -> **Open**, and browse to `configs.xml` in the first copy of the `fed` directory.
4. Under the **Listeners** -> **Management Services** node, select the `/` and the `/configuration/deployments` relative paths, and set the **Path Specify Policy** to the **Protect Management Interfaces (LDAP)** policy.
5. Remove the previously created `LDAP Test` HTTP Services.
6. Close the connection to the file.
7. Copy the `fed` directory back to the Admin Node Manager's `conf` directory.
8. Reboot the Admin Node Manager.
9. Start the Policy Studio, and connect to the Admin Node Manager using `admin` with its LDAP password (for example, `Oracle123`). You should now be able to edit API Gateway configurations as usual.

Startup Instructions

Overview

This topic describes how to start the Node Manager and API Gateway on all platforms in *console mode* . It also describes how to start the Policy Studio. For details on launching API Service Manager in your browser, see *Introduction to API Service Manager*. For details on API Gateway components and concepts, see the *Oracle API Gateway Architecture*.

Setting Passphrases

By default, data is stored unencrypted in the API Gateway configuration store. However, you can encrypt certain sensitive information, such as passwords and private keys using a passphrase. When the passphrase has been set, this encrypts the API Gateway configuration data. You must enter the passphrase when connecting to the API Gateway configuration data (for example, using the Policy Studio, or when the API Gateway starts up). For more details on configuring this passphrase, see *Setting the Encryption Passphrase*.

Starting the Node Manager

The following instructions describe how to start the Node Manager on the command line in *console* mode on Windows and UNIX systems:

Windows
Complete the following steps to start the Node Manager on a Windows system:

1. Open a DOS prompt at the `/Win32/bin` directory of your API Gateway installation.
2. Run the `nodemanager.bat` file.
3. If you are using an encryption passphrase, you are prompted for this passphrase. Enter the correct encryption passphrase, and press Return. For more details, see *Setting the Encryption Passphrase*.

Linux/Solaris
To start the API Gateway and the Policy Studio on Linux/Solaris systems, complete the following instructions:

1. Open a shell at the `/posix/bin` directory of your API Gateway installation.
2. Run the `nodemanager.sh` file, for example:

```
prompt# ./nodemanager
```

3. If you are using an encryption passphrase, you are prompted for this passphrase. Enter the correct encryption passphrase and press Return. For more details, see *Setting the Encryption Passphrase*.

Starting the API Gateway

The following instructions describe how to start the API Gateway on the command line in *console* mode on Windows and UNIX systems:

Windows
Complete the following steps to start the API Gateway on a Windows system:

1. Open a DOS prompt at the `/Win32/bin` directory of your API Gateway installation.
2. Use the `startinstance` command to start the API Gateway, for example:

```
startinstance -n "my_server" -g "my_group"
```

3. If you are using an encryption passphrase, you are prompted for this passphrase. Enter the correct encryption passphrase, and press Return. For more details, see Setting Passphrases.
4. When the API Gateway has successfully started up, you can run the `policystudio.exe` file from your Policy Studio installation directory.
5. When the Policy Studio is starting up, you are prompted for connection details for the API Gateway. For more details, see Connecting to the API Gateway.

Linux/Solaris
To start the API Gateway and the Policy Studio on Linux/Solaris systems, complete the following instructions:

1. Open a shell at the `/posix/bin` directory of your API Gateway installation.
2. Use the `startinstance` command to start the API Gateway, for example:

```
startinstance -n "my_server" -g "my_group"
```

Note

You must ensure that the `startinstance` file has execute permissions.

3. If you are using an encryption passphrase, you are prompted for this passphrase. Enter the correct encryption passphrase and press Return. For more details, see Setting Passphrases.
4. When the API Gateway has successfully started up, run the `policystudio.sh` file in your Policy Studio installation directory. For example:

```
prompt# cd /usr/home/policystudio
prompt# ./policystudio
```

5. When the Policy Studio is starting up, you are prompted for connection details for the API Gateway.

Tip

You can enter the `startinstance` command without any arguments to display the servers registered on the machine. For example:

```
INSTALL_DIR\Win32\bin>startinstance

usage: "startinstance [[-n instance-name -g group-name [instance-args]] |
[directory-location [instance-args]]]"

The API Gateway instances listed below are available to run on this machine
as follows:
   startinstance -n "server1" -g "group1"
   startinstance -n "server2" -g "group2"
```

Connecting to the API Gateway

When starting up the Policy Studio, you are prompted for details on how to connect to the API Gateway. These include details such as the server session, host, port, user name, and password. The default connection URL is:

```
https://HOST:8090/api
```

where `HOST` points to the IP address or hostname of the machine on which the API Gateway is running. For more information on configuring these settings, see *Connection Details*.

Connection Details

Overview

You can use the Policy Studio to manage API Gateway, Admin Node Manager, and API Gateway Analytics servers. The **Open Connection** dialog enables you to connect to a server URL, and the **Open File** dialog enables you to connect to a server configuration file (`.xml`) or deployment archive (`.fed`) . By default, the Policy Studio connects to a server URL. This topic describes how to connect using a server URL, configuration file, or deployment archive.

Connecting to a URL

The server exposes a deployment service to its underlying configuration data. This enables Policy Studios running on different machines to that on which the server is installed to manage policies remotely. To connect to the deployment service of a running server, select **File** -> **Connect to server** from the main menu, or the equivalent button in the toolbar. Configure the following fields on the **Open Connection** dialog:

Saved Sessions:
Select the session that you wish to use from the drop-down list. You can edit a session name by entering a new name and clicking **Save**. You can also add or remove saved sessions using the appropriate button.

Connection Details
The **Connection Details** section enables you to specify the following settings:

Host:
Specify the host to connect to in this field. The default is `localhost`.

Port:
Specify the port to connect on in this field. The default Admin Node Manager port is `8090`.

Use SSL:
Specify whether to connect securely over SSL. This is selected by default.

User Name:
The deployment service is protected by HTTP Basic authentication. You must provide a user name and password so that the Policy Studio can authenticate to the server. By default, the server User Store contains an `admin` user with a `changeme` password, which can be used in this case. You can change this user's details using the *Authentication Repository* interface.

Password:
Specify the password for the user. The password for the default `admin` user is `changeme`.

Advanced
Click **Advanced** to specify the following setting:

URL:
Enter the URL of the deployment service exposed by the server. For example, the default Admin Node Manager URL is `https://localhost:8090/api`, where `HOST` points to the IP address or host name of the machine on which the API Gateway is running. You can also connect to the API Gateway Analytics server URL. The default server URL addresses are as follows:

Component	***Address***
Admin Node Manager	`https://localhost:8090/api`
API Gateway Analytics	`http://localhost:8040/configuration/deployments/DeploymentService`

Important

To manage API Gateways in your network, you must connect to the Admin Node Manager server URL.

Connecting to a File

Because the server configuration data is stored in XML files, you can specify that the Policy Studio connects directly to a file. You can connect to a server configuration file (`.xml`) or a deployment archive (`.fed`). For more details, see *Deploying Configuration*.

To connect to a file, select **File** -> **Open file** from the main menu, or click the **Open file** link on the welcome page. Complete the following fields on the **Open File** dialog:

File:
Enter or browse to the location of a server configuration file (for example, `INSTALL_DIR\groups\group-2\conf\378fd412-4e14-4924-b666-b974adf19642\configs.xml`). Alternatively, enter or browse to the location of a deployment archive (`.fed`).

Passphrase Key:
All sensitive server configuration data (password, keys, and so on) can be encrypted using a passphrase. If you wish to do this, enter a password in this field when connecting. You must use this password thereafter when connecting to the server.

Unlocking a Server Connection

You can also use the **Open File** dialog to unlock a connection to a server. This is for emergency use when you have changed configuration that results in you being locked out from the **Management Services** on port `8090`. In this case, you have misconfigured the authentication filter in the **Protect Management Interfaces** policy. For example, if you created and deployed an LDAP connection without specifying the correct associated user accounts, and are now unable to connect to the Admin Node Manager.

To unlock a server connection, perform the following steps:

1. Download all the files in the server's `conf/fed` directory to the machine on which the Policy Studio is installed.
2. Start the Policy Studio.
3. Connect to the `configs.xml` file that you downloaded from the server in step 1 (for details, see the section called “Connecting to a File”).
4. Change the configuration details as required (for example, specify the correct user account details for the LDAP connection under the **External Connections** node).
5. Upload the files back to the server's `conf/fed` directory.
6. Connect to the server URL in the Policy Studio.

For more details on Management Services, see *Policy Studio Preferences*.

Global Configuration

Overview

For convenience, the Policy Studio displays various global configuration options. For example, it includes libraries of users, X.509 certificates, and schemas that can be added globally and then referenced in filters and policies. This avoids the need to reconfigure details over and over again (for example, each time a schema or certificate is used).

The following global configuration options are available in the Policy Studio, each of which are discussed briefly in the sections below:

- Server Configuration
- API Gateway Settings
- Web Services Repository
- Processes
- Policies
- Certificates and Keys
- API Gateway Users
- Alerts
- External Connections
- Caches
- Black list
- White list
- Schema Cache
- Scripts
- Stylesheets

Server Configuration

You can manage the server configuration for the API Gateway using the **Server** menu option in the Policy Studio main menu. This includes deploying the server configuration.

For more details, see the *Server Configuration* topic.

API Gateway Settings

You can configure the underlying configuration settings for the API Gateway using the **Settings** node in the Policy Studio tree. This includes the following tabs:

- Default Settings
- Audit Log
- Namespace
- MIME/DIME
- Traffic Monitor
- Metrics
- Control Log
- Cache
- Access Log
- Security Service Module
- Kerberos

- Tivoli

For more details, see the *API Gateway Settings* topic.

Web Services Repository

The easiest way to secure a Web Service with the API Gateway is to import the WSDL (Web Services Description Language) file for the service using the Policy Studio. This creates a Service Handler for the Web Service, which is used to control and validate requests to the Web Service and responses from the Web Service.

The WSDL file is also added to the Web Services Repository, making sure to update the URL of the Web Service to point at the machine on which the API Gateway is running instead of that on which the Web Service is running. Consumers of the Web Service can then query the API Gateway for the WSDL file for the Web Service. The consumer then knows to route messages to the API Gateway instead of attempting to route directly to the Web Service, which most likely will not be available on a public IP address.

The Web Services Repository offers administrators a very simple way of securing a Web Service with minimal impact on consumers of that service. Because of this, the Web Services Repository should be used as the primary method of setting up policies within the Policy Studio. For more information on using the Repository to setup policies, see the *Web Service Repository* tutorial.

Processes

A Process represents a single running instance of the API Gateway. It enables you to configure at least two interfaces: one for public traffic, and a second for listening for and serving configuration data. The configuration interface should rarely need to be updated. However, it is likely that you will want to add several HTTP interfaces. For example, you may want to add an HTTP interface and an SSL-enabled HTTPS interface.

Furthermore, you can also add features such as the following at the Process level:

- Remote hosts to control connection settings to a server.
- SMTP interfaces to configure email relay
- File transfer services for FTP, FTPS, and SFTP
- Policy execution schedulers to run policies at regular time intervals
- JMS listeners to listen for JMS messages
- Packet sniffers to inspect packets at the network level for logging and monitoring
- FTP pollers to retrieve files to be processed by polling a remote file server.
- Directory scanners to scan messages dumped to the file system

Because the API Gateway can read messages from HTTP, SMTP, FTP, JMS, or a directory, this enables it to perform protocol translation. For example, the API Gateway can read a message from a JMS queue, and then route it on over HTTP to a Web Service. Similarly, the API Gateway can read XML messages that have been put into a directory on the file system using FTP, and send them to a JMS messaging system, or route them over HTTP to a back-end system.

For more information on configuring processes, see the *Configuring API Gateway Instances* tutorial.

Policies

A policy is made up of a sequence of modular, reusable message filters, each of which processes the message in a particular way. There are many categories of filters available, including authentication, authorization, content filtering, routing, and many more. For example, a typical policy might contain an authentication filter, followed by several content-based filters (for example, Schema Validation, Threatening Content, Message Size, XML Complexity, and so on), and provided all configured filters run successfully, the message is routed on to the configured destination.

A policy can be thought of as a network of message filters. A message can traverse different paths through the network depending on what filters succeed or fail. This enables you to configure policies that, for example, route messages that pass one Schema Validation filter to one back-end system, and route messages that pass a different Schema Validation filter to a different system.

You can use *Policy Containers* to help manage your policies. These are typically used to group together a number of similar policies (for example, all authentication policies) or to act as an umbrella around several policies that relate to a particular policy (for example, all policies for the `getQuote` Web Service). A number of useful policies that ship with the API Gateway are found in the **Policy Library** Policy Container. This container is pre-populated with policies to return various types of faults to the client and policies to block certain types of threatening content, among others. You can also add your own policies to this container, and create your own Policy Containers as necessary to suit your own requirements.

Certificates and Keys

The API Gateway must be able to trust X.509 certificates to establish SSL connections with external servers, validate XML Signatures, encrypt XML segments for certain recipients, and for other such cryptographic operations. Similarly, a private key is required to carry out certain other cryptographic operations, such as message signing and decrypting data.

The **Certificate Store** contains all the certificates and keys that are considered to be trusted by the API Gateway. Certificates can be imported into or created by the Certificate Store. You can also assign a private key to the public key stored in a certificate, by importing the private key, or by generating one using the provided interface.

For more information on importing and creating certificates and keys, see the *Certificates and Keys* topic.

API Gateway User Store

Users are mainly used for authentication purposes in the API Gateway. In this context, the **User Store** acts as a repository for user information against which users can be authenticated. You can also store user attributes for each user or user group. For example, you can then use these attributes when generating SAML attribute assertions on behalf of the user.

The *API Gateway Users* topic contains more details on how to create users, user groups, and attributes.

System Alerts

The API Gateway can send system alerts to various error reporting systems in the case of a policy error (for example, when a request is blocked by a policy). Alerts can be sent to a Windows Event Log, local syslog, remote syslog, OPSEC firewall, SNMP NMS, Twitter, or email recipient.

For more details on how to configure the API Gateway to send these alerts, see the *System Alerting* topic.

External Connections

The API Gateway can leverage your existing identity management infrastructure and avoid maintaining separate silos of user information. For example, if you already have a database full of user credentials, the API Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, the API Gateway can authorize users, lookup user attributes, and validate certificates against third-party identity management servers.

You can add each connection to an external system as a global **External Connection** in the Policy Studio so that it can be reused across all filters and policies. For example, if you create a policy that authenticates users against an LDAP directory and then validates an XML signature by retrieving a public key from the same LDAP directory, it makes sense to create a global External Connection for that LDAP directory. You can then select the LDAP Connection in both the authentication and XML signature verification filters, rather than having to reconfigure it in both filters.

For example, you can use the External Connections interface to configure global connections such as the following:

- Authentication Repository Profiles

- Database Connections
- ICAP Servers
- JMS Services
- Kerberos Services
- LDAP Connections
- OCSP Connections
- Proxy Servers
- Radius Clients
- SiteMinder Connections
- TIBCO Connections
- Tivoli Connections
- XKMS Connections

You can also use External Connections in cases where you want to configure a group of related URLs. This is most useful in cases where you want to round-robin between a number of related URLs to ensure high availability. When the API Gateway is configured to use a *URL Connection Set* (instead of just a single URL), it round-robins between the URLs in the set.

For more information on configuring External Connections and Connection Sets, see the *External Connections* topic.

Caches

You can configure the API Gateway to cache responses from a back-end Web Service. For example, if the API Gateway receives two successive identical requests it can (if configured) take the response for this request from the cache instead of routing the request on to the Web Service and asking it to generate the response again.

As a result, excess traffic is diverted from the Web Service making it more responsive to requests for other services. The API Gateway is saved the processing effort of routing identical requests unnecessarily to the Web Service, and the client benefits from the far shorter response time.

You can configure local caches for each running instance of the API Gateway. If you have deployed multiple API Gateways throughout your network, you can configure a distributed cache where cache events on one cache are replicated across all others. For example, if a response message is cached at one instance of the API Gateway, it is added to all other caches.

For more details on how to configure the API Gateway to use local and distributed caches, see the *Global Caches* topic.

Black list and White list

The **White list** is a global library of regular expressions that can be used across several different filters. For example, the **Validate HTTP Headers**, **Validate Query String**, and **Validate Message Attributes** filters all use regular expressions from the **White list** to ensure that various parts of the request contain expected content.

The **White list** is pre-populated with regular expressions that can be used to identify common data formats, such as alphanumeric characters, dates, email addresses, IP addresses, and so on. For example, if a particular HTTP header is expected to contain an email address, the **Email Address** expression from the library can be run against the HTTP header to ensure that it contains an email address as expected. This is yet another way that the API Gateway can ensure that only the correct data reaches the Web Service.

While the **White list** contains regular expressions to identify valid data, the **Black list** contains regular expressions that are used to identify common attack signatures. For example, this includes expressions to scan for SQL injection attacks, buffer overflow attacks, ASCII control characters, DTD entity expansion attacks, and many more.

You can run various parts of the request message against the regular expressions contained in the **Black list** library. For example, the HTTP headers, request query string, and message (MIME) parts can be scanned for SQL injection attacks by selecting the SQL-type expressions from the **Black list**. The **Threatening Content** filter also uses regular expressions from the **Black list** to identify attack signatures in request messages.

For more details on running regular expressions, see the following topics:

- *HTTP Header Validation*
- *Query String Validation*
- *Validate Message Attributes*
- *Threatening Content*

Schema Cache

The Schema Cache contains the XML Schemas that the API Gateway can use to validate incoming requests against. The **Schema Validation** filter validates the format of an incoming message against a schema from the cache. This ensures that only messages of the correct format are processed by the target system.

In the Policy Studio navigation tree, you can access the global Schema Library by selecting **Resources** -> **Schemas**. Select a child node to view or edit its contents. To add a schema or schema container, right-click the **Scripts** node, and select the appropriate option. For more details on importing XML Schemas into the cache, and using containers organize schemas, see the *Global Schema Cache* topic.

When you have imported your XML schemas, see the *Schema Validation* tutorial for instructions on how to validate XML messages against the schemas in the cache.

Scripts

The Scripts Library contains the JavaScript and Groovy scripts that the API Gateway can use to interact with the message as it is processed. For example, you use these scripts with the **Scripting Filter** to get, set, and evaluate specific message attributes.

In the Policy Studio navigation tree, you can access the global Scripts Library by selecting **Resources** -> **Scripts**. Select a child node to view or edit its contents. To add a script, right-click the **Scripts** node, and select **Add Script**.

For more details on using the **Scripts Library** dialog to add scripts, and on configuring the API Gateway to use scripts, see the topic on the *Scripting Language Filter*.

Stylesheets

The Stylesheet Library contains the XSLT stylesheets that the API Gateway can use to transform incoming request messages. The **XSLT Transformation** filter enables you convert the contents of a message using these stylesheets. For example, an incoming XML message that adheres to a specific XML schema can be converted to an XML message that adheres to a different schema before it is sent to the destination Web Service.

In the Policy Studio navigation tree, you can access the global Stylesheet Library by selecting **Resources** -> **Stylesheets**. Select a child node to view or edit its contents. To add a stylesheet, right-click the **Stylesheets** node, and select **Add Stylesheet**.

For more details on using the **Stylesheet Library** dialog to add stylesheets, and on configuring the API Gateway to use XSLT stylesheets, see the topic on the *XSLT Transformation* filter.

References

References can occur between API Gateway configurations items (for example, a policy might include a reference to an external connection to a database). You can view references between configuration items in the Policy Studio by right-clicking an item, and selecting **Show All References**. References are displayed in a tab at the bottom of the screen.

The **Show All References** option is enabled only for items that have references to other items. For an example in a default API Gateway installation, right-click **External Connections** -> **LDAP Connections** -> **Sample Active Directory Connection**, and select **Show all References**. Showing all references is useful for impact analysis (for example, before upgrading or migrating), and is a general navigation aid.

Server Configuration

Overview

You can manage the server configuration for the API Gateway using the **Server** menu option in the Policy Studio main menu. You can also use the **Deploy** button in the toolbar. The following menu options are available.

Deploy

When you make changes to a filter or policy using the Policy Studio, you must deploy to the API Gateway for the changes to take affect. You can use the **Server** -> **Deploy** menu option, or the **Deploy** button in the toolbar. Alternatively, you can press F6. If the server is processing a number of messages when the deploy command is issued, all of the messages are processed using the existing policy. New messages are queued until this batch of messages is completely processed. When the new policy data has been stored and loaded by the server, the queued messages are processed using the new policy.

Important

To deploy to the API Gateway, the Policy Studio sends a deployment request to the API Gateway. If necessary, you can configure the socket timeout value for this connection in the Policy Studio **Preferences** dialog, available from **Window -> Preferences**. For more details, see *Policy Studio Preferences*.

API Gateway Settings

Overview

You can configure the underlying settings for the API Gateway using the **Tasks** -> **Manage Settings** menu option in the Policy Studio main menu, or the **Settings** node in the Policy Studio tree. This topic describes the tabs available at the bottom of the **Settings** screen. You can save the settings on each tab by clicking the **Save Settings** icon at the top right of the tab.

Default Settings

The **Default Settings** entered in this screen are applied to all instances of the API Gateway that use this particular configuration. For example, you can change the trace level, timeouts, cache sizes, and other such global information. For more details, see *Default Settings*.

Audit Log

The Audit Log settings enable you to configure the default logging behavior of the API Gateway. For example, you can configure the API Gateway to log to a database, text or XML file, local or remote UNIX syslog, or the system console. For more details, see the topic on *Audit Log Settings*.

Namespace

The Namespace settings are used to determine the versions of SOAP, Web Services Security (WSSE) and Web Services Utility (WSU) that the API Gateway supports. For more details, see *Namespace Settings*.

MIME/DIME

The API Gateway can filter MIME messages based on the content types (or MIME types) of the individual parts of the message. The MIME/DIME settings list the default MIME types that the API Gateway can filter on. These types are then used by the **Content Types** filter to determine which MIME types to block or allow through to the back end Web Service. For more details, see *MIME/DIME Settings*.

Traffic Monitor

The **Traffic Monitor** settings enable you to configure the web-based Traffic Monitor tool and its message traffic log. For example, you can configure where the data is stored and what message transaction details are recorded in the log. For more details, see *Configuring Traffic Monitoring*.

Metrics

The **Metrics** settings enable you to configure statistics about the messages that the API Gateway processes in a database. The API Gateway Analytics monitoring tool can then poll this database, and produce charts and graphs showing how the API Gateway is performing. For more details, see *Real-Time Monitoring Settings*.

Session Settings

The **Session Settings** enable you to configure session management settings for the selected cache. For example, you can configure the period of time before expired sessions are cleared from the default `HTTP Sessions` cache. For more details, see the *Session Settings* topic.

Cache

If you have deployed several API Gateways throughout your network, you should configure a distributed cache. In a distributed cache, each cache is a peer in a group and needs to know where all the other peers in the group are located.

The **Cache Settings** enable you to configure settings for peer listeners and peer discovery. For more details, see the *Global Caches*.

Access Log

The Access Log records a summary of all request and response messages that pass through the API Gateway. For example, this includes details such as the remote hostname, username, date and time, first line of the request message, HTTP status code, and number of bytes. For details on configuring these settings per API Gateway, see the *Access Log Settings* topic. For details on configuring the the Access Log at the service level, see the topic on *Configuring HTTP Services*.

Security Service Module

You can configure the API Gateway to act as an Oracle Security Service Module (SSM) to enable integration with Oracle Entitlements Server 10g. The API Gateway acts as a Java SSM, which delegates to Oracle Entitlements Server 10g. For example, you can authenticate and authorize a user for a particular resource against an Oracle Entitlements Server 10g repository. For more details, see the *Oracle Security Service Module Settings (10g)* topic.

⚠ **Important**

Oracle SSM is required for integration with Oracle OES 10g only. Oracle SSM is not required for integration with Oracle OES 11g.

Kerberos

You can configure Kerberos settings such as the Kerberos configuration file to the API Gateway, which contains information about the location of the Kerberos Key Distribution Center (KDC), encryption algorithms and keys, and domain realms. You can also configure options for APIs used by the Kerberos system, such as the Generic Security Services (GSS) and Simple and Protected GSSAPI Negotiation (SPNEGO) APIs. For more details, see the *Kerberos Configuration* topic.

Tivoli

You can configure how the API Gateway Process connects to an instance of an IBM Tivoli Access Manager server. Each API Gateway process can connect to a single Tivoli server. For more details, see the Global Configuration section in the *Tivoli Integration* topic.

Policy Studio Preferences

Overview

The **Preferences** dialog enables you to configure a range of options for the Policy Studio. For example, you can configure the level at which the Policy Studio traces diagnostic output, customize the look-and-feel of the Policy Studio, or configure the timeout for the Policy Studio connection to the API Gateway. Each of the available settings is discussed in the following sections.

Management Services

The Admin Node Manager and Oracle API Gateway Analytics expose certain interfaces that are used for management purposes only, and should be edited only under strict advice from the Oracle Support team. By default, the **Management Services** policies and interfaces, interfaces and server process are hidden from view in the Policy Studio tree. You can display them by selecting the **Show Management Services** option in this dialog.

When this option is selected, you can view the **Management Services** policy container in the tree under the **Policies** node. The **Management Services** HTTP interfaces are also displayed under the **Listeners** node under the server process. For more details, see the section called “Management Services” in the *Configuring HTTP Services* topic.

Important

⚠ You should only modify **Management Services** under strict advice and supervision from the Oracle Support team.

Policy Colors

The **Policy Colors** settings enable you to customize the look-and-feel of the Policy Canvas in the Policy Studio. For example, you can change the colors of the following components:

- **Policy Background:**
  Changes the background color of the Policy Canvas.
- **Missing Attribute:**
  You can right-click the Policy Canvas, and select **Show All Attributes** from the context menu. When this is selected, each filter displays the list of required and generated message attributes that are relevant for that filter. If a required attribute has not been generated by a previous filter in the policy, the attribute is highlighted in a different color (red by default). You can change this color by selecting an appropriate color using this setting.
- **Success Path:**
  You can change the color of the Success Path link using this setting.
- **Failure Path:**
  Similarly, you can change the color of the Failure Path link here.
- **Show Link Labels:**
  If this option is selected, a Success Path is labeled with the letter S, while a Failure Path is labeled F.

Proxy Settings

You can specify global proxy settings that apply only when downloading WSDL, XSD, and XSLT files from the Policy Studio. These include the following settings:

Proxy Setting	***Description***
Host	Host name or IP address of the proxy server.
Port	Port number on which to connect to the proxy server.

Proxy Setting	***Description***
Username	Optional user name when connecting to the proxy server.
Password	Optional password when connecting to the proxy server.

You can also specify individual proxy servers under the **External Connections** node in the Policy Studio tree. These are different from the global proxy settings in the **Preferences** because you can specify these proxy servers at the filter level (in the **Connection** and **Connect To URL** filters). For more details, see the *Proxy Servers* topic.

Runtime Dependencies

The **Runtime Dependencies** setting enables you to add JAR files to the Policy Studio classpath. For example, if you write a custom message filter, you must add its JAR file, and any third-party JAR files that it uses, to the **Runtime Dependencies** list.

Click **Add** to select a JAR file to add to the list of dependencies, and click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.

Important

You must restart the Policy Studio and the server for these changes to take effect.

Server Connection

When an update is made to a configuration setting in the Policy Studio, the update is stored locally until you deploy it to the API Gateway. After making one or more configuration updates, you can deploy to the API Gateway by selecting **Server** -> **Deploy** in the main menu. Alternatively, you can click the **Deploy** button in the toolbar, or press F6.

The updated configuration is then pushed to the API Gateway, which finishes processing any current messages before flushing its cached policy configurations. The new configuration is then stored and loaded. Any subsequent messages received by the API Gateway then use the new configuration.

When deploying to the API Gateway, the Policy Studio sends a deployment request to the server. If necessary, you can configure the socket timeout value for this connection in the Policy Studio **Preferences** dialog. Enter the timeout value in milliseconds in the **Server Socket Connection Timeout** field.

SSL Settings

The **SSL Settings** enable you to specify what action is taken when an unrecognized server certificate is presented to the client. This allows the Policy Studio to connect to SSL services without a requirement to add a certificate to its JVM certificate store.

Configure one of the following options:

Prompt User	When you try to connect to SSL services, you are prompted with a dialog. If you choose to trust this particular server certificate displayed in the dialog, it is stored locally, and you are not prompted again.
Trust All	All server certificates are trusted.
Keystore	Enter or browse to the location of the **Keystore** that contains the authentication credentials sent to a remote host for mutual SSL, and enter the appropriate **Keystore Pass-**

	word.

Status Bar

The **Show Status Bar** setting enables you to specify whether the applications status bar is displayed at the bottom of the Policy Studio screen. For example, this status bar displays details such as the currently selected tree node on the left, and details such as the heap size on the right. You can also use the status bar to run garbage collection by clicking the trash icon on the right. This status bar is enabled by default.

Trace Level

You can set the level at which the Policy Studio logs diagnostic output by selecting the appropriate level from the **Tracing Level** drop-down list. Diagnostic output is written to a file in the `/logs` directory of your Policy Studio installation. You can also select **Window** -> **Show View** -> **Console** in the main menu to view the trace output in the **Console** window at the bottom of the screen. The default trace level is `INFO`.

Web and XML

The **Web and XML** settings enable you to configure a range of options that affect how XML files are treated in the Policy Studio.

XML Files
This includes the following options:

Creating or saving files	Specifies a line delimiter (for example, `Mac`, `Unix`, `Windows`, or `No translation`).
Creating files	Specifies a file suffix (`xml`), and the type of encoding (for example, `ISO 10646/Unicode(UTF-8)`).
Validating files	Configures whether to warn when no grammar is specified.

Source
This includes the following options:

Formatting	Specifies a range of formatting options (for example, line width, line breaks, and indentation).
Content assist	Specifies whether to make suggestions and which strategy to use (for example, `Lax` or `Strict`).
Grammar constraints	Specifies whether to use inferred grammar in the absence of DTD/Schema.

Syntax Coloring
These settings enable you to associate specific colors with specific XML syntax elements (for example, attribute names, comment delimiters, or processing instruction content).

WS-I Settings

Before importing a WSDL file that contains the definition of a Web Service into the Web Services Repository, you can test the WSDL file for compliance with the Web Service Interoperability (WS-I) Basic Profile. The WS-I Basic Profile contains a number of *Test Assertions* that describe rules for writing WSDL files for maximum interoperability with other WSDL authors, consumers, and other related tools.

The WS-I Settings are described as follows:

WS-I Setting	***Description***
WS-I Tool Location	Use the **Browse** button to specify the full path to the Java version of the WS-Interoperability Testing tools (for example, `C:\Program Files\WSI_Test_Java_Final_1.1\wsi-test-tools`). The WS-I testing tools are used to check a WSDL file for WS-I compliance. You can download them from www.ws-i.org [http://www.ws-i.org].
Results Type	Select the type of WS-I test results that you wish to view in the generated report from the drop-down list. You can select from `all`, `onlyFailed`, `notPassed`, or `notInfo`.
Message Entry	Specify whether message entries should be included in the report using the checkbox (selected by default).
Failure Message	Specify whether the failure message defined for each test assertion should be included in the report using the checkbox (selected by default).
Assertion Description	Specify whether the description of each test assertion should be included in the report using the checkbox (unselected by default).
Verbose Output	Specify whether verbose output is displayed in the Policy Studio console window using the checkbox (unselected by default). To view the console window, select **Window** -> **Show Console** from the Policy Studio main menu.

For details on running the WS-I Testing Tools, see the *Web Service Repository* or *Global Schema Cache* topics.

Policy Studio Viewing Options

Overview

You can filter the Policy Studio navigation tree on the left of the screen to display specified tree nodes only. You can click the **Options** link at the bottom of the tree to display additional viewing options. These enable you to configure whether management services and tree node configuration types are displayed in the tree. Finally, you can configure how the Policy Studio policy filter palette is displayed on the right of the screen when editing policies.

Filtering the Tree

To filter the tree by a specific node name, enter the name in the text box above the tree. When you enter a name (for example, `SOAP Schema`), the tree is filtered automatically, and all occurrences are displayed in the tree.

Filtering the tree is especially useful in cases where many policies have been configured in the Policy Studio, and you wish to find a specific tree node (for example, a schema filter named **Check against SOAP Schema**).

Configuring Viewing Options

When you click the **Options** link at the bottom left of the navigation tree, you can configure the following viewing option:

Show Types:
Select this option to show the **Type** column in the Policy Studio navigation tree. The shows the type of each node in the tree (for example, *HTTP Service* or *Remote Host*. This option is not selected by default. When this option is selected, you can use the **Filter by type** setting.

Configuring the Policy Filter Palette

When editing policies, you can configure how the Policy Studio policy filter palette is displayed on the right of the screen. Right-click the filter palette, and select from the following options:

Layout:
Specifies how the filters are displayed in each category in the palette. By default, the filters are displayed in a list. Select one of the following options from the context menu:

- Columns
- List
- Icons Only
- Details

Customize:
The **Customize Palette** dialog enables you to customize each of the items displayed in the filter palette. Select a node in the tree on the left to display what can be customized on the right. For example, you can edit a filter name and description, specify whether it is hidden, and add tags to help searches. In addition, you can use the buttons above the tree to add or delete new category drawers or separators. You can also move a selected category drawer up or down in the palette.

Settings:
The **Palette Settings** dialog enables you to customize settings such as fonts, layout, and category drawer options (for example, close each drawer automatically when there is not enough room on the screen).

Restore Palette Defaults:
Restores all the palette settings from a default API Gateway installation.

Web Service Repository

Overview

The **Web Services Repository** stores information about Web Services whose definitions have been imported using Policy Studio or API Service Manager. The WSDL files that contain these Web Services definitions are stored together with their related XML Schemas. Clients of the Web Service can then query the repository for the WSDL file, which they can use to build and send messages to the Web Service using the API Gateway.

When you import WSDL files into the repository, this auto-generates a **Service Handler** that is used to control and validate requests to the Web Service and responses from the Web Service. You can import the WSDL file from the file system, a URL, or from a UDDI registry. You can also test the WSDL file for compliance with Web Services Interoperability (WS-I) standards. This topic describes how to test for WS-I compliance, how to import a Web Service definition into the repository, and shows what is created at each step.

Testing WS-I Compliance

Before importing the WSDL file, you can check it for compliance with the WS-I Basic Profile. The Basic Profile consists of a set of assertions and guidelines on how to ensure maximum interoperability between different implementations of Web Services. For example, there are recommendations on what style of SOAP to use (`document/literal`), how schema information is included in WSDL files, and how message parts are defined to avoid ambiguity for consumers of WSDL files.

The Policy Studio uses the Java version of the WS-Interoperability Testing Tools to test imported WSDL files for compliance with the recommendations in the Basic Profile. A report is generated showing which recommendations have passed and which have failed. While you can still import a WSDL file that does not comply with the Basic Profile, there is no certainty that consumers of the Web Service can use it without encountering problems.

Important

Before you run the WS-I compliance test, you must ensure that the Java version of the Interoperability Testing Tools is installed on the machine on which the Policy Studio is running. You can download these tools from www.ws-i.org [http://www.ws-i.org].

To configure the location of the WS-I testing tools, select **Window** -> **Preferences** from the Policy Studio main menu. In the **Preferences** dialog, select the **WS-I Settings**, and browse to the location of the WS-I testing tools. You must specify the *full path* to these tools (for example, `C:\Program Files\WSI_Test_Java_Final_1.1\wsi-test-tools`). For more details on configuring WS-I settings, see the *Policy Studio Preferences* topic.

Running the WS-I Compliance Test
To run the WS-I compliance test on a WSDL file, perform the following steps:

1. Select **Tools** -> **Run WS-I Compliance Test** from the Policy Studio main menu.
2. In the **Run WS-I Compliance Test** dialog, browse to the **WSDL File** or specify the **WSDL URL**.
3. Click **OK**. The WS-I Analysis tools run in the background in Policy Studio.

The results of the compliance test are displayed in your browser in a **WS-I Profile Conformance Report**. The overall result of the compliance test is displayed in the **Summary**. The results of the WS-I compliance tests are grouped by type in the **Artifact: description** section. For example, you can access details for a specific port type, operation, or message by clicking the link in the **Entry List** table. Each **Entry** displays the results for the relevant WS-I Test Assertions.

Registering the WSDL File

The **Web Services Repository** is displayed under the **Business Services** node in the Policy Studio tree. WSDL files are imported into Web Service Groups, which provide a convenient way of keeping groups of related Web Service definitions together. You can import a WSDL file into the default group by right-clicking the **Web Services** node, and selecting **Register Web Service**.

Alternatively, you can add a new Web Services group by right-clicking the default **Web Services** group, or the **Web Services Repository** node, and selecting **Add a new Web Services group**. When the new group is added, you can right-click it in the tree, and select **Register Web Service**.

Loading the WSDL File

In the **Import WSDL** wizard, the **Load WSDL** screen enables you to choose the WSDL location from the following options:

- File system
- URL
- UDDI registry

Select the appropriate option depending on the location of the WSDL that you wish to import. If you wish to retrieve a WSDL file from a UDDI directory, see the *Retrieving WSDL Files from a UDDI Registry* topic.

Click **Next**.

Selecting WSDL Operations

The **WSDL Operations** screen of the wizard displays all operations defined in the WSDL file. The **Relative Path**, **Binding**, and **Namespace** of each operation are also displayed. Select the operations that you wish to create policy resolvers for. The Policy Studio uses the Web Service location, SOAP operation, and SOAP Action specified in the WSDL to create **Relative Path**, **SOAP Operation**, and **SOAP Action** policy resolving filters in the **Service Handler**.

Use the **Remove unselected operations from the WSDL** checkbox to specify whether the Policy Studio removes unselected operations from the WSDL file stored in the repository. As a result, removed operations are not exposed to clients that download the WSDL file for the Web Service.

Important

When a request is made to the API Gateway for WSDL that has been imported into the **Web Services Repository**, it changes the address of the Web Service specified in the `location` attribute of the `<soap:address>` element to point to the machine on which the API Gateway is running, instead of the machine hosting the Web Service. This means when a client downloads WSDL for the Web Service, it routes messages through the API Gateway instead of sending messages directly to the Web Service, which is typically not available on a public IP address. For more details, see the section called “Publishing the WSDL”.

Click **Next**.

WS-Policy Options

Configure the following options on this screen:

Secure this virtualized service with a WS-Policy
Specifies whether to use a WS-Policy to secure the Web Service being virtualized by the API Gateway. When this setting is selected, the **Secure Virtual Service** dialog is displayed after the **Import WSDL** wizard. This dialog enables you to configure a WS-Policy that the API Gateway enforces on messages that it receives from the client. This setting is unselected by default.

Use the WS-Policy in the WSDL to connect securely to the back-end Web Service
When the WSDL file includes WS-Policy information, this setting specifies whether to use this WS-Policy to connect securely to the Web Service. This setting is enabled (and selected by default) only if the selected WSDL file includes WS-Policy information.

Click **Next**.

Deploy Policy

The final screen in the wizard enables you to select where to deploy the newly created policy (or policies). The **Deploy Policy** screen displays a list of all available Services and their corresponding Relative Paths. Select a Relative Path under which the policy is deployed. All requests arriving on the selected path are dispatched to the newly created policy.

Click **Finish**.

Secure Virtual Service

When you click **Finish** in the wizard, the **Secure Virtual Service** dialog is displayed. This enables you to specify policies to enforce security between a client and the API Gateway. For more details, see *Securing a Virtual Service using Policies*.

In addition, if the imported WSDL file contains WS-Policy assertions, you are prompted to configure settings to enforce security between the API Gateway and the Web Service. For more details, see *Configuring Security Policies from WSDL Files*.

WSDL Import Summary

When you have finished configuring security policies, the **Summary** dialog displays details on the imported WSDL file. For example, this includes the location of the WSDL file, and a tab for each Web Service virtualized by the API Gateway. Each tab includes the path to the Web Service that is published by the API Gateway.

WSDL Import Summary Options
The **Summary** dialog also enables you to configure the following options on the tab for each Web Service:

Validation	If you wish to use a dedicated validation policy for all messages sent to the Web Service, select this checkbox, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom validation policy used by multiple Web Services.
Routing	If you wish to use a dedicated routing policy to send all messages on to the Web Service, select this checkbox, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom routing policy used by multiple Web Services.
WSDL Access Options	Select whether to make the WSDL for this Web Service available to clients. **Allow the API Gateway to publish WSDL to clients** is selected by default. The published WSDL represents a virtualized view of the Web Service. Clients can retrieve the WSDL from the API Gateway, generate SOAP requests, and send them to the API Gateway, which routes them on to the Web Service. For more details, see the section called “Publishing the WSDL”.

These options enable you to configure the underlying auto-generated Service Handler (**Web Service Filter**) without having to navigate to it in the **Policies** tree. These are the most commonly modified **Web Service Filter** options after importing a WSDL file. Changes made in the **Summary** dialog are visible in the underlying **Web Service Filter**. For more details, see the *Web Service Filter* topic.

You can also access these options under the **Listeners** node after the WSDL file has been imported. Right-click the appropriate Web Service Resolver node, and select **Quick-Edit Policy** to display a dialog that enables you to configure these options.

What is Created?

What is created when you import the WSDL file depends on whether you configured a policy to enforce security between the client and the API Gateway, and whether the WSDL file contains any WS-Policy annotations. However, assuming that the default options are selected in the **WSDL Import** wizard, the following list summarizes what is created by the wizard:

Web Service
A new Web Service tree node is created for each imported WSDL file in the **Policies** tree. This tree node contains the WSDL file, together with any imported resources, such as other WSDL files or schema files. Click the new Web Service node to view the list of imported WSDL files and XML Schemas. The Schemas are listed by namespace. You can view the imported WSDL file by clicking the location of the WSDL file under the Web Service node. If you have selected to remove unselected operations from the WSDL, these operations are removed from the stored WSDL.

Web Service Resolver
A new Web Service Resolver node is created for each imported Web Service in the **Services** tree view. The Web Service Resolver is used to identify messages destined for this Web Service, and to map them to the **Service Handler** (**Web Service Filter**) for the Web Service.

Service Handler
A **Service Handler** for the Web Service is created under the generated policy in the **Policies** tree. This is used to control and validate requests to the Web Service and responses from the Web Service. The **Service Handler** is named after the Web Service (for example, **Service Handler for 'HelloWorldService'**). The **Service Handler** is a **Web Service Filter**, and is used to control the following:

- Routing
- Validation
- Message request/response processing
- WSDL options
- Monitoring options

For more details, see the *Web Service Filter* topic.

Policy Container
A container for the newly generated policies is created under the **Generated Policies** node in the **Policies** tree. The new container is named after the service (for example, **Web Services.HelloWorldService**).

Policy
A policy for the Web Service is created in the generated policy container. The policy name includes the relative path to the service (for example, **/HelloWorldService/HelloWorld**). Clients can specify `WSDL` on a request query string to retrieve the WSDL file (for example, `http://localhost:8080/HelloWorldService/HelloWorld?WSDL`). For more details, see the section called “Publishing the WSDL”.

Security Policies
If you decided to configure a WS-policy to enforce security between the client and the API Gateway (as described earlier in the section called “Secure Virtual Service”), or if the imported WSDL file contains WS-Policy assertions, a number of additional policies are automatically created in the generated policy container. These generated policies include the fil-

ters required to generate and/or validate the relevant security tokens (for example, SAML tokens, WS-Security Username tokens, and WS-Addressing headers). These policies perform the necessary cryptographic operations (for example, signing/verifying and encryption/decryption) to meet the security requirements of the specified policies.

Publishing the WSDL

When the WSDL has been imported into the **Web Services Repository**, it can be retrieved by clients. In effect, by importing the WSDL into the repository, you are *publishing* the WSDL. In this way, consumers of the services defined in the WSDL can learn how to communicate with those services by retrieving the WSDL for those services. However, to do this, the location of the service must be changed to reflect the fact that the API Gateway now sits between the client and the defined service.

For example, assume that the WSDL file states that a particular service resides at `http://www.example.com/services/myService`:

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://www.example.com/services/myService"/>
  </port>
</service>
```

When deployed behind the API Gateway, this URL is no longer accessible to consumers of the service. Because of this, clients must send SOAP messages through the API Gateway to access the service. In other words, they must now address the machine hosting the API Gateway instead of that directly hosting the service.

When the WSDL file has been published to the repository, clients can retrieve it. However, when returning the WSDL to the client, the API Gateway dynamically changes the value of the `location` attribute in the `service` element in the WSDL file to point to the machine on which the API Gateway resides. The details regarding the physical location of the Web Service are preserved in the **Connection** filters, responsible for routing messages on to the service.

Assuming that the API Gateway is running on port `8080` on a machine called `SERVICES`, the location specified in the exported WSDL file is changed to the following:

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://SERVICES:8080/services/myService"/>
  </port>
</service>
```

When the modified WSDL file is distributed to the client, it now routes messages to the machine hosting the API Gateway instead of attempting to directly access the Web Service.

Accessing the WSDL
For the client to access this modified WSDL file, the Policy Studio provides a WSDL retrieval facility whereby clients can query the **Web Services Repository** for the WSDL file for a particular Web Service. To do this, the client must pass the name `WSDL` on the query string to the Relative Path mapped to the policy for this Web Service.

For example, if the policy is deployed under `http://SERVICES:8080/services/getQuote`, the client can retrieve the WSDL for this Web Service by sending a request to `http://SERVICES:8080/services/getQuote?WSDL`. When the client has a copy of the updated WSDL file, it knows how to create correctly formatted messages for the service, and more importantly, it knows to route messages to the API Gateway rather than to the Web Service directly.

Publishing to UDDI
For details on how to publish a WSDL file registered in the Web Services Repository to a UDDI directory, see the *Publishing WSDL Files to a UDDI Registry* topic.

Setting the Encryption Passphrase

Encryption Passphrase Overview

By default, API Gateway configuration data is stored unencrypted. However, you can encrypt certain sensitive information, such as passwords and private keys, using a passphrase. When the passphrase has been set (and the data has been encrypted with it), you must enter the passphrase when connecting to the API Gateway with the Policy Studio, or when the API Gateway is starting up, so that the encrypted data can be decrypted. This passphrase is set at the API Gateway group level.

Warning

It is *crucial* that you remember the passphrase when you change it. Failure to remember the passphrase results in the loss of private key data.

This topic describes how to specify the group passphrase when connecting to the API Gateway with the Policy Studio, in your API Gateway configuration file, or when the API Gateway is starting up. It also describes how to change the group passphrase when it has been set initially.

Setting the Group Passphrase in the Policy Studio

You can use the the Policy Studio topology view to set the group passphrase to encrypt the data. This is the table displayed when you connect to the Admin Node Manager. To change the passphrase, right-click the API Gateway group name in the table (for example, **QuickStart Group**), and select **Change Passphrase**.

Complete the following fields on the **Change Group Passphrase** dialog:

Old Passphrase:
Enter the old passphrase that you wish to change in this field. Alternatively, you can leave this field blank if you are setting the passphrase for the first time.

New Passphrase:
Enter the new passphrase.

Confirm New Passphrase:
Re-enter the new passphrase to confirm it.

Entering the Group Passphrase in the Policy Studio

When you have set the encryption passphrase for the API Gateway configuration data, you must specify this passphrase every time that you connect to the API Gateway in the Policy Studio. You can enter it in the **Passphrase** field of the **Open File** dialog, which is displayed when connecting to a configuration file. Alternatively, you can enter it in the **Enter Passphrase** dialog, which is displayed before editing an active server configuration.

Note

The different roles of the **Passphrase** and the **Password** fields are as follows:

Passphrase	Used to decrypt sensitive data (for example, private keys) that have already been encrypted. Not required by default, and only needed if you have set the group passphrase in Policy Studio.

Password	Used to authenticate to the API Gateway's management interface using HTTP basic authentication when opening a connection to a server. Required by default.

Specifying the Passphrase in a File or on Startup

For the API Gateway to read (decrypt) encrypted data from its configuration, it must be primed with the passphrase key. You can do this using the Policy Studio, as explained in the previous section. You can also specify the passphrase directly in a configuration file, or prompt for it at startup.

Specifying the Node Manager Passphrase in a Configuration File
You can specify a passphrase directly in the Node Manager's configuration file. Open the following file in your API Gateway installation:

```
INSTALL_DIR/system/conf/nodemanager.xml
```

This file contains values for general system settings, such as the server name and trace level, and also (if required) the passphrase key to use to decrypt encrypted API Gateway configuration data.

Typically, the passphrase is only entered directly in the file if the server must be started as a Windows service or UNIX daemon. In this case, the administrator cannot enter the passphrase manually when the server is starting. To avoid this, you must enter the passphrase in the configuration file. You should specify the passphrase as the value of the `secret` attribute as follows, where `"myPassphrase"` is the encryption passphrase:

```
secret="myPassphrase"
```

Specifying the API Gateway Passphrase in a Configuration File
You can also specify the passphrase for individual API Gateway instances created using the `managedomain` script. To do this, specify the `secret` attribute in the `service.xml` file for your API Gateway instance. For example:

```
INSTALL_DIR/groups/group-id/instance-id/conf/service.xml
```

Prompting for the Passphrase on Server Startup
If you do not wish to specify the passphrase directly in the Node Manager or API Gateway configuration file, and do not need to start as a Windows service or UNIX daemon, you can configure the Node Manager or API Gateway to prompt the administrator for the passphrase on the command line when starting up. To do this, enter the `"(prompt)"` special value for the `secret` attribute as follows:

```
secret="(prompt)"
```

Important

If you use this option, you must take care to remember the encryption passphrase. Failure to use the correct passphrase results in loss of private key data, and may prevent the API Gateway from functioning correctly.

For more details, see the *Oracle API Gateway Installation and Configuration Guide*.

Default Settings

Overview

The **Default Settings** screen enables you to set several global configuration settings to optimize the behavior of the API Gateway for your environment.

To configure the **Default Settings**, in the Policy Studio main menu, select **Tasks** -> **Manage Settings** -> **Default Settings**. Alternatively, in the Policy Studio tree, select the **Settings** node, and click the **Default Settings** tab.

After changing any settings, you must deploy to the API Gateway for the changes to be enforced. You can do this in the Policy Studio main menu by selecting **Server** -> **Deploy**. Alternatively, click the **Deploy** button in the toolbar, or press F6.

Settings

You can configure the following settings in the **Default Settings** screen:

Setting	***Description***
Trace Level	Enables you to set the trace level for the API Gateway at runtime. Select the appropriate option from the **Trace Level** drop-down list. Defaults to `INFO`.
Active Timeout	When the API Gateway receives a large HTTP request, it reads the request off the network when it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout** specified in milliseconds, the API Gateway closes the connection. This guards against a host closing the connection in the middle of sending data. For example, if the host's network connection is pulled out of the machine while in the middle of sending data to the API Gateway. When the API Gateway has read all the available data off the network, it waits the **Active Timeout** period before closing the connection. Defaults to `30000` milliseconds. **Note** You can configure this setting on a per-host basis using the **Remote Hosts** interface.
Date Format	Configures the format of the date for the purposes of tracing, logging, and reporting. Defaults to `MM.dd.yyyy HH:mm:ss,SSS`. For more information, see `http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html`
Cache Refresh Interval	Configures the number of seconds that the server caches data loaded from an external source (external database, LDAP directory, and so on) before refreshing the data from that source. The default value is `5` seconds. If you do not wish any caching to occur, set this value to `0`.
Idle Timeout	The API Gateway supports HTTP 1.1 persistent connections. The **Idle Timeout** specified in milliseconds is the time that the API Gateway waits after sending a message over a persistent connection before it closes the connection. Typically, the host tells the API Gateway that it wants

Setting	*Description*
	to use a persistent connection. The API Gateway acknowledges this instruction and decides to keep the connection open for a certain amount of time after sending the message to the host. If the connection is not reused within the **Idle Timeout** period, the API Gateway closes the connection. Defaults to `15000` milliseconds. **Note** You can configure this setting on a per-host basis using the **Remote Hosts** interface.
LDAP Service Provider	Specifies the service provider used for looking up an LDAP server (for example, `com.sun.jndi.ldap.LdapCtxFactory`). The provider is typically used to connect to LDAP directories for certificate and attribute retrieval.
Maximum Memory per Request	The maximum amount of memory allocated to each request. **Note** You can configure this setting on a per-host basis using the **Remote Hosts** interface.
Realm	Specifies the realm for authentication purposes.
Schema Pool Size	Sets the size of the Schema Parser pool.
Server Brand	Specifies the branding to be used in the API Gateway.
SSL Session Cache	Specifies the number of idle SSL sessions that can be kept in memory. You can use this setting to improve performance because the slowest part of establishing the SSL connection is cached. A new connection does not need to go through full authentication if it finds its target in the cache. Defaults to `32`. If there are more than 32 simultaneous SSL sessions, this does not prevent another SSL connection from being established, but means that no more SSL sessions are cached. A cache size of `0` means no cache, and no outbound SSL connections are cached.
Token Drift Time	Specifies the number of seconds drift allowed for WS-Security tokens. This is important in cases where the API Gateway is checking the date on incoming WS-Security tokens. It is likely that the machine on which the token was created is out-of-sync with the machine on which the API Gateway is running. The drift time allows for differences in the respective machine clock times.
Allowed number of operations to limit XPath transforms	Specifies the total number of node operations permitted in XPath transformations. Complex XPath expressions (or those constructed together with content to produce expensive processing) might lead to a denial-of-service risk. Defaults to `4096`.
Input Encodings	Click the browse button to specify the HTTP content en-

Setting	*Description*
	codings that the API Gateway can accept from peers. The available content encodings include `gzip` and `deflate`. For more details, see the topic on *Compressed Content Encoding*.
Output Encodings	Click the browse button to specify the HTTP content encodings that the API Gateway can apply to outgoing messages. The available content encodings include `gzip` and `deflate`. For more details, see the topic on *Compressed Content Encoding*.
Server's SSL cert's name must match name of requested server	Ensures that the certificate presented by the server matches the name of the host address being connected to. This prevents host spoofing and man-in-the-middle attacks. This setting is enabled by default.
Send desired servername to server during TLS negotiation	Specifies whether to add a field to outbound TLS/SSL calls that shows the name that the client used to connect. For example, this can be useful if the server handles several different domains, and needs to present different certificates depending on the name that the client used to connect.

Namespace Settings

Overview

The API Gateway exposes global settings that enable you to configure which versions of the SOAP and WSSE specifications it supports. You can also specify which attribute is used to identify the XML Signature referenced in a SOAP message.

To configure the namespace settings, in the Policy Studio tree, select the **Settings** node, and click the **Namespace** tab at the bottom of the screen. Alternatively, in the Policy Studio main menu, select **Tasks** -> **Manage Settings** -> **Namespace**.

SOAP Namespace

The **SOAP Namespace** tab can be used to configure the SOAP namespaces that are supported by the API Gateway. In a similar manner to the way in which the API Gateway handles WSSE namespaces, the API Gateway will attempt to identify SOAP messages belonging to the listed namespaces in the order given in the table.

The default behavior is to attempt to identify SOAP 1.1 messages first, and for this reason, the SOAP 1.1 namespace is listed first in the table. The API Gateway will only attempt to identify the message as a SOAP 1.2 message if it can't be categorized as a SOAP 1.1 message first.

Signature ID Attribute

The **Signature ID Attribute** tab allows you to list the supported attributes that can be used by the API Gateway to identify a Signature reference within an XML message.

An XML-signature `<signedInfo>` section may reference signed data via the `URI` attribute. The `URI` value may contain an id that identifies data in the message. The referenced data will hold the "URI" field value in one of its attributes.

By default, the server uses the `Id` attribute for each of the WSSE namespaces listed above to locate referenced signed data. The following sample XML Signature illustrates the use of the `Id` attribute:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
    ...
    <dsig:Reference URI="#Oracle:sLmDCph3tGZ10">
      ...
    </dsig:Reference>
   </dsig:SignedInfo>
    ....
  </dsig:Signature>
 </soap:Header>
 <soap:Body>
  <getProduct wsu:Id="Oracle:sLmDCph3tGZ10"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
   <Name>SOA Test Client</Name>
   <Company>Company</Company>
  </getProduct>
 </soap:Body>
</soap:Envelope>
```

It is clear from this example that the Signature reference identified by the `URI` attribute of the `<Reference>` element refers to the nodeset identified with the `Id` attribute (the `<getProduct>` block).

Because different toolkits and implementations of the XML-Signature specification can use attributes other than the `Id` attribute, the API Gateway allows the user to specify other attributes that should be supported in this manner. By default, the API Gateway supports the `Id`, `ID`, and `AssertionID` attributes for the purposes of identifying the signed content within an XML Signature.

However you can add more attributes by clicking the **Add** button and adding the attribute in the interface provided. The priorities of attributes can be altered by clicking the **Up** and **Down** buttons. For example, if most of the XML Signatures processed by the API Gateway use the `ID` attribute, this attribute should be given the highest priority.

WSSE Namespace

The **WSSE Namespace** tab is used to specify the WSSE (and corresponding WSSU) namespaces that are supported by the API Gateway.

The API Gateway attempts to identify WS Security blocks belonging to the WSSE namespaces listed in this table. It first attempts to locate Security blocks belonging to the first listed namespace, followed by the second, then the third, and so on until all namespaces have been utilized. If no Security blocks can be found for any of the listed namespaces, the message will be rejected on the grounds that the API Gateway does not support the namespace specified in the message. To add a new namespace, click the add button.

Note

Every WSSE namespace has a corresponding WSSU namespace. For example, the following WSSE and WSSU namespaces are inextricably bound:

WSSE Namespace	`http://schemas.xmlsoap.org/ws/2003/06/secext`
WSSU Namespace	`http://schemas.xmlsoap.org/ws/2003/06/utility`

First, enter the WSSE namespace in the **Name** field. Then enter the corresponding WSSU namespace in the **WSSU Namespace** field.

MIME/DIME Settings

Overview

The MIME/DIME settings list a number of default common content types that are used when transmitting MIME messages. You can configure the API Gateway's **Content Type** filter to accept or block messages containing specific MIME types. Therefore, the contents of the MIME types library act as the set of all MIME types that the API Gateway can filter messages with.

All of the MIME types listed in the table are available for selection in the **Content Type** filter. For example, you can configure this filter to accept only XML-based types, such as `application/xml`, `application/*+xml`, `text/xml`, and so on. Similarly, you can block certain MIME types (for example, `application/zip`, `application/octet-stream`, and `video/mpeg`). For more details on configuring this filter, see the *Content Type Filtering* filter topic.

Configuration

To configure the MIME/DIME settings, in the Policy Studio main menu, select **Tasks** -> **Manage Settings** -> **MIME/DIME**. Alternatively, in the Policy Studio tree, select the **Settings** node, and click the **MIME/DIME** tab at the bottom of the screen.

The MIME/DIME settings screen lists the actual MIME types on the left column of the table, together with their corresponding file extensions (where applicable) in the right column.

To add a new MIME type, click the **Add** button. In the **Configure MIME/DIME Type** dialog, enter the new content type in the **MIME or DIME Type** field. If the new type has a corresponding file extension, enter this extension in the **Extension** field. Click the **OK** button when finished.

Similarly, you can edit or delete existing types using the **Edit** and **Delete** buttons.

Session Settings

Overview

The **Session Settings** tab enables you to configure session management settings for the selected cache. For example, you can configure the period of time before expired sessions are cleared from the `HTTP Sessions` cache, which (selected by default).

To configure session settings, select the **Settings** node in the Policy Studio tree, and click the **Session Settings** tab at the bottom of the screen. Alternatively, in the Policy Studio main menu, select **Tasks** -> **Manage Settings** -> **Session Settings**.

Configuration

Configure the following session settings:

Cache:
Specifies the cache that you wish to configure. Defaults to `HTTP Sessions`. To configure a different cache, click the button on the right, and select the cache to use. The list of currently configured caches is displayed in the tree.

To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Clear Expired Sessions Period:
Enter the number of seconds before expired sessions are cleared from the selected cache. Defaults to `60`.

Exporting API Gateway Configuration

Overview

You can export API Gateway configuration data by right-clicking a Policy Studio tree node (for example, policy or policy container), and selecting the relevant export menu option (for example, **Export Policy**). The configuration is exported to an XML file, which you can then import into a different API Gateway configuration. This is useful if you have configured the API Gateway in a testing environment and wish to move this configuration to a live production environment. By exporting configuration data from a test environment, and importing into a production environment, you can effectively migrate your API Gateway configuration. This is the recommended way to export API Gateway configuration data, and enables you to manage references between configuration components.

For details on importing configuration data, see *Importing API Gateway Configuration*.

What is Exported

You can export API Gateway configuration items by right-clicking a node in the Policy Studio tree. For example, this includes the following Policy Studio tree nodes:

- Policies
- Policy Containers
- Schemas
- Alerts
- Caches
- Regular Expressions (White List)
- Attacks (Black List)
- Users
- Certificates
- Relative Paths
- Remote Hosts
- Database Connections

In addition, you can also export configuration items that are associated with the selected tree node. For example, this includes referenced policies, MIME types, regular expressions, schemas, and remote hosts. For details on exporting additional configuration items, see the next section.

Exporting Configuration Items

To export API Gateway configuration items, perform the following steps:

1. Right-click a Policy Studio tree node (for example, policy or policy container), and select the relevant menu option (for example, **Export Policy**).
2. The first screen in the export wizard is a read-only screen that displays the configuration items to be exported. The **Exporting** tree displays the selected tree node (in this case, policy), which is exported by default. **The following configuration items will also be exported** tree includes additional referenced items that are also exported by default along with the policy (for example, MIME types, regular expressions, and schemas).
3. You can click **Finish** if this selection suits your requirements. Otherwise, click **Next** to refine the selection.
4. In the next screen, you can select optional configuration items for export. The **Additional configuration items that may be exported** tree on the left includes dependent items that are not exported by default. For example, these include the following:
   - Outbound references: configuration items directly referenced out from the export set to other configuration stores (for example, Certificates, Users, or External Connections).

- Inbound references: configuration items in other configuration stores that directly reference items in the export set.
- Associated configuration directly related to the export set (for example, Remote Hosts or Relative Paths).

5. To add an item for export, select it in the **Additional configuration that may be exported** tree on the left, and click **Add**.
6. To remove an item for export, select it in the **Additional configuration that will be exported** tree on the right, and click **Remove**.

Note

The original set of items in the **Additional configuration that will be exported** tree cannot be removed. Only items added from the **Additional configuration that may be exported** tree can be removed.

7. By default, items displayed in the **Additional configuration that may be exported** tree are scoped to direct references to the export set (inbound, outbound, and associated). You can select **Display additional configuration that depends on items to be exported** to recursively add references to this tree when additional configuration items are added to the export set.
8. Click **OK** to export the selected configuration.

Referenced Policies
When exporting a policy or policy container, by default, any policies referenced by the policy are included for export and displayed in the **Additional configuration that will be exported** list.

Exporting All API Gateway Configuration

To export all API Gateway configuration data, perform the following steps:

1. Click **Export Configuration** button in the Policy Studio toolbar.
2. In the **Save As** dialog, specify a file name, and click **Save** to export the entire API Gateway configuration data to a file. This includes all references between configuration components.

Importing API Gateway Configuration

Overview

You can import configuration data into your API Gateway configuration (for example, policies, certificates, and users). This is useful if you have configured the API Gateway in a testing environment and want to move this configuration to a live production environment. By exporting configuration data from a test environment, and importing into a production environment, you can effectively migrate your API Gateway configuration. This is the recommended way to import API Gateway configuration data, and enables you to manage references between configuration components.

For details on exporting configuration data, see *Exporting API Gateway Configuration*.

Importing Configuration

To import API Gateway configuration data, perform the following steps:

1. Click the **Import Configuration** button in the Policy Studio toolbar.
2. Browse to the location of the XML file that contains the previously exported configuration data that you wish to import.
3. Select the XML file, and click **Open**.
4. If a passphrase was set on the configuration from which the data was previously exported, enter it in the **Enter Passphrase** dialog, and click **OK**.
5. In the **Import Configuration** dialog, all configuration items are selected for import by default. If you do not wish to import specific items, unselect them in the tree. For more details, see Viewing Differences.
6. Click **OK** to import the selected configuration items.
7. The selected configuration items are imported into your API Gateway configuration and displayed in the Policy Studio tree. For example, any imported policies and containers are displayed under the **Policies** node.

Important

Be careful when deselecting configuration nodes for import. Unselecting certain nodes may make the imported configuration inconsistent by removing supporting configuration.

Viewing Differences

The **Import Configuration** dialog displays the differences between the existing stored configuration data (destination) and the configuration data to be imported (source). Differences are displayed in the tree as follows:

Addition	Exists in the source Configuration being imported but not in the destination Configuration. Displayed as a green plus icon.
Deletion	Exists in the destination Configuration but not in the source Configuration being imported. Displayed as a red minus icon.
Conflict	Exists in both Configurations but is not the same. Displayed as a yellow warning icon.

If you select a particular node in the **Import Configuration** tree, the **Differences Details** panel at the bottom of the screen shows details for this Configuration entity (for example, added or removed fields). In the case of conflicts, changed fields are highlighted. Some Configuration entities also contain references to other entities. In this case, an icon is displayed for the field in the **Difference Details** panel. If you double-click a row with an icon, you can drill down to view further **Difference Details** dialogs for those entities.

What is Imported

When configuration data is imported, some configuration items are imported in their entirety. For example, if the contents of a particular policy are different, the entire policy is replaced (new filters are added, missing filters are removed, and conflicting filters are overwritten). In addition, if a complex filter differs in its children, child items are removed and added as required (for example, WS Filter, Web Service, User, and so on). Other imports are additive only. For example, importing a single certificate does not remove the certificates already in the destination Certificate store. All references to other policies are also maintained during import.

Important

Although importing some configuration items removes child items by default, you can deselect child nodes to keep existing child items. However, you should take care to avoid inconsistencies. The default selection applies in most cases.

Configuring the API Gateway for API Gateway Analytics

Overview

This topic explains how to configure the API Gateway to store message metrics in the same reports database used by Oracle API Gateway Analytics. It also includes how to configure database logging and monitoring settings.

Important

To view API Gateway metrics in Oracle API Gateway Analytics, you must configure the API Gateway for use with Oracle API Gateway Analytics.

Prerequisites
This topic assumes that you have already installed and configured Oracle API Gateway Analytics using the steps described in the *Oracle API Gateway Installation and Configuration Guide*.

Connecting to the API Gateway

To connect to the API Gateway in Policy Studio, perform the following steps:

1. Ensure the Admin Node Manager and API Gateway are running. For more details, see *Startup Instructions*.
2. On the Policy Studio welcome page, click the Admin Node Manager server session to make a connection.
3. Specify your connection details (host, port, user name, and password). The default connection URL is:

   ```
   https://localhost:8090/api
   ```

   where `HOST` points to the IP address or hostname of the machine on which the API Gateway is running. For more details, see *Connection Details*.
4. In the API Gateway topology view, double-click the **API Gateway** instance to load its configuration.

Configuring the Database Connection

To configure the database connection, perform the following steps:

1. Expand the **External Connections** -> **Database Connections** node in the Policy Studio tree.
2. Right-click the **Default Database Connection** tree node, and select **Edit**.
3. Configure the database connection to point to the same reports database created when API Gateway Analytics was installed. For more details, see *Database Connection*.

You can verify that your database connection is configured correctly by clicking the **Test Connection** button on the **Configure Database Connection** dialog. You can also view the contents of your server `.trc` file in the `INSTALL_DIR/trace` directory. For more details on tracing, see *Troubleshooting*.

Configuring the Database Logging

To configure the database logging settings, perform the following steps:

1. In the Policy Studio tree view, select the **Settings** node.
2. Select the **Audit Log** tab at the bottom, select the **Database** tab, select **Enable logging to database**.
3. Select the **Default Database Connection** from the drop-down list if appropriate. Alternatively, select a database connection that you have configured. You must ensure that your database connection points to the same reports

database configured when API Gateway Analytics was installed. For more details, see *Database Connection*.

Note

If you wish to write the contents of message requests and responses to the database, you must configure the **Log Message Payload** filter for the relevant policies (for example, at the start and end of the policy). For more details, see *Log Message Payload*.

Configuring Monitoring Settings

To configure the API Gateway to monitor traffic and store message metrics in the reports database, perform the following steps:

1. In the Policy Studio tree, select the **Settings** node, and select the **Metrics** tab at the bottom of the screen.
2. Ensure **Enable real time monitoring** is selected.
3. In the **Reports settings** panel, select **Store real time monitoring data for charts/reports**, and specify the database connection that points to the reports database.
4. If you wish to enable monitoring of traffic per API Gateway, select the **Traffic Monitor** tab at the bottom of the screen, and ensure **Enable Traffic Monitor** is selected.

Important

Enabling traffic monitoring has a negative impact on performance. If you wish to maximize performance, do not enable this setting. For more details, see *Configuring Traffic Monitoring*.

Deploying to the API Gateway

You must deploy these configuration changes to the API Gateway. Click the **Deploy** button in the toolbar, or press F6. The API Gateway now sends audit trail and message metrics data to the reports database. This database is then queried by API Gateway Analytics to produce reports showing system health, service usage, clients, message size and volume, and so on.

Note

These monitoring settings are API Gateway instance-wide. You can also send metrics data to the database at the interface level. Right-click the HTTP Services Group (for example, **Default Services**), and select **Edit**. To monitor traffic for this Services Group, ensure that **Include in real-time monitoring** is selected. Similarly, you can disable monitoring for this Services Group by unselecting this setting.

Using Oracle API Gateway Analytics

Overview

Oracle API Gateway Analytics monitors, records, and reports on the history of message traffic between API Gateway instances and various services, remote hosts, and clients running in a managed domain. You can use API Gateway Analytics to monitor traffic and perform root cause analysis at the level of the domain, API Gateway instance, service, remote host, and client. You can also filter the display based on any selected time period. For example, this defaults to the last 7 days, but you can specify any date range.

This topic describes how to use the views available in the Oracle API Gateway Analytics web-based interface to monitor your domain. It assumes that you have already performed the steps described in *Configuring the API Gateway for API Gateway Analytics*.

Launching API Gateway Analytics

To launch API Gateway Analytics, perform the following steps:

1. Start the API Gateway Analytics server using the `oaganalytics` script in the `/bin` directory of your API Gateway Analytics installation.
2. Using the default port, connect to the API Gateway Analytics interface in a browser at the following URL:

   ```
   http://HOST:8040/
   ```

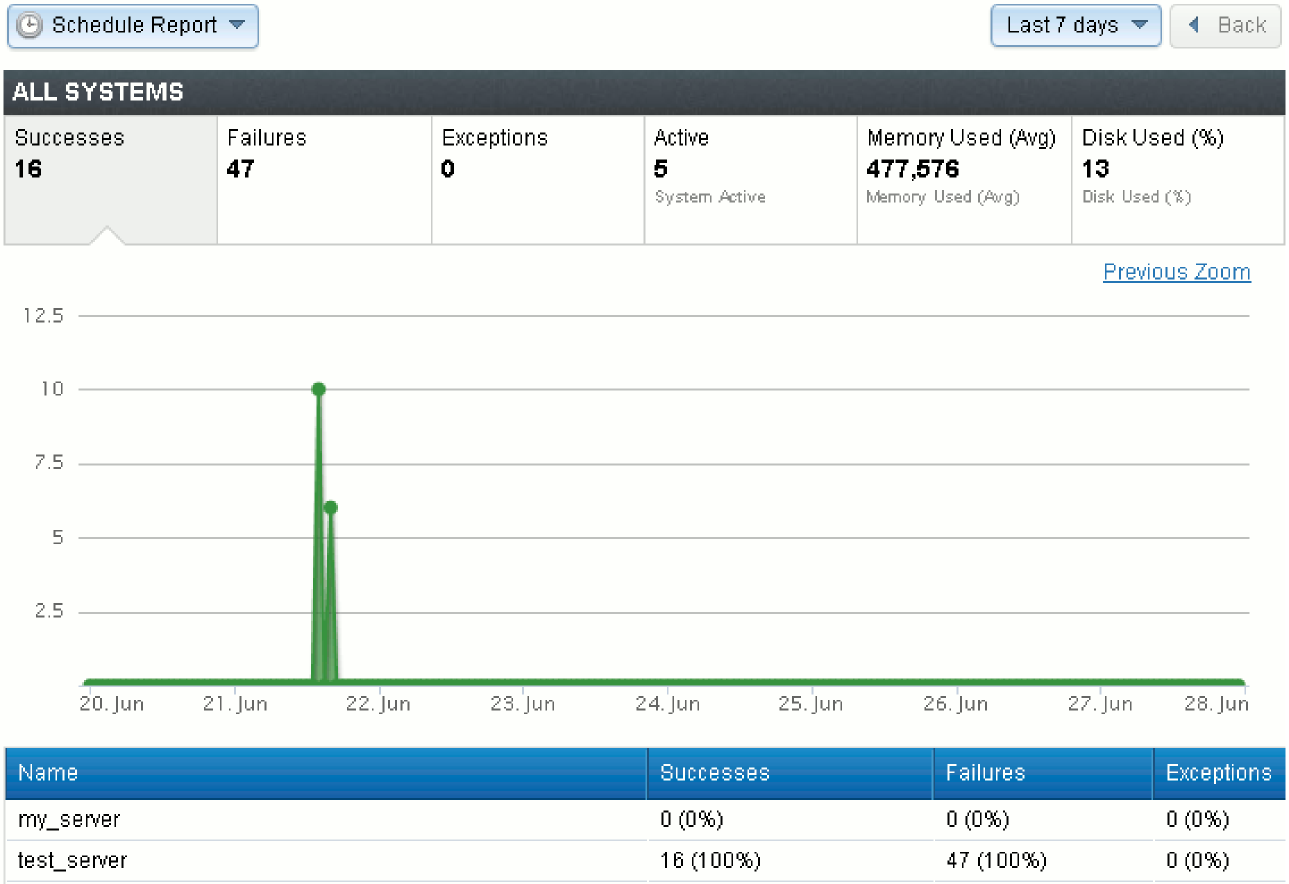
   where `HOST` points to the IP address or hostname of the machine on which API Gateway Analytics is installed.
3. Log in using the default `admin` user with password `changeme`. You can edit this user in Policy Studio using the **Users** interface from the Policy Studio tree view.

System

In the API Gateway Analytics **System** view, click a panel in the **ALL SYSTEMS** section at the top to display graph for the selected system-level metrics below. For example, the available metrics include the following:

- **Successes**
- **Failures**
- **Exceptions**
- **Active**
- **Memory Used (Avg)**
- **Disk Used (%)**

The following example shows messages successfully sent displayed in a simple domain with two API Gateway instances:



Successes	Failures	Exceptions	Active	Memory Used (Avg)	Disk Used (%)
16	47	0	5	477,576	13
			System Active	Memory Used (Avg)	Disk Used (%)

Name	Successes	Failures	Exceptions
my_server	0 (0%)	0 (0%)	0 (0%)
test_server	16 (100%)	47 (100%)	0 (0%)

The table at the bottom shows all the API Gateway instances that are sending monitored traffic to protected services, clients, and remote hosts in your domain. You can click a API Gateway instance in the table to drill down and view graphs for the selected instance (for example, **SYSTEM - Test Server**). You can click the **Back** button at the top right to return to the **ALL SYSTEMS** view.

Note

Each of the API Gateway instances must already be configured to store message metrics in the same database that API Gateway Analytics is configured to read from. This enables API Gateway Analytics to obtain the API Gateway metrics and logs from the database for display. For more details, see the chapter on configuring the database in the *Oracle API Gateway Installation and Configuration Guide*.

API Services

The **API Services** view shows metrics for services that are virtualized by API Gateway instances in your domain. You can virtualize services using the web-based API Service Manager tool and the Oracle Policy Studio. For details, see *Managing API Services*.

In the **API Services** view, click a panel in the **ALL API SERVICES** section at the top to display graph for the selected service-level metric below. For example, the available metrics include the following:

- **Messages**

- **Successes**
- **Failures**
- **Exceptions**
- **Processing Time (Min)**
- **Processing Time (Max)**
- **Processing Time (Avg)**

The table at the bottom shows all the services protected by API Gateway instances in your domain. You can click a service in the table to drill down and view graphs for the selected service (for example, **API SERVICE - Google Search**). You can click the **Back** button at the top right to return to the **ALL API SERVICES** view.

Note

A service must have been sent a message before it is displayed in the **API Services** view.

Remote Hosts

The **Remote Hosts** view displays metrics for all the remote hosts that have been configured in your domain. It shows details such as the number of message transactions that have been sent to this remote host, together with the total number of bytes sent to and received from this host.

In the **Remote Hosts** view, click a panel in the **ALL REMOTE HOSTS** section at the top to display graph for the selected remote host metric below. For example, the available metrics include the following:

- **Transactions**
- **Volume Bytes (In)**
- **Volume Bytes (Out)**
- **Response Time (Min)**
- **Response Time (Max)**
- **Response Time (Avg)**

The table at the bottom shows all the remote hosts connected to by API Gateway instances in your domain. You can click a remote host in the table to drill down and view graphs for the selected remote host (for example, **REMOTE HOST - stockquote.com:80**). This also displays which services have been connecting to the remote host. You can click the **Back** button at the top right to return to the **ALL REMOTE HOSTS** view.

Clients

This **Clients** view displays metrics for all clients that have been authenticated for services in your domain. In the **Clients** view, click a panel in the **ALL CLIENTS** section at the top to display graph for the selected clients metric below. For example, the available metrics include the following:

- **Messages**
- **Successes**
- **Failures**
- **Exceptions**

The table at the bottom shows all clients that have been authenticated for services in your domain. You can click a client in the table to drill down and view graphs for the selected client (for example, **CLIENT - test.client**). This also displays which services have been connected to by the client. You can click the **Back** button at the top right to return to the **ALL CLIENTS** view.

Audit Trail

The **Audit Trail** view enables you to filter the audit log messages generated by API Gateway instances in your domain. You can filter log messages by clicking the **Search** button on the right in the toolbar. The **Query Editor** enables you to create a query to filter log messages by details such as time period, severity level, filter type, filter name, and log message text. When you have added your search criteria, click **Search** at the bottom to run the query. You can also save the query for later use.

When you click **Search**, the log messages that match the search criteria specified in the query are displayed in the table. For example, the details displayed in the table include the log message text, API Gateway name, alerts, and time. You can also double-click an item in the list for more details (for example, transaction ID, filter category, and filter name).

Reports

API Gateway Analytics uses message metrics stored in the centralized database by the API Gateway instances running in your domain. The API Gateway stores metrics for the virtualized services that it exposes, and for the services, and client and remote host connections that it protects. API Gateway Analytics can generate usage reports and charts based on this data, and display them in the **Reports** view.

You can create reports by selecting the appropriate button in the API Gateway Analytics toolbar, and clicking the **Schedule Report** button. For example, to generate reports on API Services, perform the following steps:

1. Click the **API Services** button in the API Gateway Analytics toolbar.
2. Click the **Schedule Report** button at the top left.
3. Complete the following fields:

Schedule	Select when to schedule the report. Defaults to `Now`.
Time	If you do not select to run the report now, select the time of day to run the report.
On	If you do not select to run the report now or daily, select the day of week on which to run the report.
From	Select the period of time for the report. Defaults to `Last 7 days`.
Email	Enter the email address to which to send the report.

4. Click **Create**.

Similarly, you can follow the same sequence of steps to generate reports in the **System**, **Remote Hosts**, and **Clients** views.

Custom Reporting

Oracle API Gateway Analytics enables you to configure custom reports to provide flexible reporting to suit various requirements. This includes viewing any available metric for each target report type, grouping metrics, and filtering metrics.

Custom Report Types
Custom reporting enables you to produce the following types of reports:

- API service usage
- API service usage for selected API service(s)
- Client usage
- Client usage for selected API service(s)

- Client(s) that used an API service
- API service(s) that a client used

Grouping and Filtering Metrics

Custom reporting enables you to group by or filter by any or all of the following:

- Client Name
- Service Name
- Instance Name
- Group Name

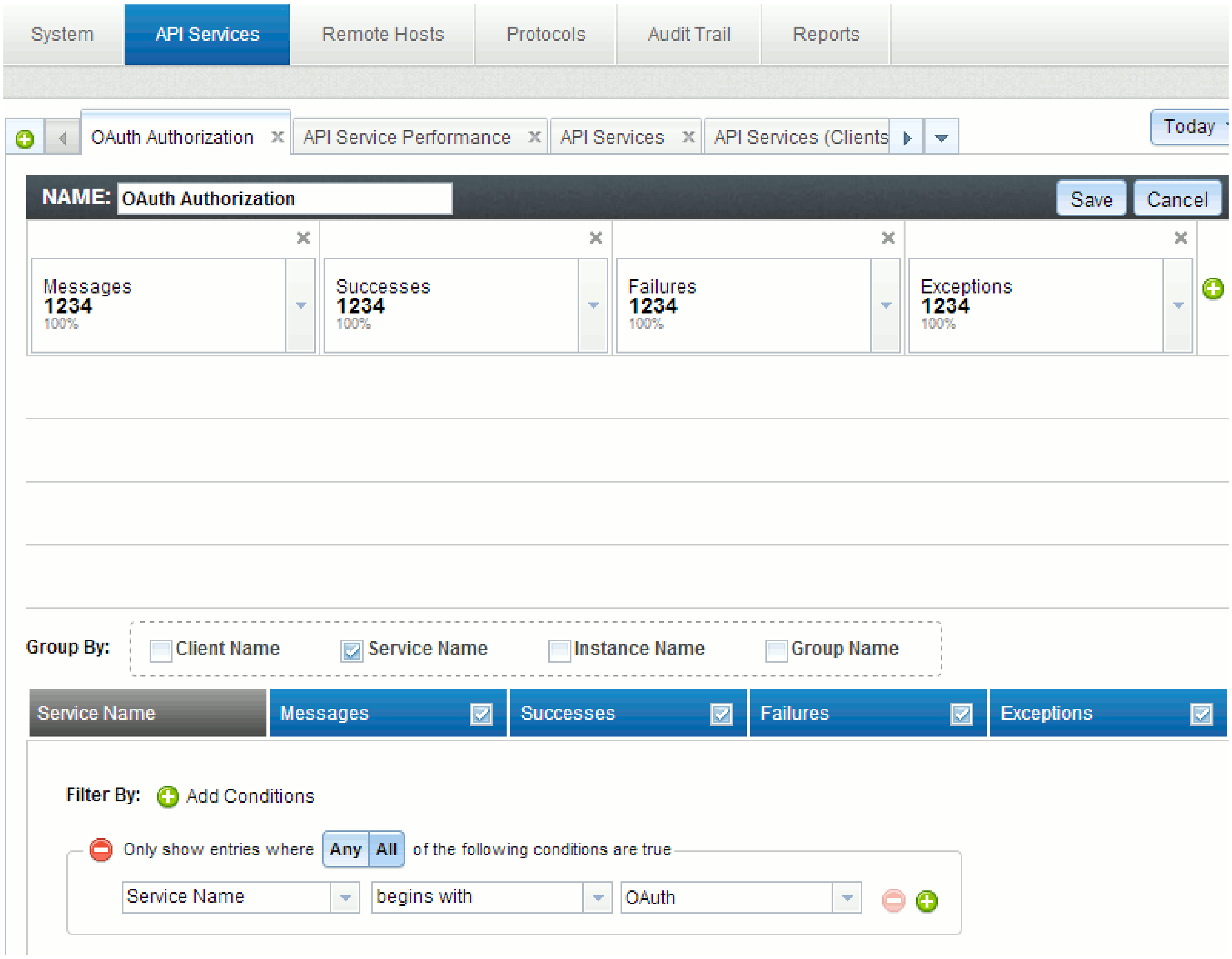
Note

The group-by mechanism only applies to the data table below the report chart. The chart remains the same.

These filtering and grouping mechanisms enable you to answer questions such as **Client(s) that used an API Service**, or **API service(s) that a client used**. For example, to show clients that used `WebService1`, you can create a custom report that groups by **Client Name** and filters where **Service Name** is `WebService1`.

Defining Custom Reports

Oracle API Gateway Analytics enables you to define custom report names, the metrics to be displayed and charted, and what to display on drill through. For example, the following screen shows a custom report named **OAuth Authorization**, which is grouped by **Service Name**, and filtered by service name beginning with `OAuth`.



NAME: OAuth Authorization

Group By: ☐ Client Name ☑ Service Name ☐ Instance Name ☐ Group Name

Service Name	Messages ☑	Successes ☑	Failures ☑	Exceptions ☑

Filter By: Add Conditions

Only show entries where Any All of the following conditions are true

Service Name | begins with | OAuth

Scheduled Reports

Overview

You can schedule reports to run on a regular basis, and have the results emailed to the user in PDF format. These reports include summary values at the top (for example, the number of requests, SLA breaches, alerts triggered, and unique clients in a specified week) followed by a table of services, and their aggregated usage data (for example, the number of requests on each service).

The report data is for the configured *current week of the report*, which is compared to the week before. You can set the configured *current week of the report* to be the actual current calendar week or any prior week (provided there is corresponding data in the database).

To configure scheduled reports, right-click the **Listeners** -> **Oracle API Gateway Analytics** node in the Policy Studio tree, and select **Database Archive**.

Database Configuration

Click the browse button the right, and select a pre-configured database connection in the dialog. This setting defaults to the `Default Database Connection`. To add a new database connection, right-click the **Database Connections** node, and select **Add DB connection**. You can also edit or delete existing nodes by right-clicking and selecting the appropriate option. Alternatively, you can add database connections under the **External Connections** node in the Policy Studio tree view. For more details on creating database connections, see the *Database Connection* topic.

Scheduled Reports Configuration

You can configure the following settings for scheduled reports:

Enable Report Generation:
Select whether to enable scheduled reports in PDF format. When selected, by default, this runs a scheduled weekly report on Monday morning at 0:01. For details on configuring a different time schedule, see the next setting. This setting is not selected by default.

When **Enable Report Generation** is enabled, you can configure the following settings on the **Report Generator Process** tab:

Connect to API Gateway Analytics as User:
Enter the username and password used to connect to the report generator process. Defaults to the values entered using the `configureserver` script (for example, `admin/changeme`).

Output:
Enter the directory used for the generated report files in the **Output Directory** field, or click **Choose** to browse to the directory. Defaults to the directory entered using the `configureoaganalytics` script (for example, `c:\temp\reports`). You can also select to **Do not delete report files after emailing**. This setting is not selected by default.

SMTP Configuration

When **Enable Report Generation** is enabled, you can configure the following settings on the **SMTP** tab. These settings default to those entered using the `configureoaganalytics` script.

Email Recipient (To):
Enter the recipient of the automatically generated email (for example, `user@mycorp.com`). Use a semicolon-separated list of email addresses to send reports to multiple recipients.

Email Sender (From):
The generated report emails appear *from* the sender email address specified here (for example, `no-`

reply@mycorp.com).

Note

Some mail servers do not allow relaying mail when the sender in the **From** field is not recognized by the server.

SMTP Server Settings:
Specify the following fields:

Outgoing Mail Server (SMTP)	Specify the SMTP server used to relay the report email (for example, `smtp.gmail.com`).
Port	Specify the SMTP server port to connect to. Defaults to port 25.
Connection Security	Select the connection security used to send the report email (`SSL`, `TLS`, or `NONE`). Defaults to `NONE`.

Log on Using:
If you are required to authenticate to the SMTP server, specify the following fields:

User Name:	Enter the user name for authentication.
Password:	Enter the password for the user name specified.

Real-Time Monitoring Settings

Overview

You can configure real-time monitoring settings at the API Gateway process level. For example, these enable you to specify monitoring of the message content. API Service Manager and API Gateway Analytics use this data to display graphical reports in their web-based interfaces. In addition, for API Gateway Analytics, you can also specify the database to which the Process writes metrics data.

Configuring Metrics Settings

To configure real-time monitoring settings, select the **Settings** node in the Policy Studio tree, and click the **Metrics** tab at the bottom. The following fields are available on the **Metrics** tab.

Enable real time monitoring:
This enables real-time monitoring globally for the API Gateway. This setting must be selected to monitor traffic processed by the API Gateway, and is enabled by default. To disable real-time monitoring, disable this setting.

Important

Enabling real-time monitoring has a negative impact on performance. If you wish to maximize performance, disable this setting.

Configuring Reports Settings

Store real time monitoring data for charts/reports:
If you wish real-time monitoring data to be written to a database, select this setting. This enables the following fields.

Use the following database:
Click the button on the right to select a database in which you wish to store the metrics data. If you have already configured the database connection, you can select it in the tree. To add a database connection, right-click the **Database Connections** tree node, and select **Add DB Connection**. Alternatively, you can add database connections under the **External Connections** node in the Policy Studio tree view. For more information on configuring database connections, see the *Database Connection* topic.

Note

API Gateway Analytics must be configured to read metrics data from the database that the API Gateway is configured to write the metrics data to. For more details, see the chapter on configuring the database in the *Oracle API Gateway Installation and Configuration Guide*.

Store specified message attribute values:
Select this setting to store the values of specified messages attributes in the database. Click the **Add** button, enter the appropriate message attribute name in the **Value** field, and click **OK**. Repeat to add multiple message attributes.

When the specified message attributes are present in a message running through the API Gateway, their values are then stored the database. For example, you could specify `my.example.attribute` to be stored in the database. Then for any message running through the API Gateway that has the `my.example.attribute` attribute set, the message ID, attribute name, and the value (for example, `My example message subject`) are stored in database. You could then use this information to view reports for messages with specific message subjects or IDs.

CPU, Memory, Disk polling interval:
Specifies the interval in seconds when polling CPU use, free memory, and disk settings used in system monitoring. Defaults to 3 seconds.

Configuring Traffic Monitoring

Overview

The **Traffic Monitor** settings enable you to configure the traffic monitoring available in the web-based API Service Manager tool. For example, you can configure where the data is stored and what message transaction details are recorded in the message traffic log.

To access the **Traffic Monitor** settings, click the **Settings** node in the Policy Studio tree, and select the **Traffic Monitor** tab at the bottom of the screen. To access traffic monitoring in API Service Manager, go to `http://localhost:8090`, and click the **Traffic** button in the toolbar. For more details, see the topic on *Monitoring Services*.

Important

Traffic monitoring may have a negative impact on performance. If you wish to maximize performance, you can disable traffic monitoring.

Configuration

You can configure the following **Traffic Monitor** settings:

Enable Traffic Monitor:
Select whether to enable the web-based Traffic Monitor tool. This is enabled by default.

Data Directory:
Enter the directory that stores the traffic monitoring data files and database (`data.sdb`). Enter the location relative to the directory in which the API Gateway is installed. Defaults to `conf/opsdb.d`. If this directory and/or the database do not exist when the API Gateway starts up, they are recreated automatically.

Data Persistence Settings:
You can configure the following data persistence settings:

Record inbound transactions	Select whether to record inbound message transactions received by the API Gateway. This is enabled by default.
Record outbound transactions	Select whether to record outbound message transactions sent from the API Gateway to remote hosts. This is enabled by default.
Record policy path	Select whether to record the policy path for the message transaction, which shows the filters that the message passes through. This is enabled by default.
Record trace	Select whether to record the trace output for the message transaction. This is enabled by default.

Note

These settings are global for all traffic passing through the API Gateway. You can override these persistence settings at the port level when configuring an HTTP or HTTPS interface. For more details, see *Configuring HTTP Services*.

Data Expiry Settings:
You can configure the following data expiry settings:

Max. database entries	Enter the maximum number of rows stored in the traffic monitoring database. Defaults to `1000000` rows.
Max. database size	Enter the maximum size of the traffic monitoring database file, and select the database size units from the list. Defaults to 1 GB.
Purge data older than	Enter the period of time to store data, and select the time units from the list. Data older than the specified time is purged from the database. Defaults to 5 days.

Purging the Reports Database

Overview

You can use the `dbpurger` command to connect to your reports database and to purge old data. This command also enables you to retain a specified amount of data, and to archive all data.

For details on configuring the connection to your reports database, see the *API Gateway Installation and Configuration Guide*.

Running the dbpurger Command

You can run the `dbpurger` command from the following directory:

Windows	`REPORTER_INSTALL\oaganalytics\Win32\bin`
Linux/UNIX	`REPORTER_INSTALL/oaganalytics/posix/bin`

Command Options
You can specify the following options to the `dbpurger` command:

Option	**Description**
`-h, --help`	Displays help message and exits.
`-p PASSPHRASE, --passphrase=PASSPHRASE`	Specifies the configuration passphrase (leave blank for zero length).
`--dbname=DBNAME`	Specifies the database name (mutually exclusive with `dburl`, `dbuser`, and `dbpass` options).
`--dburl=DBURL`	Specifies the database URL.
`--dbuser=DBUSER`	Specifies the database user.
`--dbpass=DBPASS`	Specifies the database passphrase.
`--archive`	Archive all data.
`--out=OUT`	Archive all data in the specified directory.
`--purge`	Purge data from the database. You must also specify the `--retain` option.
`--retain=RETAIN`	Specifies the amount of data to retain (for example, `30days`, `1month`, or `1year`). You must specify this option with the `--retain` option.

Example Commands

This section shows examples of running `dbpurger` in default interactive mode and of specifying command-line options.

Running dbpurger in Interactive Mode
The following example shows the output when running the `dbpurger` command in interactive mode. This example archives all data, retains three months of data, and purges older data from the database:

```
    >dbpurger
    Choosing: Default Database Connection
    Archive database (Y, N) [N]: y
    Archive path [./archive]:
    Purge an amount of data from the database (Y, N) [N]: y
    Amount of data to retain (e.g. 1year, 3months, 7days) [3months]:
    Wrote archive: .\archive\process_groups.xml
    Wrote archive: .\archive\processes.xml
    Wrote archive: .\archive\metric_types.xml
    Wrote archive: .\archive\audit_log_sign.xml
    Wrote archive: .\archive\time_window_types.xml
    Wrote archive: .\archive\audit_log_points.xml
    Wrote archive: .\archive\audit_message_payload.xml
    Wrote archive: .\archive\transaction_data.xml
    Wrote archive: .\archive\metric_groups.xml
    Wrote archive: .\archive\metric_group_types.xml
    Wrote archive: .\archive\metrics_alerts.xml
    Wrote archive: .\archive\metrics_data.xml
    Purging data older than: Wed Jun 27 15:26:00 BST 2012
    Purging table: audit_log_sign... deleted 0 rows
    Purging table: transaction_data... deleted 0 rows
    Purging table: audit_message_payload... deleted 7 rows
    Purging table: audit_log_points... deleted 16 rows
    Purging table: metrics_alerts... deleted 4 rows
    Purging table: metrics_data... deleted 703 rows
```

Specifying Command-Line Options to dbpurger

The following example shows the output when specifying options the `dbpurger` command. This example retains 30 days of data, and purges older data from the database:

```
dbpurger.bat --dburl=jdbc:mysql://127.0.0.1:3306/reports --dbuser=root
--dbpass=fred --purge --retain=30days
```

Configuring API Gateway Instances

Overview

This topic shows how to configure a running instance of the API Gateway. You can configure the options described in the following sections on the API Gateway instance in the Policy Studio tree.

Add Remote Host

Remote Host settings configure the way in which the API Gateway routes to another host machine. For example, if a destination server may not fully support HTTP 1.1, you can configure **Remote Host** settings for the server to optimize the way in which the API Gateway sends messages to it. Similarly, if the server requires an exceptionally long timeout, you can configure this in the **Remote Host** settings. For more details, see the *Remote Host Settings* topic.

Add HTTP Services

You can add a container for HTTP-related services, including HTTP and HTTPS Interfaces, Directory Scanners, Static Content Providers, Servlet Applications, and Packet Sniffers.

HTTP Services act as a container for all HTTP-related interfaces to the API Gateway's core messaging pipeline. You can configure HTTP and HTTPS interfaces to accept plain HTTP and SSL messages respectively. A Relative Path interface is available to map requests received on a particular URI or path to a specific policy. The Static Content Provider interface can retrieve static files from a specified directory, while the Servlet Application enables you to deploy servlets under the service. Finally, the Packet Sniffer interface can read packets directly of the network interface, assemble them into HTTP messages, and dispatch them to a particular policy. The *Configuring HTTP Services* topic explains how to configure the available HTTP Interfaces.

Add SMTP Services

Simple Mail Transfer Protocol (SMTP) support enables the API Gateway to receive email and to act as a mail relay. The API Gateway can accept email messages using the SMTP protocol, and forward them to a mail server. You can also configure optional policies for specific SMTP commands (for example, `HELO/EHLO` and `AUTH`). The *Configuring SMTP Services* topic explains how to configure SMTP services, interfaces, and handler policies.

Add File Transfer Services

You can configure the API Gateway to listen for remote clients that connect to it as a file server. This enables the API Gateway to apply configured policies on transferred files (for example, for schema validation, threat detection or prevention, routing, and so on). The API Gateway supports File Transfer Protocol (FTP), FTP over SSL (FTPS), and Secure Shell FTP (SFTP). The *File Transfer Service* topic explains how to configure the API Gateway as a file transfer service.

Add Policy Execution Scheduler

Policy Execution Scheduling enables you to schedule the execution of any policy on a specified date and time in a recurring manner. The API Gateway provides a pre-configured library of schedules to select from. You can also add your own schedules to the library. The *Policy Execution Scheduling* topic explains how to add a policy execution schedule, and how to add schedules.

Messaging System

You can configure the API Gateway to read JMS messages from a JMS queue or topic, run them through a policy, and then route onwards to a Web Service or JMS queue or topic.

The API Gateway can consume a JMS queue or topic as a means of passing XML messages to its core message processing pipeline. When the message has entered the pipeline, it can be validated against all authentication, authorization, and content-based message filters. Having passed all configured message filters, it can be routed to a destination Web Service over HTTP, or it can be dropped back on to a JMS queue or topic using the **Messaging System** Connection filter. For more details, see the *Messaging System* topic.

FTP Poller

The **FTP Poller** enables you to query and retrieve files by polling a remote file server. When files are retrieved, they can be passed into the API Gateway core message pipeline for processing. For example, this is useful in cases where an external application drops files on to a remote file server, which can then be validated, modified, or routed on over HTTP or JMS by the API Gateway. For more details, see the *FTP Poller* topic.

Directory Scanner

The **Directory Scanner** reads XML files from a specified directory and dispatches them to a selected policy. This enables you to search a local directory for XML files, which can then be fed into a security policy for validation. Typically, XML files are FTP-ed or saved to the file system by another application. The API Gateway can then pick these files up, run the full array of authentication, authorization, and content-based filters on the messages, and then route them over HTTP or JMS to a back-end system. For more details, see the *Directory Scanner* topic.

POP Client

The **POP Client** enables you to poll a POP mail server to read email messages from it, and pass them into a policy for processing. For more details, see the *POP Client* topic.

TIBCO

You can configure a TIBCO Rendezvous® Listener or a TIBCO Enterprise Messaging Service™ Consumer. For more details, see the following topics:

- *TIBCO Rendezvous Listener*
- *TIBCO Enterprise Messaging Service Consumer*

API Gateway Settings

You can configure per-instance global configuration settings by clicking the **Settings** node in the Policy Studio tree. For more details on configuring API Gateway instance settings, see the *API Gateway Settings* topic.

API Gateway Logging

You can configure a API Gateway instance to log messages to a database, file system, GUI Console, log files, or UNIX syslog. A Log Viewer for examining log entries is also available. For more details, see the topic on *Audit Log Settings* .

Cryptographic Acceleration

The API Gateway can leverage the OpenSSL Engine API to offload complex cryptographic operations (for example, RSA and DSA) to a hardware-based cryptographic accelerator, and to act as an extra layer of security when storing private keys on a Hardware Security Module (HSM).

The API Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation, and SSL tunneling. OpenSSL exposes an *Engine API*, which enables you to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. OpenSSL can, when configured appropriately, call the engine's implementation of these operations instead of its own. For more information on configuring the API Gateway to use an OpenSSL engine, see the *Cryptographic Acceleration* topic.

Configuring HTTP Services

Overview

The API Gateway uses *HTTP Services* to handle traffic from various HTTP-based sources. The following list summarizes the list of available HTTP Services:

HTTP Interfaces
HTTP Interfaces are used in the API Gateway to define the ports and IP addresses on which the API Gateway instance listens for incoming requests. You can also add *HTTPS Interfaces* and select the SSL certificate to use to authenticate to clients, and the certificates that are considered trusted for establishing SSL connections with clients.

Relative Path
While HTTP and HTTPS Interfaces open sockets and listen for traffic, you can configure *Relative Paths* so that when a request is received on that path, the API Gateway instance can map it to a specific policy, or chain of policies.

Static Content Provider
You can use a *Static Content Provider* to serve static HTTP content from a particular directory. In this case, the API Gateway instance is effectively acting as a Web Server.

Servlet Applications
The API Gateway instance can act as a servlet container, which enables you to drop servlets into the HTTP Services configuration. This should only be used by developers with very specific requirements and under strict advice from the Oracle support team.

Packet Sniffer
Finally, you can add a *Packet Sniffer* to intercept network packets from the client, assemble them into complete HTTP messages, and log these messages to an audit trail. Because the Packet Sniffer operates at the network layer (unlike an HTTP-based traffic monitor at the application layer), the packets are intercepted transparently to the client. This means that the Packet Sniffer is a *passive service*, which is typically used for management and monitoring instead of general policy enforcement. For more details, see Packet Sniffers.

HTTP Services Groups

An *HTTP Services Group* is a container around one or more HTTP Services. Usually, an HTTP Services Group is configured for a particular type of HTTP Service. For example, it is typical to have an `HTTP Interfaces` Group that contains the configured HTTP Interfaces, and another `Static Providers` Group to manage Static Content Providers. While organizing HTTP Services by type eases the task of managing Services, the API Gateway is flexible enough to enable administrators to organize Services into Groups according to whatever scheme best suits their requirements.

This section describes a scenario where HTTP Services Groups can prove useful. It first describes an HTTP Service Group that handles HTTP traffic, and then shows how you can use a second SSL Service Group to process SSL traffic on a separate channel.

HTTP Interfaces and Relative Paths
HTTP Services Groups should consist of at least one HTTP Interface together with at least one Relative Path. The HTTP Interface determines which TCP port the API Gateway instance listens on, while you can use the Relative Path to map a request received on a particular path (request URI) to specific policies. You can add several HTTP Interfaces to the Groups, in which case requests received on any one of the opened ports are processed in the same manner. For example, `http://[HOST]:8080/test` and `http://[HOST]:8081/test` requests can both be processed by the same policy (mapped to the `/test` Relative Path).

Similarly, you can add multiple Relative Paths to this HTTP Services Group, where each Path is bound to a specific policy or chain of policies. For example, if a request is made to `http://[HOST]:8080/a`, it is processed by Policy A; whereas if a request is made to `http://[HOST]:8080/b`, it is handled by Policy B. As a side-effect of this configuration, requests made to the other Interface are also processed by the same policy, meaning that a request made to `ht-`

tp://[HOST]:8081/b is also handled by Policy B.

Effectively, this means that the Relative Paths configured under the HTTP Services Group are bound to all HTTP Interfaces configured for that Group. If you have two Interfaces listening on ports 8080 and 8081, requests to http://[HOST]:8080/a and http://[HOST]:8081/a are handled identically by the API Gateway instance.

Example HTTP Service Group

What happens if you want to distinguish between receiving requests on the two different ports? For example, you want requests to http://[HOST]:443/a to be processed by an **SSL Validation** policy, while requests for http://[HOST]:8080/a to be handled by a standard **Schema Validation** policy.

The addition of a new HTTP Services Group can resolve this issue. The new SSL HTTP Services Group opens a single HTTPS Interface that listens on port 443, and is configured with a Relative Path of /a to handle requests on this path. The configuration is summarized in the following table:

Services Group	***HTTP Port***	***Relative Path***	***Policy***
HTTP Services Group	8080	/a	Schema Validation Policy
SSL HTTP Services Group	443	/a	SSL Validation Policy

With this configuration, when you receive a request on http://[HOST]:8080/a, it is handled by the **Schema Validation Policy**. But when you get a request to the SSL port on http://[HOST]:443/a it is processed by the **SSL Validation Policy**. Using HTTP Services Groups in this way, you can configure the API Gateway instance to dispatch requests received on the same path (for example, /a) to different policies depending on the port on which the request was accepted.

Default HTTP Service Groups

By default, the API Gateway ships with pre-configured HTTP Services Groups (for example, Default Services and Management Services). By default, Management Services are not displayed in the Policy Studio, but can be displayed using the **Preferences** menu. The Default Services Group contains some general purpose default policies for use with an out-of-the-box installation of the API Gateway. In addition to the pre-configured Service Groups, you can add new HTTP Services Groups when you wish to dispatch requests to different policies, based on the port on which the requests are received. For more details on the default Management Services Group, see the section called “Management Services”.

Adding an HTTP Service Group

To add a Service Group, right-click the API Gateway instance, and select the **Add HTTP Services** menu option. Enter a name for the group in the **HTTP Services** dialog. If you want to monitor traffic processed by this Service Group using API Gateway Analytics, select the **Include in real time monitoring** checkbox. For details on configuring the API Gateway to store real-time monitoring data, which can be used to produce graphical reports in a web-based interface, see the *Real-Time Monitoring Settings*.

When a Service Group has been created, the following types of HTTP Services can be associated with it.

HTTP and HTTPS Interfaces

An HTTP Interface defines the address and port that the API Gateway instance listens on. There are two types of Interface: HTTP and HTTPS. The HTTP interface handles standard non-authenticated HTTP requests, while the HTTPS interface can accept mutually authenticated SSL connections.

To create an HTTP Interface, under the HTTP Service Group (for example, Default Services) in the Policy Studio tree, right-click **Ports**, and select **Add HTTP** or **Add HTTPS**.

Configuring Network Settings

The following fields on the **Network** tab are common to both the **HTTP Interface** and **HTTPS Interface** dialogs, and must be configured for both types of interface:

Port:
The port number that the API Gateway instance listens on for incoming HTTP requests.

Address:
The IP address or host of the network interface on which the API Gateway instance listens. For example, you can configure the instance to listen on port 80 on the external IP address of a machine, while having a Web server running on the same port but on the internal IP address of the same machine. By entering *****, the instance listens on all interfaces available on the machine hosting the API Gateway.

Protocol:
Select the Internet Protocol version that this Interface uses. You can select `IPv4`, `IPv6`, or both of these protocol versions. Defaults to `IPv4`.

Trace level:
The level of trace output. The possible values in order of least verbose to most verbose output are:

- `FATAL`
- `ERROR`
- `INFO`
- `DEBUG`
- `DATA`

The default trace level is read from the system settings.

Enable interface:
Deselect this setting to disable this HTTP interface. This setting is enabled by default.

Configuring Traffic Monitor Settings

The fields on the **Traffic Monitor** tab are common to both the **HTTP Interface** and **HTTPS Interface** dialogs. To override the system-level settings at the HTTP or HTTPS interface level, select **Override settings for this port**, and configure the following options:

Record inbound transactions	Select whether to record inbound message transactions received by the API Gateway. This is enabled by default.
Record outbound transactions	Select whether to record outbound message transactions sent from the API Gateway to remote hosts. This is enabled by default.
Record policy path	Select whether to record the policy path for the message transaction, which shows the filters that the message passes through. This is enabled by default.
Record trace	Select whether to record the trace output for the message transaction. This is enabled by default.

For more details, see *Configuring Traffic Monitoring*.

Configuring Advanced Settings

The following fields on the **Advanced** tab are common to both the **HTTP Interface** and **HTTPS Interface** dialogs, and must be configured for both types of interface:

Backlog:

When the API Gateway instance is busy handling concurrent requests, the operating system can accept additional incoming connections. In such cases, a backlog of connections can build up while the operating system waits for the instance to finish handling current requests.

The specified **Backlog** value is the maximum number of connections that the API Gateway instance allows the operating system to accept and queue up until the instance is ready to read them. There is a trade off: the larger the backlog, the larger the memory usage; the smaller the backlog, the greater the potential for dropped connections.

Idle Timeout:
The API Gateway supports the use of HTTP 1.1 persistent connections. The **Idle Timeout** is the time that the API Gateway instance waits after sending a response over a persistent connection before it closes the connection. Defaults to 60000 milliseconds (60 seconds).

Typically, a client tells the instance that it wants to use a persistent connection. The instance acknowledges this instruction and decides to keep the connection open for a certain amount of time after sending the response to the client. If the client does not reuse the connection by sending up another request within the **Idle Timeout** period, the instance closes the connection.

Active Timeout:
When the API Gateway instance receives a large HTTP request, it reads the request off the network as it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, the instance closes the connection. Defaults to 60000 milliseconds (60 seconds).

This guards against a client closing the connection while in the middle of sending data. Imagine the client's network connection is pulled out of the machine while in the middle of sending data to the instance. When the instance has read all the available data off the network, it waits the **Active Timeout** period of time before closing the connection.

Maximum Memory per Request:
The maximum amount of memory that the API Gateway instance allocates to each request, not including the HTTP body.

Input Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can accept from peers. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured in the **Default Settings** are used. You can override this setting at the HTTP interface level and in the **Remote Host Settings**. For more details, see the topic on *Compressed Content Encoding*.

Output Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can apply to outgoing messages. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured in the **Default Settings** are used. You can override this setting at the HTTP interface level and in the **Remote Host Settings**. For more details, see the topic on *Compressed Content Encoding*.

Transparent Proxy - allow bind to foreign address:
Enables the use of the API Gateway as a *transparent proxy* on Linux systems with the `TPROXY` kernel option set. When selected, the value in the **Address** field can specify any IP address, and incoming traffic for the configured address/port combinations is handled by the API Gateway. For more details and an example, see *Configuring a Transparent Proxy*.

Configuring Conditions for an HTTP Interface
You can configure the API Gateway to bring down an active HTTP Interface if certain *conditions* fail to hold. For example, the HTTP Interface can be brought down if a Remote Host is not available or if a physical network interface on the machine on which the API Gateway is running loses connectivity to the network.

For more information on configuring *conditions* for HTTP Interfaces, see the *Configuring Conditions for HTTP Interfaces* help page.

HTTPS Interfaces Only

You must complete the same fields for an HTTPS Interface on the **Network** tab as for an HTTP Interface, with the addition of the following setting:

X.509 Certificate:
Click the **X.509 Certificate** button to select the certificate that the API Gateway uses to authenticate itself to clients during the SSL handshake. The list of certificates currently stored in the **Certificate Store** is displayed. Select a single certificate from this list.

Configuring Mutual Authentication Settings
You can configure clients to authenticate to the API Gateway on the **Mutual Authentication** tab. The following options are available:

- **Ignore Client Certificates**
  The API Gateway ignores client certificates if they are presented during the SSL handshake.
- **Accept Client Certificates**
  Client certificates are accepted when presented to the API Gateway, but clients that do not present certificates are not rejected.
- **Require Client Certificates**
  The API Gateway only accepts connections from clients that present a certificate during the SSL handshake.

Client certificates are typically issued by a Certificate Authority (CA). In most cases, the CA includes a copy of its certificate in the client certificate so that consumers of the certificate can decide whether or not to trust the client based on the issuer of the certificate.

It is also possible that a *chain* of CAs were involved in issuing the client certificate. For example, a top-level organization-wide CA (for example, Company CA) may have issued department-wide CAs (for example, Sales CA, QA CA, and so on), and each department CA would then be responsible for issuing all department members with a client certificate. In such cases, the client certificate may contain a chain of one or more CA certificates.

Maximum depth of client certificate chain:
You can use this field to configure how many CA certificates in a chain of one or more are trusted when validating the client certificate. By default, only one issuing CA certificate is used, and this certificate must be checked in the list of trusted root certificates. If more than one certificate is used, only the top-level CA must be considered trusted, while the intermediate CA certificates are not.

Root Certificate Authorities trusted for mutual authentication:
Select the root CA certificates that the API Gateway considers trusted when authenticating clients during the SSL handshake. Only certificates signed by the CAs selected here are accepted.

Configuring Advanced SSL Settings
You can complete the following settings on the **Advanced (SSL)** tab:

Check that the SSL certificate's Subject CN resolves to network address:
When this setting is selected the API Gateway attempts to resolve the SSL certificate's `Subject` Common Name (CN) to the network address configuring the SSL interface. If the `Subject` CN cannot be resolved to the network address, a warning is output in the error traces. This setting is selected by default. You can deselect this setting to disable checking the certificate's `Subject` CN.

SSL Server Name Identifier (SNI):
You can specify the hostnames that are requested by clients in the **SSL Server Name Identifier (SNI)** table. SNI is an optional TLS feature whereby the client indicates to the server the hostname used to resolve the server's address. This enables a server to present different certificates for clients to ensure the correct site is contacted.

For example, the server IP address is `192.168.0.1`. The DNS is consulted by clients to resolve a hostname to an address, and the server address is contacted using TCP/IP. If both `www.acme.com` and `www.anvils.com` resolve to `192.168.0.1`, without SNI, the server does not know which hostname the client uses to resolve the address, because it is not party to the client's DNS name resolution. The server may be able to certify itself as either service, but when the connection is established, it does not know which hostname the client connects to.

With SNI, the client provides the name of the host (for example, `www.anvils.com`) in the initial SSL exchange, before the server presents its certificate in its distinguished name (for example, `CN=www.anvils.com`). This enables the server to certify itself correctly as providing a service for the client's requested hostname.

To specify an SNI, perform the following steps:

1. Click the **Add** button to configure a server hostname in the **SSL Server Name Identifier (SNI)** dialog.
2. Specify the server hostname in the **Client requests server name** field.
3. Click the **Server assumes identity** button to import a Certificate Authority certificate into the Certificate Store.
4. **Click OK**.

Ciphers:
You can specify the ciphers that the server supports in the **Ciphers** field. The server selects the highest strength cipher (that is also supported by the client) from this list as part of the SSL handshake. For more information on the syntax of this setting, see the OpenSSL documentation [http://www.openssl.org/docs/apps/ciphers.html].

SSL session cache size:
Specifies the number of idle SSL sessions that can be kept in memory. Defaults to `32`. If there are more than 32 simultaneous SSL sessions, this does not prevent another SSL connection from being established, but means that no more SSL sessions are cached. A cache size of `0` means no cache, and no outbound SSL connections are cached.

Tip

You can use this setting to improve performance because it caches the slowest part of establishing the SSL connection. A new connection does not need to go through full authentication if it finds its target in the cache.

At `DEBUG` level or higher, the API Gateway outputs trace when an entry goes into the cache, for example:

```
DEBUG   09:09:12:953 [0d50] cache SSL session 11AA3894 to support.acme.com:443
```

If the cache is full, the output is as follows:

```
DEBUG   09:09:12:953 [0d50] enough cached SSL sessions 11AA3894 to
support.acme.com:443 already
```

Ephemeral DH key parameters:
The Diffie Hellman (DH) key agreement algorithm is used to negotiate a shared secret between two SSL peers. This enables two parties without prior knowledge of each other to jointly establish a shared secret key over an insecure communication channel. The **Ephemeral DH key parameters** setting specifies the parameters used to generate the DH keys.

When DH key parameters are not specified, the SSL client uses the public RSA key in the server's certificate to encrypt data sent to the SSL server, and establish a shared secret with the server. However, if the RSA key is ever discovered, any previously recorded encrypted conversations can be decrypted. DH key agreement offers Perfect Forward Secrecy (PFS) because there is no such key to be compromised.

There are two options when setting the DH parameters: you can enter a number (for example, `512`), and the server automatically generates DH parameters with a prime number of the correct size. Alternatively, you can paste the Base64 encoding of an existing serialized DH parameters file. You can use standard DH parameters based on known good prime numbers. OpenSSL ships with the `dh512.pem` and `dh1024.pem` files. For example, you can set the DH parameters to the following Base64-encoded string in `pdh512.pem`:

```
-----BEGIN DH PARAMETERS-----
MEYCQQD1Kv884bEpQBgRjXyEpwpy1obEAxnIByl6ypUM2Zafq9AKUJsCRtMIPWakXUGfnHy9iUsiGSa6q6Jew1X
pKgVfAgEC
-----END DH PARAMETERS-----
```

The DH parameters setting is required if the server is using a DSA-keyed certificate, but also has an effect when using RSA-based certificates. DH (or similar) key agreement is required for DSA-based certificates because DSA keys cannot be trivially used to encrypt data like RSA keys can.

SSL Protocol Options:
You can configure the following SSL protocol options:

Option	*Description*
Use EDH key once only	Creates a new key from the DH parameters for every SSL session. This is not strictly necessary if you are sure about the quality of the prime number in the DH key parameters. When using well-known DH parameters like the example above, you can safely leave this option unselected. However, given a bad prime number in the parameters, gathering enough key exchanges from a single DH key can allow an eavesdropper to work out the DH key used. Selecting this option slows down the SSL session establishment and has a negative impact on performance.
Do not use the SSL v1 protocol	Specifies not to use SSL v1 to avoid any weaknesses in this protocol. This option is not selected by default.
Do not use the SSL v2 protocol	Specifies not to use SSL v2 to avoid any weaknesses in this protocol. This option is not selected by default.
Do not use the SSL v3 protocol	Specifies not to use SSL v3 to avoid any weaknesses in this protocol. This option is not selected by default.
Do not use the TLS v1 protocol	Specifies not to use TLS v1 to avoid any weaknesses in this protocol. This option is not selected by default.
Prefer local cipher preferences over client's proposal	When choosing a cipher during the SSL/TLS handshake, the client's preferences are selected by default from the list of ciphers supported by the client and the server. When this option is selected, the server's preferences are used instead. This option is not selected by default. For more details on ciphers, see the OpenSSL documentation [http://www.openssl.org/docs/apps/ciphers.html]

Relative Paths

A Relative Path binds policies to a specific relative path location (for example `/healthcheck`). When the API Gateway receives a request on this relative path, it invokes the specified policies. For details on how to configure policies, see Chapter 4, Governance.

To configure a Relative Path for a given HTTP Service Group, under the Service Group in the tree view, right-click **Paths**, and select **Add Relative Path**. Alternatively, you can also click the **Add Relative Path** button at the bottom of the policy canvas beside the **Context** drop-down list. Complete the following settings on the **Configure Relative Path** dialog:

Enable this path resolver:
You can specify whether to enable listening on the specified path. This is enabled by default.

Policies:
On the **Policies** tab, specify the relative path and the policies that are called. The API Gateway invokes the selected

policies when it receives a request on the specified path. You can specify a single policy or a chain of policies. Policies are called in the order displayed on this tab. Complete the following fields:

When a request arrives that matches the path:	Enter a relative path (for example, `/test`) for the selected HTTP Service Group. Requests received on this relative path are processed by the policies selected on this tab.
Global Request Policy	If a global request policy is configured, when you select this setting, the global request policy is called first in the policy chain. For more details, see *Configuring Global Policies*.
Path Specific Policy	To configure a path-specific policy, select the checkbox, and click the browse button to select a policy from the dialog. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically. The **Path Specific Policy** field is auto-populated with the currently selected policy when the dialog is launched using the **Add Relative Path** button at the bottom of the policy canvas.
Global Response Policy	If a global response policy is configured, when you select this setting, the global response policy is called last in the policy chain. For more details, see *Configuring Global Policies*.

When you select multiple policies to form a policy chain, the behavior is the same as for a policy shortcut chain filter. Policies are only evaluated when selected, and when the policy can be reached. If any reachable policy fails, the chain fails, and no more policies are evaluated.

Audit Settings:
The **Audit Settings** tab enables you to configure the logging level for all filters executed on the relative path, and to configure when message payloads are logged.

You can configure the following **Logging Level** settings on all filters executed on the specified relative path:

Logging Level	*Description*
Fatal	Logs Fatal log points that occur on all filters executed.
Failure	Logs Failure log points that occur on all filters executed. This is the default logging level.
Success	Logs Success log points that occur on all filters executed.

For more details on logging levels, and on how to configure logging for a specific filter, see the *Log Level and Message* topic.

You can configure the following **Payload Logging** settings on the specified relative path:

Payload Logging	*Description*
On receive request from client	Log the message payload when a request arrives from the client.
On send response to client	Log the message payload before the response is sent back to the client.

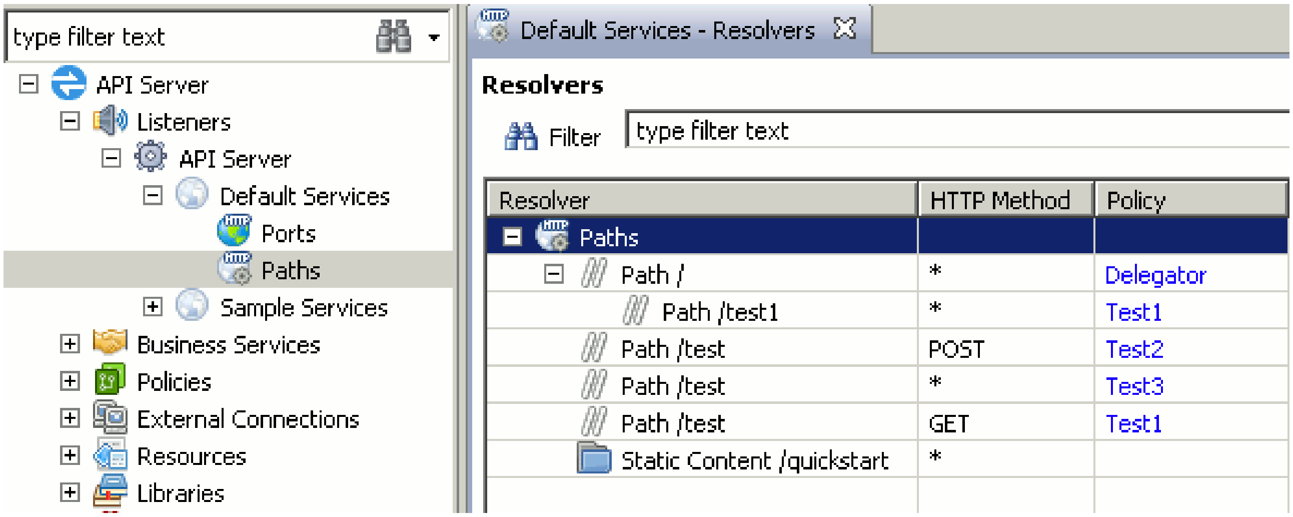
On send request to remote server	Log the message payload before the request is sent using any **Connection** or **Connect to URL** filters deployed in any policies executed.
On receive response from remote server	Log the message payload when the response is received using any **Connection** or **Connect to URL** filters deployed in any policies executed.

For details on how to log message payloads at any point in a specific policy, see the *Log Message Payload* topic.

Select the **Include in server access log records** setting if you wish to add this relative path to the API Gateway Access Log. This enables the Access Log at the service level. This setting is not selected by default. For more details, see the topic on *Access Log Settings*.

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which means that all HTTP Methods are accepted. You can override the default behavior, and select an appropriate HTTP method for the relative path from the list.

You can configure multiple HTTP methods on paths of the same name. This enables you to call different policies for different HTTP methods, as shown in the following example:



Resolver	HTTP Method	Policy
Paths		
Path /	*	Delegator
Path /test1	*	Test1
Path /test	POST	Test2
Path /test	*	Test3
Path /test	GET	Test1
Static Content /quickstart	*	

In this example, the `/test` path is configured three times, each using a different HTTP Method as follows:

- If a `GET` request is sent to the API Gateway on the `/test` path, the `Test1` policy is executed.
- If a `POST` request is sent, the `Test2` policy is executed.
- If any other type of request is sent (for example, `DELETE`, `PUT`, and so on), the `Test3` policy is executed.

Web Service Resolvers

A Web Service Resolver is used to identify messages destined for a Web Service, and to map them to the **Service Handler** (**Web Service Filter**) for that Web Service. When you import a WSDL file, a new Web Service Resolver node is created for each imported Web Service in the **Services** tree view. You can edit the Web Service Resolver settings by right-clicking this tree node, and selecting **Edit**.

The following settings are available in the **Web Service Resolver** dialog:

Enable this web service resolver:
Specify whether to enable this Web Service resolver. This is enabled by default.

Name:
You can edit the name of the Web Service Resolver.

Web Service:
Click the browse button to select a Web Service to resolve to. Defaults to the Web Service imported into the Web Services Repository when this resolver was created.

Policy:
On the **Policy** tab, select the path and the policies to use for the Web Service. You can specify a single policy or a chain of policies. Policies are called in the order displayed on this tab. The global request policy, the policy automatically generated when the WSDL file is imported, and the global response policy are all selected in a chain by default. Complete the following fields:

matches the paths in the WSDL:	Select this option if you want the Web Service Resolver to use the paths specified in the WSDL file. This is the default.
matches this path:	Select this option if you want to specify a different path from the WSDL file, and enter the path.
Global Request Policy	If a global request policy is configured, when you select this setting, the global request policy is called first in the policy chain. For more details, see *Configuring Global Policies*.
Path Specific Policy	To configure a path-specific policy, select the checkbox, and click the browse button to select a policy from the dialog.
Global Response Policy	If a global response policy is configured, when you select this setting, the global response policy is called last in the policy chain. For more details, see *Configuring Global Policies*.

Policies are only evaluated when selected, and when the policy can be reached. If any selected policy fails, the chain fails, and no more policies are evaluated.

Audit:
The **Audit** tab enables you to configure the logging level for all filters executed on the Web Service, and to configure when message payloads are logged. The default logging level for all filters on the Web Service is `Failure`. These logging settings are the same as those already described for the **Audit** tab used for the Relative Path. For more details, see Relative Paths.

Editing Service Handler Options
You also edit options for the **Service Handler** for the Web Service. Right-click the Web Service Resolver node, and select **Quick-Edit Policy** to display a dialog that enables you to configure the following options:

Validation	If you wish to use a dedicated validation policy for all messages sent to the Web Service, select this checkbox, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom validation policy used by multiple Web Services.

Routing	If you wish to use a dedicated routing policy to send all messages on to the Web Service, select this checkbox, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom routing policy used by multiple Web Services.
WSDL Access Options	Select whether to make the WSDL for this Web Service available to clients. The **Allow the API Gateway to publish WSDL to clients** checkbox is selected by default. The published WSDL represents a virtualized view of the Web Service. Clients can retrieve the WSDL from the API Gateway, generate SOAP requests, and send them to the API Gateway, which routes them on to the Web Service.

These options enable you to configure the underlying auto-generated Service Handler (**Web Service Filter**) without navigating to it in the **Policies** tree. These are the most commonly modified **Web Service Filter** options after importing a WSDL file. Changes made in this dialog are visible in the underlying **Web Service Filter**. For more details, see the *Web Service Filter* topic.

Static Content Provider

A *Static Content Provider* can be used in conjunction with an HTTP Interface to serve static content from a specified directory. A relative path is associated with each Static Content Provider so that requests received on this path are dispatched directly to the Provider and are not mapped to a Policy in the usual manner.

For example, you can configure a Static Content Provider to serve content from the `c:\docs` folder (on a Windows system) when it receives requests on the relative path **/docs**.

To add a Static Content Provider, right-click the Service Group under the API Gateway instance, and select the **Static Content Provider** menu option. Complete the following fields on the **General** tab:

Relative Path:
Enter the path that you want to receive requests for static content on.

Content Directory:
Enter or browse to the location of the directory that you want to serve static content from.

Index File:
Enter the name of the file that you want to use as the index file for content retrieved from the directory specified in the field above. This file is retrieved by default if no resource is explicitly specified in the request URI. For example, if the client requests `http://[HOST]:8080/docs` (with only a relative path specified as opposed to a specific resource), the file specified here will be retrieved. This file must exist in the directory specified in the previous field.

Allow Directory Listings:
If this option is selected, the Static Content Provider returns full directory listings for requests specifying a relative path only. For example, if this option is selected, and if a request is received for `http://[HOST]:8080/docs/samples`, the list of directories under this directory is returned, assuming that this directory exists on the file system. This option can be turned off to prevent attacks where a hacker can send up different request URIs in the hope that the server returns some information about the directory structure of the server.

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which means that all HTTP Methods are accepted. You can override the default behavior, and select an appropriate HTTP method for this resolver from the list.

Servlet Applications

Developers may wish to write their own Java servlets and deploy them under the API Gateway to serve HTTP traffic. Conversely, they may wish to remove some of the default servlets from the out-of-the-box configuration (for example, to remove the ability to view logging remotely). This pairing down of unwanted functionality may be required to further lock down the machine on which the API Gateway is running.

Note

Adding and removing Servlet Applications should be performed only by developers with very specific requirements and under strict guidance from the Oracle Support team. These instructions simply outline how to configure the fields on the dialogs used to set up Servlet Applications. For more detailed instructions, please contact the Oracle Support Team (see *Oracle Contact Details*).

There are a few default Servlet Applications available under the **Management Services** group. By default, this services group is not displayed, but can be displayed using the **Preferences** dialog in the Policy Studio. For example, the `/configuration/` Servlet Application updates configuration information for the API Gateway.

Warning

Deleting any of these Servlet Applications may prevent the API Gateway from functioning correctly. Default Servlet Applications should only be deleted under strict supervision of the Oracle Support team.

To add a new Servlet Application, right-click the Services Group that you wish to add the servlet to, and select **Add Servlet Application**. Configure the following fields on the **Servlet Application** dialog:

Relative Path:
Enter the servlet *context* in this field. You can add multiple servlets under this context, where each servlet is mapped to a unique URI.

Session Timeout:
Enter the timeout in seconds after which an inactive session is closed. Click **OK**.

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which means that all HTTP Methods are accepted. You can override the default behavior, and select an appropriate HTTP method for this resolver from the list.

The new Servlet Application now appears in the Policy Studio tree view. To add a new servlet, right-click the new Servlet Application, and select **Add Servlet**. Configure the following fields on the **Servlet** dialog:

URI:
The path entered here maps incoming requests on a particular request URI to the Java servlet class entered in the field below. This path must be unique across all Servlets that are added under this Servlet Application (servlet context).

Class:
Enter the fully qualified class name of the servlet class. This class can be added to the server runtime by adding the JAR, class file, or package hierarchy to the `[VINSTDIR]/ext/lib` folder, where `VINSTDIR` points to the root of your API Gateway installation.

Read Timeout:
Specify the time in seconds that the servlet should wait before closing an idle connection.

Servlet Properties:
You can configure properties for each servlet by clicking the **Add** button, and entering a name and value in the fields provided on the **Properties** dialog.

Management Services

The **Management Services** group exposes a number of services used by the Admin Node Manager and Oracle API Gateway Analytics for remote configuration and monitoring. By default, this group is not displayed in the Policy Studio tree view. However, you can view it by selecting the **Window** -> **Preferences** main menu option, and selecting **Show Management Services**. Click the **Apply** button to view the services. This setting also displays the **Management Services** under the **Policies** node in the Policy Studio tree view.

By default, the **Management Services** group consists of the following:

HTTP Interface:
By default, the Admin Node Manager exposes all its management services on port `8090` so that they can be configured remotely. At startup, the Policy Studio can connect to this port to read and write API Gateway configuration data. By default, the Oracle API Gateway Analytics exposes all its management services on port `8040`. For more details, see the section called “Changing the Management Services Port”.

Relative Path: /
The **/** Relative Path is mapped to a default management policy called **Protect Management Interface**, which is available under the **Management Services** Policy Container. This policy performs HTTP Basic Authentication and passes control to the **Call Internal Service** filter. This is a special filter that dispatches a message to a Servlet Application or Static Content Provider based on the path on which the request was received.

For example, with the default configuration, assume that a request is received on `http://localhost:8090/configuration`. The following steps summarize the request processing cycle:

1. When a Relative Path of `/` is configured, it matches all incoming requests, and requests are dispatched to whatever policy the Relative Path is mapped to. In this case, the Relative Path is mapped to the **Protect Management Interface** policy, and so the request is passed to this policy.

2. The **Protect Management Interface** policy performs HTTP basic authentication on the originator of the request. Authentication is necessary because all configuration operations are considered privileged operations and should only be carried out by those with the authority to do so. If the originator can be successfully authenticated, the **Call Internal Service** filter is invoked.

3. The **Call Internal Service** filter is a special filter that passes messages to a Servlet Application or Static Content Provider. In this case, because the message is received on the management interface (port 8090), the filter attempts to match the Relative Path on which the request was received against all the Servlets and Content Providers configured in the same **Services Group** as this interface.

4. The configured Servlets and Content Providers for the **Management Services** group include `/configuration/` and `/api/`. Because the request is received on the `/configuration/` path, this matches the `/configuration/` Servlet Application, which is invoked.

Servlet Application: /configuration/
The Policy Studio running on a different host to the API Gateway can connect to this URL to remotely configure the API Gateway. For example if the API Gateway is running on a host called SERVER, the Policy Studio can connect to `http://SERVER:8090/configuration/` on startup so that it can remotely configure policies running at the API Gateway on the SERVER host.

Important

Changing the Interfaces, Relative Path, Servlet Applications, or Static Content Provider exposed under the **Management Services** group may prevent the Admin Node Manager from functioning correctly. Because of this, the **Management Services** group is hidden by default, and should only be modified under strict supervision from the Oracle Support team.

Changing the Management Services Port

The default Management Services port used by the Admin Node Manager is 8090. To specify a different port, perform the following steps:

1. In the Policy Studio main menu, select **Window** -> **Preferences**, and ensure that the **Show Management Services** setting is selected.
2. Under the **Listeners** node in the Policy Studio tree, right-click the **Management Services** HTTP Interface, and select **Edit**.
3. Specify an updated value in the **Port** field (for example, `8091`), and click **OK**.
4. Click the **Deploy** button in the Policy Studio toolbar, or press F6 to deploy the update.
5. Restart Policy Studio. You must restart Policy Studio when Management Services are updated.
6. Use the updated port number in the URL to reconnect Policy Studio (for example, `https://HOST:8091/api`).

Important

Management Services apply to the Admin Node Manager and Oracle API Gateway Analytics only. You should only modify **Management Services** under strict advice and supervision from the Oracle Support team.

Configuring SMTP Services

Overview

The API Gateway provides support for Simple Mail Transfer Protocol (SMTP), which enables the API Gateway to receive email and to act as a mail relay. The API Gateway can accept incoming email messages using the SMTP protocol, and then forward them on to a configured mail server. You can also use the Policy Studio to configure optional policies for specific SMTP commands (for example, `HELO/EHLO`, `AUTH`, `MAIL FROM`, and so on).

When an SMTP command is configured in the Policy Studio, each time the SMTP command is accepted by the API Gateway, the appropriate policy is executed. When the policy completes successfully, the SMTP conversation resumes. This topic shows how to configure SMTP services, interfaces, and handler policies using the Policy Studio.

Adding an SMTP Service

To add an SMTP service to enable the API Gateway to accept SMTP connections, perform the following steps in the Policy Studio:

1. Under the **Listeners** node in the tree, select a Process node (for example, the default **Oracle API Gateway**).
2. Right-click, and select **Add SMTP Services**.
3. In the **SMTP Services** dialog, specify a unique name for the SMTP service in the **Name** field.
4. In the **Outgoing Server Settings** section, complete the following settings:

Host	Host name or IP address of the remote mail server. This is the server to which the API Gateway forwards incoming SMTP commands (for example, `smtp.gmail.com`). You can also specify a mail server running locally on the same machine as the API Gateway using an address of `localhost` or `127.0.0.1`.
Port	Port on which to connect to the remote mail server. Defaults to port `25`.

5. In the **Security** section, complete the following settings:

Connection Security	Select the type of security used for the connection to the remote mail server. Defaults to `None`. Other possible values are `SSL` and `STARTTLS`.
Trusted Certificates	Use this tab to select the trusted CA certificates used in the security handshake for the connection to the remote mail server. This field is mandatory if SSL or STARTTLS connection security is selected.
Client SSL Authentication	Use this tab to specify the trusted client certificates used in the security handshake for the connection to the remote mail server. This field is optional if SSL or STARTTLS connection security is selected.
Advanced	Use this tab to specify a list of ciphers to use during the security handshake for the connection to the remote mail server. Defaults to `DEFAULT`. For more details, see the OpenSSL ciphers manpage. This field is optional if SSL or STARTTLS connection security is selected.

6. In the **Authentication** section, complete the following settings:

Username	Specify the username used to authenticate the API Gateway with a remote SMTP server using the `AUTH` SMTP command. For more details, see the section on SMTP Authentication.
Password	Specify the password used to authenticate the API Gateway with a remote SMTP server using the `AUTH` SMTP command. For more details, see the section on SMTP Authentication.

7. Select the **Include in real time monitoring** checkbox to monitor the SMTP services using the API Gateway real-time monitoring and API Gateway Analytics tools.
8. **Click OK**. This creates a tree node for the SMTP service under the selected process in the **Services** tree.

Adding an SMTP Interface

When you have configured the outbound SMTP protocol, you must then set up an inbound interface to accept client connections. You can choose from the following interface types:

TCP	Non-secure connection. All traffic is sent in-the-clear.
SSL	SSL handshake is performed at connection time, so the entire SMTP conversation is secure.
STARTTLS	Initial connection is in the clear. The API Gateway advertises STARTTLS during the initial SMTP `HELO/EHLO` handshake. If the client supports this, it can send a STARTTLS command to the API Gateway, which in turn promotes connection security, and upgrades the connection to SSL/TLS.

Because the SSL and STARTTLS interface types have the potential to be secure (STARTTLS starts off non-secure, but can be upgraded during the SMTP conversation), a common configuration screen is used for both protocols in the Policy Studio.

To configure an inbound interface, perform the following steps in the Policy Studio:

1. Under the **Listeners** node in the tree, select the SMTP node under the Process.
2. Right-click, and select **Add Interface** -> interface type (**TCP**, **SSL**, or **STARTTLS**).
3. Complete the settings on the relevant dialog. For full details on these settings, see the *Configuring HTTP Services* topic.
4. **Click OK**.

Configuring Policy Handlers for SMTP Commands

You can use the Policy Studio to configure optional policy handlers for each of the following SMTP commands:

- `HELO/EHLO`
- `AUTH`
- `MAIL FROM`

- RCPT TO
- DATA

The next sections explain how to configure policy handlers for each command.

Adding an HELO/EHLO Policy Handler

The **HELO/EHLO** policy handler is invoked when a HELO/EHLO SMTP command is received from a client. This handler enables you to modify the HELO/EHLO greeting and the client domain. You can configure the greeting message sent back to the client from the API Gateway during the HELO/EHLO handshake as required. You can also configure a policy to replace the value of smtp.helo.greeting. The domain specified by the connected client in the HELO/EHLO command can be modified before forwarding on to the remote mail server. You can also configure a policy to replace the value of smtp.helo.domain.

To configure a policy handler for the HELO/EHLO command, perform the following steps:

1. Under the **Listeners** node in the tree, select the SMTP node under the Process.
2. Right-click, and select **Add Policy Handler** -> **HELO/EHLO**.
3. In the **Configure HELO Host** dialog, specify the **Greeting** to be sent back to the client as part of the HELO/EHLO handshake. Defaults to Hello ${smtp.helo.domain}.
4. In the **Policy** tree, select the policy that you wish to handle the HELO/EHLO command.
5. **Click OK**.

Message attributes
The following message attributes are generated during processing:

Message Attribute	***Description***
smtp.helo.domain	The domain specified by the connecting client in its HELO/EHLO SMTP command.
smtp.helo.greeting	The HELO greeting to be sent back to the client after HELO/EHLO processing is performed. The default value is: Hello ${smtp.helo.domain}.
monitoring.enabled	true if monitoring is enabled for the protocol, otherwise false.
message.monitoring.enabled	true if message-monitoring is enabled for the protocol, otherwise false.

Adding an AUTH Policy Handler

The **AUTH** policy handler is invoked when an AUTH SMTP command is received from a client. You can use the **AUTH** handler to run a policy to perform user authentication checks. For example, during the Authentication phase of the SMTP conversation, the client-supplied username and password can be verified against an Authentication Repository using a policy containing an **Attribute Authentication** filter. For details on possible authentication scenarios, see the section on SMTP Authentication.

To configure a policy handler for the AUTH command, perform the following steps:

1. Under the **Listeners** node in the tree, select the **SMTP** node under the Process.
2. Right-click, and select **Add Policy Handler** -> **AUTH**.
3. In the **Configure AUTH** dialog, in the **Policy** tree, select the policy that you wish to handle the AUTH command.

4. **Click OK**.

Message attributes
The following message attributes are generated during processing:

Message Attribute	*Description*
`authentication.subject.id`	The username supplied by the client.
`authentication.subject.password`	The password supplied by the client.
`monitoring.enabled`	`true` if monitoring is enabled for the protocol, otherwise `false`.
`message.monitoring.enabled`	`true` if message-monitoring is enabled for the protocol, otherwise `false`.

Adding a MAIL Policy Handler

The **MAIL** policy handler is invoked when a `MAIL FROM` SMTP command is received from a client. Emails can be rejected based on wildcard matching of the supplied sender address in the `MAIL FROM` SMTP command. For example, email addresses containing `GMAIL.COM` (`fromAddress` of `*@gmail.com`) as the domain could be accepted using a simple **True** filter. Whereas, email addresses containing `YAHOO.COM` (`fromAddress` of `*@yahoo.com`) could be rejected using a simple **False** filter.

To configure a policy handler for the `MAIL FROM` command, perform the following steps:

1. Under the **Listeners** node in the tree, select the SMTP node under the Process.
2. Right-click, and select **Add Policy Handler** -> **MAIL**.
3. In the **Configure MAIL Address** dialog, you must specify the **From Address**. This is an email address used to filter addresses specified in the `MAIL FROM` SMTP command. You can specify this as a wildcard. The following are some example values:

From Address	*Description*
`*`	Runs the policy for any email address received.
`*@gmail.com`	Runs the policy for all email addresses with the `gmail.com` domain.
`S*@oracle.*`	Runs the policy for all email addresses with any `oracle` domain, and beginning with the letter `s`.

The policy selection is performed on a best-match basis.

4. In the **Policy** tree, select the policy that you wish to handle the `MAIL FROM` command.
5. **Click OK**.

You can configure multiple **MAIL** handlers so that different policies are executed, depending on the received mail address.

Message attributes
The following message attributes are generated during processing:

Message Attribute	*Description*
`smtp.mail.from`	The email address specified in the `MAIL FROM` SMTP command received from the client.
`monitoring.enabled`	`true` if monitoring is enabled for the protocol, otherwise `false`.
`message.monitoring.enabled`	`true` if message-monitoring is enabled for the protocol, otherwise `false`.

Adding a RCPT Policy Handler

The **RCPT** policy handler is invoked when a `RCPT TO` SMTP command is received from a client. You can use this handler to filter addresses specified in the `RCPT TO` SMTP command. Recipients can be rejected based on wildcard matching of the supplied recipient address in the `RCPT` SMTP command. For example, recipient addresses containing `GMAIL.COM` (`toAddress` of `*@gmail.com`) as the domain could be accepted using a simple **True** filter. Whereas, addresses containing `YAHOO.COM` (`toAddress` of `*@yahoo.com`) could be rejected using a simple **False** filter. You can configure multiple **RCPT** handlers so that different policies are executed, depending on the received email address.

To configure a policy handler for the `RCPT TO` command, perform the following steps:

1. Under the **Listeners** node in the tree, select the SMTP node under the Process.
2. Right-click, and select **Add Policy Handler** -> **RCPT**.
3. In the **Configure Recipient Address** dialog, you must specify the **To Address**. This is an email address used to filter addresses specified in the `RCPT TO` SMTP command. You can specify this as a wildcard. The following are some example values:

To Address	*Description*
`*`	Runs the policy for any email address received.
`*@oracle.com`	Runs the policy for all email addresses with the `oracle.com` domain.
`d*@yahoo.*`	Runs the policy for all email addresses with any `yahoo` domain, and beginning with the letter d.

The policy selection is performed on a best-match basis.

4. In the **Policy** tree, select the policy that you wish to handle the `MAIL FROM` command.
5. **Click OK**.

Message attributes
The following message attributes are generated during processing:

Message Attribute	*Description*
`smtp.rcpt.to`	The email address specified in the `RCPT TO` SMTP command received from the client.
`monitoring.enabled`	`true` if monitoring is enabled for the protocol, otherwise `false`.
`message.monitoring.enabled`	`true` if message-monitoring is enabled for the protocol, otherwise `false`.

Adding a DATA Policy Handler

The **DATA** policy handler is invoked when a `DATA` SMTP command is received from a client. For example, for emails that contain SOAP/XML content, you can add an XML signature to the XML data, stored in the `content.body` message attribute, using an **XML Signature Generation** filter. For emails containing attachments, the attached mail data can be run through one of the API Gateway anti-virus filters. Alternatively, you can use **SMIME Encrypt** or **SMIME Decrypt** filters to encrypt or decrypt emails (including attachments) passing through the API Gateway. You can also digitally sign emails using an **SMIME Sign** filter, or verify signatures on already digitally signed emails using an **SMIME Verify** filter.

To configure a policy handler for the `DATA` command, perform the following steps:

1. Under the **Listeners** node in the tree, select the **SMTP** node under the Process.
2. Right-click, and select **Add Policy Handler** -> **DATA**.
3. In the **Policy** tree, select the policy that you wish to handle the `DATA` command.
4. **Click OK**.

Message attributes
The following message attributes are added during processing:

Message Attribute	***Description***
`content.body`	The stream representing the body of the mail. **Note** The `content.body` does not include MIME headers.
`monitoring.enabled`	`true` if monitoring is enabled for the protocol, otherwise `false`.
`message.monitoring.enabled`	`true` if message monitoring is enabled for the protocol, otherwise `false`.

SMTP Authentication

The SMTP protocol supports Extended SMTP (ESMTP) PLAIN authentication. The following matrix shows the possible authentication scenarios and actions based on the SMTP Services configuration:

Scenario	***AUTH Handler***	***AUTH Username and Password***	***Mail Server advertises AUTH***	***API Gateway advertises AUTH***	***Proxy client AUTH***	***Authenticate API Gateway to Server***
1	No	No	No	No	No	No
2	No	No	Yes	Yes	Yes	No
3	No	Yes	No	No	No	No
4	No	Yes	Yes	No	No	Yes
5	Yes	No	No	Yes	No	No
6	Yes	No	Yes	Yes	No	No
7	Yes	Yes	No	Yes	No	No

Scenario	***AUTH Handler***	***AUTH Username and Password***	***Mail Server advertises AUTH***	***API Gateway advertises AUTH***	***Proxy client AUTH***	***Authenticate API Gateway to Server***
8	Yes	Yes	Yes	Yes	No	Yes

These authentication scenarios are described as follows:

1. No authentication username and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. The mail server does not advertise authentication so the API Gateway does not advertise AUTH to the client. The client authentication is not proxied because the server does not support it.
2. No authentication username and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. The mail server advertises AUTH, so the API Gateway advertises AUTH to the client. No AUTH handler is configured, so the client authentication details are proxied to the server.
3. Same as 1 above.
4. The authentication username and password are specified so the API Gateway authenticates with the server. The mail server advertises AUTH, but because a username and password are specified, the API Gateway does not advertise AUTH to the client because the API Gateway authenticates with the server using the configured credentials. This also implies no client authentication proxying.
5. No authentication username and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. AUTH handler configured, which implies the API Gateway performs authentication, so advertise AUTH to the client.
6. Same as 5 above.
7. AUTH handler configured, which implies the API Gateway performs authentication, so advertise AUTH to the client. No proxying occurs because the API Gateway performs the authentication. No authentication is performed with the server because the server does not support it.
8. AUTH advertised to the client because the API Gateway performs authentication (and the mail server supports it). AUTH handler configured, which implies the API Gateway performs authentication. No proxying occurs because the API Gateway performs the authentication. Authentication is performed with the server because the server supports AUTH and a username and password is configured.

SMTP Content-Transfer-Encoding

The SMTP protocol supports automatic Content-Transfer-Encoding/Decoding. For `DATA` SMTP commands, the content of the incoming mail body may be encoded. To enable policy filters to view and/or manipulate the raw body data, the contents are automatically decoded before policy execution, and re-encoded afterwards (before being forwarded on to the configured outbound mail server).

Supported encodings
The following encodings are supported:

- Base64
- 7-bit
- 8-bit
- quoted-printable
- binary

However, Base64 is the only encoding that results in decoding/re-encoding of the mail data.

Multi-part MIME content, generally used for sending attachments in SMTP, is also supported. Each separate body in the multi-part is checked for a Content-Transfer-Encoding, and the decoding/re-encoding is performed as appropriate.

Deployment Example

This section provides a step-by-step example of how to configure and deploy SMTP services using the API Gateway. In this example, the API Gateway acts as a relay between a Thunderbird email client and the Google Gmail service.

Configuring the API Gateway SMTP Services

The API Gateway connects to the Gmail STARTTLS interface, which is available at `smtp.gmail.com`, and listening on port `587`. To configure the SMTP Services, perform the following steps in the Policy Studio:

1. Under the **Listeners** node in the tree, select a Process node (for example, the default **Oracle API Gateway**).
2. Right-click, and select **Add SMTP Services**.
3. Enter `smtp.gmail.com` for the **Host**.
4. Enter `587` for the **Port**.
5. Select `STARTTLS` from the **Connection Security** drop-down list. This is selected because `smtp.gmail.com:587` exposes the Gmail STARTTLS SMTP interface.
6. Because STARTTLS has the potential to be upgraded to a secure connection, you must also select some **Trusted Certificates**.
7. Accept all other defaults, and click **OK** to add the SMTP services.

Configuring the SMTP Client Interface

To configure a STARTTLS client interface, perform the following steps in the Policy Studio:

1. Right-click the **SMTP** Services node, and select **Add Interface** -> **STARTTLS**.
2. Enter a **Port** (for example, `8026`). This is the port on which the API Gateway's incoming SMTP traffic is accepted. You can enter any port that is not already in use.
3. Because STARTTLS has the potential to be upgraded to a secure connection, you must configure a trusted certificate. Click the **X.509 Certificate** button.
4. Select a certificate in the **Select Certificate** dialog.
5. **Click OK** to return to the **Configure STARTTLS Interface** dialog.
6. When the certificate has been configured, accept all other defaults, and click **OK** to add the incoming STARTTLS interface.

When the SMTP services and STARTTLS client interface have been configured, you must deploy the changes to the API Gateway.

Configuring Thunderbird Client Settings

This example uses Thunderbird as the email client. However, you can use any standard email client that supports SMTP. Thunderbird is available as a free download from http://www.mozillamessaging.com/.

To configure a STARTTLS outgoing server in your Thunderbird client, perform the following steps:

1. Launch the Thunderbird email client.
2. From the main menu, select **Tools** -> **Account Settings**.
3. Expand the **Local Folders** tree node in the left pane.
4. Select the **Outgoing Server** node to create a new outgoing server configuration.
5. Click **Add** to display the **SMTP Server** dialog.
6. Enter `Oracle API Gateway [STARTTLS]` in the **Description** field.
7. Enter `localhost` (or the IP Address of the machine on which the API Gateway service is running) in the **Server Name** field.

8. Enter `8026` in the **Port** field. This will send SMTP traffic to the STARTTLS interface configured above, so the ports must match.
9. Select `STARTTLS` from the **Connection security** drop-down list. Traffic on this connection may be upgraded to secure during the SMTP conversation.
10. Select `Normal Password` from the **Authentication method** drop-down list. This indicates that Authentication will be performed.
11. Enter a valid Gmail user-name for the **User Name**.
12. Click **OK** to add the new outgoing server configuration.

Configuring Certificates in Thunderbird

To enable Thunderbird to successfully negotiate the STARTTLS conversation with the API Gateway, you must import a CA certificate into Thunderbird. This is also because a certificate was already generated and imported into the API Gateway when configuring its STARTTLS client interface.

To configure a STARTTLS outgoing server in your Thunderbird client, perform the following steps:

1. From the Thunderbird main menu, select **Tools** -> **Options**.
2. Select the **Certificates** tab.
3. Click the **View Certificates** button, to display the **Certificate Manager** dialog.
4. Click **Import**, and import the appropriate CA certificate.
5. Click **OK** when finished.

Testing the STARTTLS Client Interface

To test the STARTTLS client interface using Thunderbird, perform the following steps:

1. Launch the Thunderbird email client, and create a new mail message.
2. Enter a valid Gmail address in the **To** field.
3. Enter `API Gateway Test` as the **Subject**.
4. Enter `This mail has been sent using Oracle API Gateway` in the mail body.
5. To specify the appropriate outgoing mail server, select **Tools** -> **Account Settings** from the main menu.
6. Select `Oracle API Gateway [STARTTLS] - localhost` from the **Outgoing Server** drop-down list.
7. Click **OK**.
8. Send the mail.

The following example from the API Gateway trace shows the SMTP commands that occur. Commands marked in **bold text** shows traffic from the Thunderbird client to the API Gateway and vice versa. Commands marked in ***bold italics*** shows traffic from the API Gateway to the Gmail server at `smtp.gmail.com:587`, and vice versa.

```
DEBUG  14:46:46:546 [14b4] incoming call on interface *:8026 from 127.0.0.1:1487
DEBUG  14:46:46:546 [14b4] new connection 08133248,  settings source incoming interface
(force 1.0=no, idleTimeout=60000, activeTimeout=60000)
DATA   14:46:46:546 [14b4] snd 0018: <220 doejOracle>
DATA   14:46:46:562 [14b4] rcv 18: <EHLO [127.0.0.1]>
DEBUG  14:46:46:562 [14b4] 080BE260: new connection cache set SMTP Client
DEBUG  14:46:46:562 [159c] idle connection monitor thread running
DEBUG  14:46:46:562 [14b4] new endpoint smtp.gmail.com:587
DEBUG  14:46:46:640 [14b4] Resolved smtp.gmail.com:587 to:
DEBUG  14:46:46:640 [14b4]     209.85.227.109:587
DEBUG  14:46:46:718 [14b4] connected to 209.85.227.109:587
DEBUG  14:46:46:718 [14b4] new connection 08135BA0,  settings source service-wide
defaults (force 1.0=no, idleTimeout=15000, activeTimeout=30000)
DATA   14:46:46:765 [14b4] rcv 44: <220 mx.google.com ESMTP v11sm7979387weq.40>
DATA   14:46:46:765 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA   14:46:46:812 [14b4] rcv 125: <250-mx.google.com at your service, [87.198.245.194]
```

```
250-SIZE 35651584
250-8BITMIME
250-STARTTLS
250 ENHANCEDSTATUSCODES>
DATA   14:46:46:812 [14b4] snd 0010: <starttls>
DATA   14:46:46:843 [14b4] rcv 30: <220 2.0.0 Ready to start TLS>
DEBUG  14:46:46:843 [14b4] push SSL protocol on to connection
DEBUG  14:46:46:906 [14b4] No SSL host name provided: using default certificate for
interface
DEBUG  14:46:46:906 [14b4] verifyCert: preverify=1, depth=2, subject /C=US/O=Equifax/
OU=Equifax Secure Certificate Authority, issuer /C=US/O=Equifax/OU=Equifax Secure
Certificate Authority
DEBUG  14:46:46:906 [14b4] ca cert? 1
DEBUG  14:46:46:906 [14b4] verifyCert: preverify=1, depth=1, subject /O=Google
Inc/CN=Google Internet Authority, issuer /C=US/O=Equifax/OU=Equifax Secure Certificate
Authority
DEBUG  14:46:46:906 [14b4] verifyCert: preverify=1, depth=0, subject
/C=US/ST=California/L=Mountain View/O=Google Inc/CN=smtp.gmail.com,
issuer /C=US/O=Google Inc/CN=Google Internet Authority
DEBUG  14:46:46:952 [14b4] negotiated SSL cipher "RC4-MD5",session 00000000 (not reused)
DATA   14:46:46:952 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA   14:46:46:999 [14b4] rcv 140: <250-mx.google.com at your service, [87.198.245.194]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES>
DATA   14:46:46:999 [14b4] snd 0109: <250-OracleAPI Gateway Hello [127.0.0.1]
250-SIZE 35651584
250-8BITMIME
250-STARTTLS
250 ENHANCEDSTATUSCODES>
DEBUG  14:46:46:999 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA   14:46:46:999 [14b4] rcv 10: <STARTTLS>
DATA   14:46:46:999 [14b4] snd 0014: <220 Go ahead>
DEBUG  14:46:46:999 [14b4] push SSL protocol on to connection
DEBUG  14:46:46:999 [14b4] Servername CB: SSL host name: localhost, not in host map -
using default certificate for interface
DEBUG  14:46:47:031 [14b4] negotiated SSL cipher "AES256-SHA", session 00000000
(not reused)
DATA   14:46:47:031 [14b4] rcv 18: <EHLO [127.0.0.1]>
DATA   14:46:47:031 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA   14:46:47:077 [14b4] rcv 140: <250-mx.google.com at your service, [87.198.245.194]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES>
DATA   14:46:47:077 [14b4] snd 0124: <250-OracleAPI Gateway Hello [127.0.0.1]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES>
DEBUG  14:46:47:077 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA   14:46:47:077 [14b4] rcv 41: <AUTH PLAIN ADGzaHllaDe0SHF1ex2r82Su555=>
DATA   14:46:47:077 [14b4] snd 0041: <auth PLAIN ADGzaHllaDe0SHF1ex2r82Su555=>
DATA   14:46:47:718 [14b4] rcv 20: <235 2.7.0 Accepted>
DATA   14:46:47:718 [14b4] snd 0020: <235 2.7.0 Accepted>
DATA   14:46:47:718 [14b4] rcv 45: <MAIL FROM:<john.doe@oracle.com> SIZE=444>
DATA   14:46:47:718 [14b4] snd 0036: <mail from:<john.doe@oracle.com>>
DATA   14:46:47:765 [14b4] rcv 33: <250 2.1.0 OK v11sm7979387weq.40>
DATA   14:46:47:765 [14b4] snd 0033: <250 2.1.0 OK v11sm7979387weq.40>
DATA   14:46:47:765 [14b4] rcv 30: <RCPT TO:<test@gmail.com>>
DATA   14:46:47:765 [14b4] snd 0030: <rcpt to:<test@gmail.com>>
DATA   14:46:47:812 [14b4] rcv 33: <250 2.1.5 OK v11sm7979387weq.40>
DATA   14:46:47:812 [14b4] snd 0033: <250 2.1.5 OK v11sm7979387weq.40>
DATA   14:46:47:812 [14b4] rcv 6: <DATA>
DATA   14:46:47:812 [14b4] snd 0006: <data>
DATA   14:46:48:609 [14b4] rcv 34: <354  Go ahead v11sm7979387weq.40>
DATA   14:46:48:609 [14b4] snd 0008: <354 OK>
DATA   14:46:48:609 [14b4] rcv 447: <Message-ID: <4CB85B46.4060205@oracle.com>
Date: Fri, 15 Oct 2010 14:46:46 +0100
From: John Doe <john.doe@oracle.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.2.9) Gecko/20100915
Thunderbird/3.1.4
MIME-Version: 1.0
To: test@gmail.com
Subject: API Gateway Test
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit

 This mail has been sent via Oracle API Gateway

.>
DATA    14:46:48:609 [14b4] snd 0442: <Message-ID: <4CB85B46.4060205@oracle.com>
Date: Fri, 15 Oct 2010 14:46:46 +0100
From: John Doe <john.doe@oracle.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.2.9) Gecko/20100915
Thunderbird/3.1.4
MIME-Version: 1.0
To: test@gmail.com
Subject: API Gateway Test
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit

 This mail has been sent via Oracle API Gateway>
DATA    14:46:48:609 [14b4] snd 0005: <.>
DATA    14:46:49:874 [14b4] rcv 44: <250 2.0.0 OK 1287150409 v11sm7979387weq.40>
DATA    14:46:49:874 [14b4] snd 0044: <250 2.0.0 OK 1287150409 v11sm7979387weq.40>
DEBUG   14:46:49:874 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA    14:46:49:874 [14b4] rcv 6: <QUIT>
DATA    14:46:49:874 [14b4] snd 0006: <quit>
DATA    14:46:49:921 [14b4] rcv 49: <221 2.0.0 closing connection v11sm7979387weq.40>
DEBUG   14:46:49:921 [14b4] delete transaction 08040BD8 on connection 08135BA0
DATA    14:46:49:921 [14b4] snd 0049: <221 2.0.0 closing connection v11sm7979387weq.40>
DEBUG   14:46:49:921 [14b4] delete transaction 0B95D2C0 on connection 08133248
DEBUG   14:46:49:921 [14b4] delete connection 08133248, current transaction 00000000
```

File Transfer Service

Overview

The API Gateway can act as a file transfer service that listens on a port for remote clients to connect to it. The API Gateway file transfer service supports the following protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

For all file transfer protocols, you must configure a file upload policy and an authentication policy. For FTP and FTPS, you must configure a password authentication policy. While for SFTP, you can configure a password authentication policy or a public key authentication policy. The API Gateway can also restrict access to the server based on IP address.

When a file transfer service is configured, users are presented with a personal file system view when they log in. The root of this file system is specified in a configurable request directory. Any files they upload are processed by the file upload policy. If this policy succeeds, the output of the policy is stored in a configurable response directory. If the policy fails, the original file is moved to a configurable quarantine directory.

Configuring a file transfer service can be useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the other system can upload files to the API Gateway. The added benefit is that the file transfer can be controlled and secured using API Gateway policies designed to suit system needs.

General Configuration

You can configure the following settings in the **General** section:

Name:
Enter an appropriate name for the file transfer service.

Service Type:
Select the file transfer protocol type for this service from the drop-down list (`ftp`, `ftps`, or `sftp`). Defaults to `ftp`.

Implicit:
This setting applies to FTPS only. When selected, security is automatically enabled as soon as the remote client makes a connection to the file transfer service. No clear text is passed between the client and server at any time. In this case, the file transfer service defines a specific port for the remote client to use for secure connections (`990`). This setting is not selected by default.

Explicit:
This setting applies to FTPS only. When selected, the remote client must explicitly request security from the file transfer server, and negotiate the required security. If the client does not request security, the file transfer server can allow the client to continue insecure or refuse and/or limit the connection. This setting is selected by default.

Binding Address:
Enter a network interface to bind to. Defaults to `*`, which means bind to all available network interfaces on the host on which the API Gateway is installed. You can enter an IP address to bind to a specific network interface.

Port:
Enter a file transfer port to listen on for remote clients to connect to. Defaults to `21`.

Note

The API Gateway must execute with superuser privileges to bind to a port number less than `1024` on Linux and Solaris.

File Upload

You can configure the following settings in the **File Upload** section:

Request Directory:
Specifies the directory into which files and directories are uploaded. Defaults to `${environment.VINSTDIR}/file-transfer/in/`.

Delete File on Successful Response:
Select whether to delete the uploaded files from the **Request Directory** when successfully processed. This setting is not selected by default.

Response Directory:
Specifies the name of the directory into which files output by the API Gateway processing of uploaded files are placed if the **File Upload** policy returns true. Defaults to `out`. The original files remain in the **Request Directory**.

Response Suffix:
Specifies the filename suffix that is appended to the output files in the **Response Directory**. Defaults to `.resp.${id}`, which enables a unique suffix to be appended to the files.

Quarantine Directory:
Specifies the directory into which the uploaded files are moved if the **File Upload** policy returns false, or an exception is thrown. Defaults to `quarantine`.

Important

The response and quarantine directories can be relative or absolute. Relative directories reside under the request directory. The user can manage the uploaded files using their file transfer session (for example, by accessing API Gateway file processing results). Absolute directories must reside outside of the request directory. The user cannot view or manage uploaded files using their file transfer session. Specifying absolute directories hides API Gateway file processing from the user.

In this way, the request directory can be seen as the user's home directory for the duration of the connection. Therefore, anything created under the same home directory is visible to the user. However, if the response is created outside this directory (for example, in `/tmp/response`), the files are not visible to the user.

Specifying a File Upload Policy
For all file transfer protocols, you must specify a **File Upload** policy. This enables files and directories to be uploaded to a user subdirectory of the **Request Directory**. For example, files uploaded by user `fred` are placed in `${environment.VINSTDIR}/file-transfer/in/persistent/fred`. The specified policy is then invoked with the raw file data. If this policy returns true, the output is placed in the corresponding **Response Directory**. If this policy returns false or an exception is thrown, the uploaded file is moved to the **Quarantine Directory**.

To specify a **File Upload** policy, perform the following steps:

1. Click the **Add** button to display the dialog.
2. In the **Pattern** field, select or enter a regular expression to match against the filename. For example, the following expression means that the configured policy is run against these files types only:

```
([^\s]+(\.(?i)(xml|xhtml|soap|wsdl|asmx))$)
```

3. In the **Policy** field, click the browse button on the right to select a policy, and click **OK**.
4. Click **OK** to display the configured pattern and policy in the table.

Message Attributes
The **File Upload** policy uses the following message attributes:

- `content.body`: Raw message file content.
- `file.src.name`: Filename of the uploaded file.
- `file.src.path`: Full file path of the uploaded file.

Secure Services

On the **Secure Services** tab, you can configure the following **Client Authentication** policies:

Password Authentication Policy:
For FTP and FTPS, you must configure a **Password Authentication Policy**. Click the browse button on the right to select a configured policy. This policy uses the `authentication.subject.id` and `authentication.subject.password` message attributes, which store the username and password entered by the client user.

Public Key Authentication Policy:
For SFTP, you can configure a **Public Key Authentication Policy** and/or **Password Authentication Policy**. Click the browse button on the right to select a configured policy. This policy uses the `authentication.subject.id` and `authentication.subject.public.key` message attributes, which store the username and public key used by the client.

You can configure the following **Server Authentication** settings:

Server Certificate:
For FTPS or SFTP, click the **Signing Key** button to specify a server certificate. You can select a certificate in the dialog, or click to create or import a certificate. The selected certificate must contain a private key. For more details, see the *Certificates and Keys* topic. Alternatively, you can specify a certificate to bind to at runtime using an environment variable selector (for example, `${env.serverCertificate}`). For more details, see *Deploying the API Gateway in Multiple Environments*.

Server Key Pair:
For SFTP, click the button on the right, and select a previously configured key pair that the file transfer service must present from the tree. To add a key pair, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Certificates and Keys** node in the Policy Studio tree. For more details, see the topic on *Certificates and Keys*.

Commands

For FTP and FTPS, the **Commands** tab enables you to specify commands that can be enabled for this service (other FTP and FTPS commands are enabled by default). The following commands are specified in the table:

- `DELE`: Allow user to delete files
- `PASV`: Support passive mode
- `REST`: Support restart mode
- `RMD`: Allow user to remove directories
- `STOU`: Support unique filename

To enable an existing command, click **Edit**, select **Enabled**, and click **OK**. The command is displayed as enabled in the table.

Adding New Commands
To add a new command, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter the command **Name** (for example, `STAT`).
3. Select the file transfer protocol **Type** from the drop-down list (for example, `ftp` or `ftp(s)`).
4. Enter the command **Description**.
5. Select whether the command is **Enabled**.
6. Click **OK** to display the new command in the table.

Supported Commands
For a full list of supported commands, see http://mina.apache.org/ftpserver/ftp-commands.html [http://mina.apache.org/ftpserver/ftp-commands.html]

Access Control

The **Access Control** tab enables you to restrict or block access to the file transfer service based on IP address. All IP addresses are allowed by default.

Restrict Access to the following IP Ranges:
To restrict access to specified IP addresses, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter an **IP Address** (for example, `192.168.0.16`).
3. Enter a **Net Mask** for the specified IP address. For example, if you wish to restrict access to the specified IP address only, enter `255.255.255.255`. Alternatively, you can restrict access to a range of IP addresses by entering a value such as `255.255.255.253`, which restricts access to `192.168.0.16`, `192.168.0.17`, and `192.168.0.18`.
4. Click **OK** to display the IP address details in the table.

Block Access to the following IP Ranges:
To block access to specified IP addresses, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter an **IP Address** (for example, `192.168.0.16`).
3. Enter a subnet **Mask** for the specified IP address. For example, if you wish to block access to the specified IP address only, enter `255.255.255.255`. Alternatively, you can block access to a range of IP addresses by entering a value such as `255.255.255.253`, which blocks access to `192.168.0.16`, `192.168.0.17`, and `192.168.0.18`.
4. Click **OK** to display the IP address details in the table.

Messages

For FTP and FTPS, the you can specify a welcome message and a goodbye message on the **Messages** tab. These are output to the user at the start and at the end of each session:

Welcome Message:
Enter a short welcome text message for the file transfer service (for example, `Connected to FTP Test Service...`).

Goodbye Message:
Enter a short goodbye text message for the file transfer service (for example, `Leaving FTP Test Service...`).

Directory

You can configure the following settings on the **Directory** tab:

Directory Type:
Select one of the following directory types:

- **Persistent**: This is a sharable directory created per user, which persists between connections (for example, `${environment.VINSTDIR}/file-transfer/in/persistent/fred`). This is the default directory type.
- **Transient**: This is an isolated directory created per connection, which is destroyed after the connection (for example, `${environment.VINSTDIR}/file-transfer/in/2/fred`).

Note

Some clients, notably FileZilla, generate multiple client connection sessions. For these clients, files uploaded in one session will not be available in the viewing session.

Directory expiry in seconds:
Specifies how long the directory remains on the system. Defaults to 3600 seconds (1 hour). A setting of 0 seconds means that the directory never expires.

Directory expiry check period in seconds:
Specifies how often the API Gateway checks to see if a directory has expired. Defaults to 1800 seconds (30 minutes). A setting of 0 seconds means that it never checks.

Policy Execution Scheduling

Overview

You can configure a policy execution scheduler at the level of the API Gateway process. This enables you to schedule the execution of any policy on a specified date and time in a recurring manner. The API Gateway provides a pre-configured library of schedules to select from when creating a policy execution scheduler. You can also add your own schedules to the globally available library in the Policy Studio.

You can use policy execution scheduling in any policy (for example, to perform a message health check). This feature is also useful when polling a service to enforce a Service Level Agreement (for example, to ensure the response time is less than 1000 ms, and if not, to send an alert).

Cron Expressions

In the Policy Studio, policy execution schedules are based on cron expressions. A cron expression is a string that specifies a time schedule for triggering an event (for example, executing a policy). It consists of six required fields and one optional field, each separated by a space, which together specify when to trigger the event. For example, the following expression specifies to run at 10:15am every Monday, Tuesday, Wednesday, Thursday, and Friday in 2011:

```
0 15 10 ? * MON-FRI 2011
```

Syntax
The following table shows the syntax used for each field:

Field	***Values***	***Special Characters***
Seconds	`0-59`	`, - * /`
Minutes	`0-59`	`, - * /`
Hours	`0-23`	`, - * /`
Day of Month	`1-31`	`, - * ? / L W`
Month	`1-12 or JAN-DEC`	`, - * /`
Day of Week	`1-7 or SUN-SAT`	`, - * ? / L #`
Year (optional)	`empty or 1970-2199`	`, - * /`

Special Characters
The special characters are explained as follows:

Special Character	***Description***
`,`	Separates values in a list (for example, `MON,WED,SAT` means Mondays, Wednesdays, and Saturdays only).
`-`	Specifies a range of values (for example, 2011-2015 means every year between 2011 and 2015 inclusive).
`*`	Specifies all values of the field (for example, every minute).
`?`	Specifies no value in the Day of Month and Day of Week fields. This enables you to specify a value in one field, but not in the other.
`/`	Specifies time increments (for example, in the Minutes

Special Character	*Description*
	field, 0/15 means minutes 0, 15, 30, and 45, while 5/15 means minutes 5, 20, 35, and 50). Specifying * before the / is the same as specifying 0 as the start value. The / character enables you to turn on every nth value in the set of values for the specified field. For example, 7/6 in the month field only turns on month 7, and does not mean every 6th month.
L	Specifies the last value in the Day of Month and Day of Week fields. In the Day of Month field, this means the last day of the month (for example, January 31, or February 28 in non-leap years). In the Day of Week field, when used alone, this means 7 or SAT. When used after another value, it means the last XXX day of the month (for example, 5L means the last Thursday of the month). When using the L character, do not specify lists or ranges because this can give confusing results.
W	Specifies the weekday (Monday-Friday) nearest the given day. For example, 15W means the nearest weekday to the 15th of the month. If the 15th is a Saturday, the trigger fires on Friday 14th. If the 15th is a Sunday, it fires on Monday 16th. If the 15th is a Tuesday, it fires on Tuesday 15th. However, if you specify 1W, and the 1st is a Saturday, the trigger fires on Monday 3rd to avoid crossing the month boundary. You can only specify the W character for a single day, and not a range or list of days.
#	Specifies the nth XXX weekday of the month in the Day of Week field. For example, a value of FRI#2 means the second Friday of the month. However, if you specify #5, and there are not 5 of the specified Day of the Week in the month, no policy is run that month. When the # character is specified, there can only be one expression in the Day of Week field (for example, 2#1,6#4 is not valid because there are two expressions).

Examples
The following are some of the cron expressions provided in the **Schedule Library** in the Policy Studio:

Cron Expression	*Description*
0 15 10 ? * *	Run at 10:15am every day.
0 15 10 ? * 6L 2011-2015	Run at 10:15am on every last Friday of every month during the years 2011, 2012, 2013, 2014, and 2015.
0 15 10 ? * 6#3	Run at 10:15am on the third Friday of every month.
0 0 10 1,15 * ?	Run at 10am on the 1st and 15th days of the month.
0 10,44 14 ? 3 WED	Run at 2:10pm and at 2:44pm every Wednesday in the month of March.
0,30 * * ? * SAT,SUN	Run every 30 seconds but only on Weekends (Saturday and Sunday).
0 0/5 14,18 * * ?	Run every 5 minutes starting at 2pm and ending at 2:55pm,

Cron Expression	***Description***
	and every 5 minutes starting at 6pm and ending at 6:55pm, every day.
`0 0-5 14 * * ?`	Run every minute starting at 2pm and ending at 2:05pm, every day.

Important

Please note the following:

- Support for specifying both a Day of Week and a Day of Month value is not complete. You must use the ? or `*` character in one of these fields.
- Overflowing ranges with a larger number on the left than the right are supported (for example, `21-2` for 9pm until 2am , or `OCT-MAR`). However, overuse may cause problems with daylight savings (for example, `0 0 14-6 ? * FRI-MON`).

Adding a Schedule

To add a schedule to the globally available library in the Policy Studio, perform the following steps:

1. Select the **Libraries** -> **Schedules** node in the tree.
2. Click the **Add** button at the bottom of the **Schedules** screen.
3. In the **Schedules** dialog, enter a **Name** (for example, `Run every 30 seconds`).
4. Enter a **Cron expression** (for example, `0/30 * * * * ?`). Alternatively, click the browse button to select an expression in **Cron dialog**. For more details, see the topic on *Configuring Cron Expressions*.
5. Click **OK**.

You can also edit or delete a selected schedule using the appropriate button.

Adding a Policy Execution Scheduler

To add a policy execution scheduler in the Policy Studio, perform the following steps:

1. Select the **Listeners** node on the left.
2. Right-click the Process node (for example, the default **Oracle API Gateway**), and select **Add policy execution scheduler**.
3. Click the button next to the **Schedule** field, select a cron expression in the dialog, and click **OK**.
4. Click the button next to the **Policy** field, select a policy in the dialog, and click **OK**. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically.
5. Click **OK**.

FTP Poller

Overview

The FTP Poller enables you to query and retrieve files to be processed by polling a remote file server. When the files are retrieved, they can be passed into the API Gateway core message pipeline for processing. For example, you can use the FTP Poller in cases where an external application drops files on to a remote file server, which can then be validated, modified, and potentially routed on over HTTP or JMS by the API Gateway.

This kind of protocol mediation can be useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the API Gateway can download the files from the remote file server, and then route them on over HTTP to another back-end system. The added benefit is that messages are exposed to the full compliment of message processing filters available in the API Gateway. This ensures that only properly validated messages are routed on to the target system.

The FTP Poller supports the following file transfer protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

To add a new FTP Poller, in the Policy Studio tree, under the **Listeners** node, right-click the process name (for example, `Oracle API Gateway`), and select **FTP Poller** -> **Add**. This topic describes how to configure the fields on the **FTP Poller Settings** dialog.

General Settings

This filter includes the following general settings:

Name:
Enter a descriptive name for this FTP Poller.

Enable:
Select whether this FTP Poller is enabled. This is selected by default.

Host:
Enter the host name of the file transfer server to connect to.

Port:
Enter the port on which to connect to the file transfer server. Defaults to `20`.

User name:
Enter the username to connect to the file transfer server.

Password:
Specify the password for this user.

Scan Details

The fields configured in the **Scan details** section determine when to scan, where to scan, and what files to scan:

Poll every (ms):
Specifies how often in milliseconds the API Gateway scans the specified directory for new files. Defaults to `60000`. To optimize performance, it is good practice to poll often to prevent the number of files from building up.

Look in directory:
Enter the full path of the directory to scan for new files.

For files that match the pattern:
Specifies to scan only for files based on a pattern in a regular expression. For example, if you wish to scan only for files with a particular file extension (for example, `.xml`), enter an appropriate regular expression. Defaults to the following expression:

```
([^\s]+(\.(?i)(xml|xhtml|soap|wsdl|asmx))$)
```

Process file with following policy:
Click the browse button to select the policy to process each file with. For example, this policy may perform tasks such as validation, threat detection, content filtering, or routing over HTTP or JMS.

Delete file when complete:
Select whether to delete each processed file when complete. This is selected by default.

Establish new session for each file found:
Select whether to establish a new file transfer session for each file found. This is selected by default.

Connection Type

The fields configured in the **Connection Type** section determine the type of file transfer connection. Select the connection type from the drop-down list:

- FTP - File Transfer Protocol
- FTPS - FTP over SSL
- SFTP - SSH File Transfer Protocol

FTP and FTPS Connections

The following general settings apply to FTP and FTPS connections:

Passive transfer mode:
Select this option to prevent problems caused by opening outgoing ports in the firewall relative to the file transfer server (for example, when using *active* FTP connections). This is selected by default.

File Type:
Select **ASCII** mode for sending text-based data or **Binary** mode for sending binary data over the connection. Defaults to **ASCII** mode.

FTPS Connections

The following security settings apply to FTPS connections only:

SSL Protocol:
Enter the SSL protocol used (for example, `SSL` or `TLS`). Defaults to `SSL`.

Implicit:
When this option is selected, security is automatically enabled as soon as the **FTP Poller** client makes a connection to the remote file transfer service. No clear text is passed between the client and server at any time. In this case, the client defines a specific port for the remote file transfer service to use for secure connections (`990`). This option is not selected by default.

Explicit:
When this option is selected, the remote file transfer service must explicitly request security from the **FTP Poller** client,

and negotiate the required security. If the file transfer service does not request security, the client can allow the file transfer service to continue insecure or refuse and/or limit the connection. This option is selected by default.

Trusted Certificates:
To connect to a remote file server over SSL, you must trust that server's SSL certificate. When you have imported this certificate into the Certificate Store, you can select it on the **Trusted Certificates** tab.

Client Certificates:
If the remote file server requires the FTP Poller client to present an SSL certificate to it during the SSL handshake for mutual authentication, you must select this certificate from the list on the **Client Certificates** tab. This certificate must have a private key associated with it that is also stored in the Certificate Store.

SFTP Connections

The following security settings apply to SFTP connections only:

Present following key for authentication:
Click the button on the right, and select a previously configured key to be used for authentication from the tree. To add a key, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Certificates and Keys** node in the Policy Studio tree. For more details, see the topic on *Certificates and Keys*.

SFTP host must present key with the following finger print:
Enter the fingerprint of the public key that the SFTP host must present (for example, `43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8`).

Directory Scanner

Overview

The Directory Scanner enables you to scan a specified directory on the filesystem for files containing XML messages. When the messages have been read, they can be passed into the core message pipeline, where the full collection of message processing filters can act on them.

The Directory Scanner is typically used in cases where an external application is dropping XML files (perhaps by FTP) on to the file system so that they can be validated, modified, and potentially routed onwards over HTTP or JMS. Alternatively, they can simply be stored to another directory where the application can pick them up again.

This sort of protocol mediation is very useful in cases where legacy systems are involved. Instead of making drastic changes to the legacy system by adding an HTTP engine, for example, the API Gateway can be used to pull the files from the file system, and then route them on over HTTP to another back-end system. The added benefit is that messages are exposed to the full compliment of message processing filters made available by the API Gateway. This ensures that only properly validated messages are routed on to the target system.

To add a new Directory Scanner, in the Policy Studio tree, under the **Listeners** node, right-click the name of the Process (for example, `Oracle API Gateway`), and select the **Directory Scanner** -> **Add** menu option. This topic describes how to configure the fields on the **Directory Scanner Settings** dialog.

Directory to Scan

The fields configured in this section determine what files to scan, where to scan for them, and when to scan.

Name:
Enter or browse to the directory that the API Gateway scans for XML files.

File Name Pattern:
If you wish to scan only for files based on some pattern, you can specify the pattern as a regular expression. For example, if you wish to scan only for files with a particular file extension, such as `.xml`, you can enter a regular expression such as the following:

```
^[a-zA-Z\s]*.xml
```

Similarly, if a particular naming scheme is used when dropping the files into the configured directory, you can enter a regular expression to scan only for these files. For example, the following regular expression scans only files named using a `yyyy-mm-dd` date format:

```
^(\d{4})[- /.]((1[012])|(0?[1-9]))[- /.]((3[01])|([012]?[1-9])|([12]0))$
```

Poll Rate:
The poll rate entered in milliseconds determines how often the API Gateway scans the directory for new XML files.

Directory for Output

When the Directory Scanner has finished scanning the files it moves them to another directory to avoid processing them again. The fields configured in this section determine where processed files are placed and what they are called.

Name:
Enter the name of the directory to place the processed files. This may be the response from the Web Service (after the policy has routed it onwards and received a response), or a modified message in cases where the policy has inserted a security token into the message, or converted the message (for example, using XSLT).

File Prefix:
If you would like to save processed files into the directory above with a prefix added to the filename, enter the prefix here. For example, you may want to prepend `_PROCESSED` to all processed files.

File Suffix:
Similarly, you can add a suffix to the output files by entering the suffix in this field.

Completed Directory

Processed files are removed from the source directory and placed into this directory post-processing to avoid re-processing the same files over and over again.

Working Directory

Files are moved to the temporary directory specified here during processing. This is necessary in cases where the poll rate is quite low, and there is a chance that the file may be scanned again before it is processed fully and moved to the completed directory.

Policy to Use

The messages in the scanned XML files are passed into the policy selected here. If the policy routes messages to a Web Service, the response from the service is placed in the output directory specified above. Similarly, if the policy modifies the request, for example, by signing it or adding a security token to it, the updated message is also placed in the output directory.

Packet Sniffers

Overview

Packet Sniffers are a type of Passive Service. Rather than opening up a TCP port and *actively* listening for requests, the Packet Sniffer *passively* reads raw data packets off the network interface. The Sniffer assembles these packets into complete messages that can then be passed into an associated policy.

Because the Packet Sniffer operates passively (does not listen on a TCP port) and, therefore, completely transparently to the client, it is most useful for monitoring and managing Web Services. For example, the Sniffer can be deployed on a machine running a Web Server acting as a container for Web Services. Assuming that the Web Server is listening on TCP port 80 for traffic, the Packet Sniffer can be configured to read all packets destined for port 80 (or any other port, if necessary). The packets can then be marshalled into complete HTTP/SOAP messages by the Sniffer and passed into a policy that logs the message to a database, for example.

Important

On Linux and Solaris platforms, the API Gateway must be started by the root user to gain access to the raw packets.

Configuration

Since Packet Sniffers are mainly for use as passive monitoring agents, they are usually created within their own HTTP Service Group. For example, you can create a new Service Group for this purpose by right-clicking on the Process, selecting the **Add HTTP Services** menu option, and entering `Packet Sniffer Group` on the **HTTP Services** dialog.

We can then add a Relative Path Service to this Group by right-clicking the **Packet Sniffer Group**, and selecting the **Add Relative Path** menu option. Enter a path in the field provided, and select the policy that you want to dispatch messages to when the Packet Sniffer detects a request for this path (after it assembles the packets). For example, if the Relative Path is configured as `/a`, and the Packet Sniffer assembles packets into a request for this path, the request will be dispatched to the policy selected in the Relative Path Service.

Finally, you can add the Packet Sniffer by right-clicking the **Packet Sniffer Group** node, selecting **Packet Sniffer**, and then the **Add** menu option. Complete the following fields on the **Packet Sniffer** dialog:

Device to Monitor:
Enter the name or identifier of the network interface that the Packet Sniffer will monitor. The default entry is `any`.

Important

This setting is only valid on UNIX. On UNIX-based systems, network interfaces are usually identified using names like `eth0`, `eth1`, and so on. On Windows, these names are more complicated (for example, `\Device\NPF_{00B756E0-518A-4144 ... }`.

Filter:
The Packet Sniffer can be configured to only intercept certain types of packets. For example, it can ignore all UDP packets, only intercept packets destined for port 80 on the network interface, ignore packets from a certain IP address, listen for all packets on the network, and so on.

The Packet Sniffer uses the **libpcap** library filter language to achieve this. This language has a complicated but powerful syntax that allows you to *filter* what packets are intercepted and what packets are ignored. As a general rule, the syntax consists of one or more expressions combined with conjunctions, such as `and`, `or`, and `not`. The following table lists a few examples of common filters and explains what they filter:

Filter Expression	*Description*
`port 80`	Capture only traffic for the HTTP Port (`80`).
`host 192.168.0.1`	Capture traffic to and from IP address `192.168.0.1`.
`tcp`	Capture only TCP traffic.
`host 192.168.0.1 and port 80`	Capture traffic to and from port 80 on IP address `192.168.0.1`.
`tcp portrange 8080-8090`	Capture all TCP traffic destined for ports from 8080 through to 8090.
`tcp port 8080 and not src host 192.168.0.1`	Capture all TCP traffic destined for port 8080 but not from IP address `192.168.0.1`.

The default filter of `tcp` captures all TCP packets arriving on the network interface. For more information on how to configure filter expressions like these, please refer to the **tcpdump** man page available from http://www.tcpdump.org/tcpdump_man.html.

Promiscuous Mode:
When listening in promiscuous mode, the Packet Sniffer captures all packets on the same Ethernet network, regardless of whether or not the packets are addressed to the network interface that the Sniffer is monitoring.

Messaging System

Overview

A *messaging system* is a loosely coupled, peer-to-peer facility where clients can send messages to, and receive messages from any other client. In a messaging system, a client sends a message to a messaging agent. The recipient of the message can then connect to the same agent and read the message. However, the sender and recipient of the message do not need to be available at the same time to communicate (for example, unlike HTTP). The sender and recipient need only know the name and address of the messaging agent to talk to.

The Java Messaging System (JMS) is an implementation of such a messaging system. It provides an API for creating, sending, receiving, and reading messages. Java-based applications can use it to connect to other messaging system implementations. A *JMS provider* can deliver messages synchronously or asynchronously, which means that the client can fire and forget messages or wait for a response before resuming processing. Furthermore, the JMS API ensures different levels of reliability in terms of message delivery. For example, it can ensure that the message is delivered once and only once, or at least once.

The API Gateway uses the JMS API to connect to other messaging systems that expose a JMS interface, including Oracle WebLogic Server, IBM MQSeries, JBoss Messaging, TIBCO EMS, IBM WebSphere Server, and Progress SonicMQ. As a consumer of a JMS queue or topic, the API Gateway can read XML messages and pass them into a policy for validation. These messages can then be routed on over HTTP or dropped on to another JMS queue or topic.

Important

You must add the JMS provider's JAR files to the API Gateway classpath for this feature to function correctly. Copy the provider JAR files to the `INSTALL_DIR/ext/lib` folder, where `INSTALL_DIR` points to the root of your API Gateway installation.

Configuring a JMS Service

You can configure a global JMS service under the **External Connections** node in Policy Studio by right-clicking the **JMS Services** node, and selecting **Add a JMS Service**. The details entered in the **JMS Service** dialog are used by the API Gateway to drop messages on to a JMS queue or topic. The **Messaging System** Connection filter uses JMS Services configured here to do this.

Alternatively, you can configure a JMS service at the Process level, and configure the API Gateway to consume a JMS queue or topic. Right-click the Process under the **Listeners** node in the Policy Studio, and select **JMS Wizard**.

Configure the following fields on the **JMS Service** dialog:

Name:
Enter a descriptive name for the JMS Provider in the **Name** field.

Provider URL:
Enter the URL of the JMS provider (for example, `jnp://localhost:1099`).

Initial Context Factory:
The API Gateway uses a connection factory to create a connection with a JMS provider. A connection factory encapsulates a set of connection configuration parameters that have been defined by the administrator. For example, the initial context factory class for the JBoss application server is `org.jnp.interfaces.NamingContextFactory`.

Connection Factory:
Enter the name of the connection factory to use when connecting to the JMS provider. The name of the connection factory is vendor-specific. For example, the connection factory used for the JBoss application server is `org.jnp.interfaces:javax.jnp`.

Username:
If a user name is required to connect to this JMS provider, enter it in this field.

Password:
Enter the password for this user.

Custom Message Properties:
You can add JNDI context settings by clicking the **Add** button, and adding name and value properties in the fields.

When the JMS Service has been configured, you can configure the API Gateway to drop messages on to a queue or topic exposed by this service by selecting it from the **JMS Service** field on the **Messaging System** Connection filter dialog when configuring a policy. For more details, see the *Messaging System Filter* topic.

You can also configure JMS sessions for the newly added JMS Service at the Process level. For more details, see the next section.

Configuring a JMS Session

JMS services have JMS sessions, which can be shared by multiple JMS consumers, or used by a single JMS consumer only. To configure a JMS session, right-click the Process under the **Listeners** node in the Policy Studio tree, and select **Messaging System** -> **Add JMS Session**. Alternatively, you can configure a JMS session using the JMS Wizard.

Configure the following fields on the JMS Session screen:

Service
Select a pre-configured JMS Service from the drop-down list. For more details, see the section called “Configuring a JMS Service”.

Allow Duplicates
Typically, a JMS session operates in *auto* mode, meaning that messages received by the session are automatically acknowledged to guarantee once-only message delivery. By selecting the **Allow Duplicates** checkbox, you are configuring the JMS session to operate in *duplicates okay* mode. This means that messages are acknowledged lazily by the JMS session with the result that duplicate messages are possible. If messages are not automatically acknowledged, the client has no way of telling whether the JMS consumer received the message, and so may re-send the message. Running in this mode reduces the overhead associated with the guarantee of once-only message delivery offered by *auto* mode.

Listener Count
Specify the number of listeners permitted for this JMS session. Defaults to 1. If the volume of messages arriving at the queue is more than a single thread can process, you can increase the number of threads listening on the queue by increasing the listener count.

Configuring a JMS Consumer

You can configure multiple JMS consumers under a single JMS session, which share that session. Alternatively, you can configure a single JMS consumer per JMS session. Consumers sharing a JMS session access that session serially. Each consumer blocks until a response (if any) is received. Consumers with their own session do not encounter this problem, which may improve performance.

You can configure JMS consumers using the JMS Wizard, or by right-clicking an existing JMS session, and selecting **Add JMS Consumer**.

Configure the following fields on the JMS consumer screen:

Source:
Enter the name of the queue or topic from which you want to consume JMS messages.

Selector:
The entered expression specifies the messages that the consumer is interested in receiving. Using a selector, the task of

filtering the messages is performed by the JMS provider instead of by the consumer. The selector is a string that specifies an expression whose syntax is based on the SQL-92 conditional expression syntax. The Process only receives messages whose headers and properties match the selector.

Extraction Method:
Specify how to extract the data from the JMS message from the drop-down list:

- Create a `content.body` attribute based on the SOAP over JMS draft specification (the default)
- Insert the JMS message directly into the attribute named below
- Populate the attribute below with the value inferred from message type to Java

Attribute Name:
The name of the message attribute that holds the data extracted from the JMS message. Defaults to the `jms.message` message attribute.

Policy:
Select the appropriate policy to run on the JMS message after it has been consumed by the API Gateway.

Send Response to Configured Destination:
Specifies whether the API Gateway sends a reply to the response queue named in the incoming message (in the `ReplyTo` header). This option is selected by default. Deselecting this option means that the API Gateway never sends a reply to the response queue named in the `ReplyTo` header.

Configuring the JMS Wizard

You can use the **JMS Wizard** to configure a Process to consume JMS messages from a JMS queue or topic. When a message has been consumed by the API Gateway, it can be dispatched to a specified policy where the full compliment of message filters can run on the message. The message can then be routed onwards over HTTP or dropped back on to a JMS queue or topic.

To launch the **JMS Wizard**, right-click the Process under the **Listeners** node in the Policy Studio tree, and select **Messaging System** -> **JMS Wizard**. The wizard includes the following configuration screens:

JMS Service Provider
The first screen in the wizard enables you to configure connection details to the JMS provider that produces the JMS messages that are consumed by the API Gateway. For details on configuring the fields on this screen, see the section called “Configuring a JMS Service”.

JMS Session Configuration
The second screen in the wizard enables you to configure the **Allow Duplicates** option for the JMS session that is established with the JMS provider. For details on configuring this option, see the section called “Configuring a JMS Session”.

JMS Consumer Configuration
The third screen in the wizard enables you to configure JMS consumer settings. For details on configuring the fields on this screen, see the section called “Configuring a JMS Consumer”.

Remote Host Settings

Overview

You can use the **Remote Host Settings** to configure the way in which the API Gateway connects to a specific external server or routing destination. For example, typical use cases for configuring Remote Hosts with the API Gateway are as follows:

- Allowing the API Gateway to send HTTP 1.1 requests to a destination server when that server supports HTTP 1.1.
- Resolving inconsistencies in the way the destination server supports HTTP.
- Mapping a hostname to a specific IP address or addresses (for example, if a DNS server is unreliable or unavailable).
- Setting the timeout, session cache size, input/output buffer size, and other connection-specific settings for a destination server (for example, if the destination server is particularly slow, you can set a longer timeout).
- Stop accepting inbound connections on the HTTP Interface when the API Gateway loses connectivity to the remote host.

You can add Remote Hosts *per-process* by right-clicking the Process in the Policy Studio tree view, and selecting **Add Remote Host**. The tabs in the **Remote Host Settings** configuration screen are described in the next sections.

General Settings

You can configure the following settings on the **General** tab:

Host Name:
The host name or IP address of the Remote Host to connect to. If the host name entered in a **Static Router** filter matches this host name, the connection-specific settings configured on the **Remote Host** dialog are used when connecting to this host. This also includes any IP addresses listed on the **Addresses** tab, which override the default network DNS server mappings, if configured.

Port:
The TCP port on the Remote Host to connect to.

Maximum Connections:
The maximum number of connections to open to a Remote Host. If the maximum number of connections has already been established, the API Gateway Process waits for a connection to drop or become idle before making another request. The default maximum is 128 connections.

Allow HTTP 1.1:
The API Gateway uses HTTP 1.0 by default to send requests to a Remote Host. This prevents any anomalies if the destination server does not fully support HTTP 1.1. If the API Gateway is routing on to a Remote Host that fully supports HTTP 1.1, you can use this setting to enable the API Gateway to use HTTP 1.1.

Include Content Length in Request:
When this option is selected, the API Gateway includes the Content-Length HTTP header in all requests to this Remote Host.

Include Content Length in Response:
When this option is selected, if the API Gateway receives a response from this Remote Host that contains a Content-Length HTTP header, it returns this length to the client.

Send Server Name Indication TLS extension to server:
Adds a field to outbound TLS/SSL calls that shows the name that the client used to connect. For example, this can be useful if the server handles several different domains, and needs to present different certificates depending on the name the client used to connect.

Verify server's certificate matches requested hostname:
Ensures that the certificate presented by the server matches the name of the remote host being connected to. This prevents host spoofing and man-in-the-middle attacks. This setting is selected by default.

Address and Load Balancing Settings

You can configure the following settings on the **Addresses and Load Balancing** tab:

Addresses to use instead of DNS lookup:
You can add a list of IP addresses that the API Gateway uses instead of attempting a DNS lookup on the host name provided. This is useful in cases where a DNS server is not available or is unreliable. By default, connection attempts are made to the listed IP addresses on a round-robin basis.

For example, if a **Static Router** filter is configured to route to `www.webservice.com`, it first checks if any **Remote Hosts** have been configured with a **Host Name** entry matching `www.webservice.com`. If it finds a **Remote Host** with matching **Host Name**, it resolves the hostname to the IP addresses listed here. In addition, it uses all the connection-specific settings configured on the **Remote Host** dialog when routing messages to these IP addresses. If it can not find a matching host, the **Static Router** filter uses whatever DNS server has been configured for the network on which the API Gateway is running.

To add a list of IP addresses for a Remote Host, perform the following steps:

1. In the **Addresses to use instead of DNS lookup** box, select a priority group (for example, **Highest Priority**).
2. Click **Add**.
3. Enter an IP address or server name in the **Configure IP Address** dialog.
4. Click **OK**.
5. Repeat these steps to add more IP addresses as appropriate.

Load balancing:
The **Load Balancing Algorithm** drop-down box enables you to specify whether load balancing is performed on a simple round-robin basis or weighted by response time. **Simple Round Robin** is the default algorithm. Connection attempts are made to the listed IP addresses on a round-robin basis in each priority group. The **Weighted by response time** algorithm compares the request/reply response times for the server address in each priority group. This is the simplest way of estimating the relative load of the address. This algorithm works as follows:

1. The address with the least response time is selected to send the next message to.
2. If the address fails to send the message, it ignores that address for a period of time and selects another address in the same way.
3. If all addresses in a given group fail to accept a connection, addresses in the next group in ascending order of priority are used in the same way.
4. Only when all addresses in all priorities have failed to accept connections is delivery of the message abandoned, and an error raised.

The response times used by this algorithm decline over time. You can specify the rate of exponential decline by specifying a **Period to wait before response time is halved**. The default is 10,000 ms (10 sec). This enables addresses that were heavily loaded for a period of time to eventually resume accepting messages after the load subsides. For example, server A takes 100 ms to reply, and the other servers in the same priority group reply in 25 ms. A **Period to wait before response time is halved** of 10,000 ms (10 sec) means that after 20 seconds server A is retried along with the other servers. In this case, the response time has been halved twice (100 ms / 2 / 2 = 25 ms).

Advanced Settings

The options available on the **Advanced** tab are used when creating sockets for connecting to the Remote Host. Default values are provided for all fields, which should only be modified under advice from the Oracle Support Team (see *Oracle*

Contact Details).

You can configure the following configuration options on the **Advanced** tab:

Active Timeout:
When the API Gateway receives a large HTTP request, it reads the request off the network when it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, the API Gateway closes the connection. This prevents a Remote Host from closing the connection while sending data. Defaults to 30000 milliseconds (30 seconds). For example, the Remote Host's network connection is pulled out of the machine while sending data to the API Gateway. When the API Gateway has read all the available data off the network, it waits the **Active Timeout** period before closing the connection.

Idle Timeout:
The API Gateway supports HTTP 1.1 persistent connections. The **Idle Timeout** is the time that API Gateway waits after sending a message over a persistent connection to the Remote Host before it closes the connection. Defaults to 15000 milliseconds (15 seconds). Typically, the Remote Host tells the API Gateway that it wants to use a persistent connection. The API Gateway acknowledges this, and keeps the connection open for a specified period of time after sending the message to the host. If the connection is not reused by within the **Idle Timeout** period, the API Gateway closes the connection.

Input Buffer Size:
The maximum amount of memory allocated to each request.

Output Buffer Size:
The maximum amount of memory allocated to each response.

Cache Addresses For:
The period of time to cache addressing information after it has been received from the naming service (for example, DNS).

SSL Session Cache Size:
Specifies the size of the SSL session cache for connections to the remote host. This controls the number of idle SSL sessions that can be kept in memory. Defaults to 32. If there are more than 32 simultaneous SSL sessions, this does not prevent another SSL connection from being established, but means that no more SSL sessions are cached. A cache size of `0` means no cache, and no outbound SSL connections are cached.

Tip

You can use this setting to improve performance because it caches the slowest part of establishing the SSL connection. A new connection does not need to go through full authentication if it finds its target in the cache.

At `DEBUG` level or higher, the API Gateway outputs trace when an entry goes into the cache, for example:

```
DEBUG   09:09:12:953 [0d50] cache SSL session 11AA3894 to support.acme.com:443
```

If the cache is full, the output is as follows:

```
DEBUG   09:09:12:953 [0d50] enough cached SSL sessions 11AA3894 to
support.acme.com:443 already
```

Input Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can accept from peers. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured the **Default Settings** are used. You can override this setting at the Remote Host and HTTP interface levels. For more details, see the topic on *Compressed Content Encoding*.

Output Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can apply to outgoing messages. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured the **Default Settings** are used. You can override this setting at the Remote Host and HTTP interface levels. For more details, see the topic on *Compressed Content Encoding*.

Configuring Watchdogs

You can configure an HTTP Interface to shut down based on certain *conditions*. One such condition is dependent on the API Gateway being able to contact a particular back-end Web Service running on a Remote Host. To do this, you can configure an **HTTP Watchdog** for a Remote Host to poll the endpoint. If the endpoint cannot be reached, the HTTP Interface is shut down.

To configure the API Gateway to shut down an HTTP Interface based on the availability of a Remote Host, perform the following steps:

1. Configure an **HTTP Watchdog** for the Remote Host.
2. Configure a **Requires Endpoint** condition on the HTTP Interface.
3. When configuring this condition, select the Remote Host configured in step 1 (the host with the associated Watchdog).

Note

When **Load Balancing** is configured as **Weighted by response time**, and Remote Host Watchdogs are configured, the watch dog polling also contributes to the load balancing calculations.

For more information on adding a watchdog to a Remote Host, see *Configuring an HTTP Watchdog*. For more information on adding Conditions to an HTTP Interface, see *Configuring Conditions for HTTP Interfaces*.

Configuring an HTTP Watchdog

Overview

An HTTP Watchdog can be added to a Remote Host configuration in order to periodically poll the Remote Host to check its availability. The idea being that if the Remote Host becomes unavailable for some reason, a HTTP Interface can be brought down and will stop accepting requests. Once the Remote Host comes back online, the HTTP Interface will be automatically started up and will start accepting requests again.

To learn more about the reasons for shutting down an HTTP Interface if certain *conditions* do not hold, please refer to the help page on *Configuring Conditions for HTTP Interfaces*.

To configure an HTTP Watchdog, right-click a previously configured Remote Host in the tree view of the Policy Studio and select th **Watchdog** -> **Add** menu option. Configure the following sections on the **Configure HTTP Watchdog** dialog.

Configuration

Valid HTTP Response Code Ranges:
You can use this section to specify the HTTP response codes that you will regard as proof that the Remote Host is available. For example, if a 200 OK HTTP response is received for the poll request, the Remote Host can be considered available.

To specify a range of HTTP status codes, click the **Add** button and enter the **Start** and **End** of the range of HTTP response codes in the fields provided. An exact response code can be specified by entering the response code in both fields, e.g. "200".

HTTP Request for Polling:
The fields in this section allow you to configure the type and URI of the HTTP request to use to poll the Remote Host with. The default option is to use the *Options* HTTP command with a URI of "*", which is typically used to retrieve status information about the HTTP server.

If you wish to use an alternative HTTP request to poll the Remote Host, select an HTTP request method from the **Method** dropdown and then specify the URI to use in the **URI** field.

Remote Host Polling:
The settings in this section determine when and how the HTTP Watchdog polls the Remote Host. The **Poll Frequency** determines how often the Watchdog is to send the polling request to the Remote Host.

By default, the Watchdog uses "real" HTTP requests to the Remote Host to determine its availability. In other words, if the API Gateway is sending a batch of requests to the Remote Host it will use the response codes from these requests to decide whether or not the Remote Host is up. Therefore, the Watchdog effectively "polls" the Remote Host by sending real HTTP requests to it.

If you want to configure the Watchdog to send poll requests during periods when it is not sending requests to and receiving responses from the Remote Host, you should select the **Poll if up** checkbox. In this case the Watchdog will use "real" HTTP requests to poll the Remote Host as long as it is sending them, but will start sending "test" poll requests when it is not sending HTTP requests to the Remote Host in order to test its availability.

Important

When a Remote Host is deemed to be down (an "invalid" HTTP response code was received) the Watchdog will continue to poll it at the configured **Poll Frequency** until it comes back up again (until a "valid" HTTP response code is received).

Configuring Conditions for HTTP Interfaces

Overview

In certain cases, it may be desirable to pull down the HTTP Interface that accepts traffic for the API Gateway. For example, if the back-end Web Service is unavailable or if the physical interface on the machine loses connectivity to the network, it is possible to shut down the HTTP Interface so that it stops accepting requests.

A typical scenario where this functionality proves useful is as follows:

- A load balancer sits in front of several running instances of the API Gateway and round-robins requests between them all.
- A client sends SSL requests through the load balancer, which forwards them opaquely to one of the API Gateway instances.
- The API Gateway terminates the SSL connection, processes the message with the configured policy, and forwards the request on to the back-end Web Service.

In this deployment scenario, the load balancer does not want to keep sending requests to an instance of the API Gateway if it has either lost connectivity to the network or if the back-end Web Service is unavailable. If either of these *conditions* hold, the load balancer should stop attempting to route requests through this instance of the API Gateway and use the other instances instead.

So then, how can the load balancer determine the availability of the Web Service and also the connectivity of the machine hosting the API Gateway to the network on which the Web Service resides? Given that the request from the client to the API Gateway is over SSL, the load balancer has no way of decrypting the encrypted SSL data to determine whether or not a SOAP Fault, for example, has been returned from the API Gateway to indicate a connection failure.

The solution is to configure certain *conditions* for each HTTP Interface, which must hold in order for the HTTP Interface to remain available and accept requests. If any of the associated conditions fail, the Interface will be brought down and will not accept any more requests until the failed condition becomes true and the HTTP Interface is restarted. Once the load balancer receives a connection failure from the API Gateway (which it will when the HTTP Interface is down) it will stop sending requests to this API Gateway and will choose to round-robin requests amongst the other instances instead.

The following conditions can be configured on the HTTP Interface:

- *Requires Endpoint:*
  The HTTP Interface will remain up only if the Remote Host is available. The Remote Host is polled periodically to determine availability so that the HTTP Interface can be brought back up automatically when the Remote Host becomes available again.
- *Requires Link:*
  The HTTP Interface will remain up only if a named physical interface has connectivity to the network. As soon as a "down" physical interface regains connectivity, the HTTP Interface will automatically come back up again.

Conditions can be configured for an HTTP Interface by right clicking on the HTTP Interface (e.g. "*:8080") node under the Process node in the tree view of the Policy Studio. Select the **Add Condition** menu option and then either the **Requires Endpoint** or **Requires Link** option depending on your requirements. The sections below describe how to configure these conditions.

Requires Endpoint Condition

A **Requires Endpoint** Condition can be configured in cases where you only want to keep the HTTP Interface up if the back end Web Service (i.e. the Remote Host) is available. An HTTP Watchdog can be configured for the Remote Host, which is then responsible for polling the Remote Host periodically to ensure that the Web Service is available. Take a look at the *Remote Host Settings* and *Configuring an HTTP Watchdog* help pages for more information.

Remote Host:
The HTTP Interface will be shut down if the Remote Host selected here is deemed to be unavailable. The Remote Host can be continuously polled so that the Interface can be brought up again when the Remote Host becomes available again.

Requires Link Condition

The **Requires Link** Condition is used to bring down the HTTP Interface if a named physical network interface is no longer connected to the network. For example, if the cable is removed from the ethernet switch, the dependent HTTP Interface will be brought down immediately. The HTTP Interface will only start listening again once the physical interface is connected to the network again (i.e. when the ethernet cable is plugged back in).

Important

The **Requires Link** Condition is only available on Linux and Solaris platforms.

Interface Name:
The HTTP Interface will be brought down if the physical network interface named here is no longer connected to the network. On Unix platforms, physical network interfaces are usually named "eth0", "eth1", and so on. On Solaris machines, interfaces are named according to the vendor of the network card, for example, "bge0", "bge1", etc.

POP Client

Overview

The POP Client enables you to poll a Post Office Protocol (POP) mail server and read email messages from it. When the messages have been read, they can be passed into the core message pipeline where the full collection of message processing filters can act on them.

Configuration

You can configure a POP Client by right-clicking a Process node under the **Listeners** node in the Policy Studio tree, and selecting the **POP Client** -> **Add** menu option. Complete the following fields on the **POP Mail Server** dialog:

Server Name:
Enter the hostname or IP address of the POP mail server.

Port:
Enter the port on which the POP server is listening. By default, POP servers listen on port 110.

Connection Security:
Select the security used to connect to the POP server (`SSL`, `TLS`, or `NONE`). Defaults to `NONE`.

User Name:
Enter the user name of a configured mail user for this POP server.

Password:
Enter the password for this user.

Poll Rate:
Enter the rate at which the Process polls the mail server in milliseconds.

Delete Message from Server:
Specifies whether the POP server deletes email messages after they have been read by the Process. This setting is selected by default.

Email Debugging
Select this setting to find out more information about errors encountered by the API Gateway when polling the POP server. All trace files are written to the `/trace` directory of your the API Gateway installation. This setting is not selected by default.

Policy to Use:
Select the policy that you want to use to process messages that have been read from the POP server.

TIBCO Integration

Overview

The API Gateway ships with in-built support for TIBCO Enterprise Messaging System (EMS) and TIBCO Rendezvous enterprise-level products. The API Gateway can both produce and consume messages for both systems. This topic describes how to integrate with both TIBCO EMS and TIBCO Rendezvous. The reader is advised to follow the relevant links for more information on each of the steps involved.

TIBCO Rendezvous Integration

The API Gateway can act as both a producer and consumer of TIBCO Rendezvous messages. In both cases, a Rendezvous daemon must be configured, which is responsible for communicating with other Rendezvous programs on the network.

Producing TIBCO Rendezvous Messages:
The following steps must be configured in order to produce messages and send them to another Rendezvous program. Follow the links below to learn more about how to configure each of the steps involved.

- *TIBCO Rendezvous Daemon*
- *TIBCO Rendezvous Routing*

Consuming TIBCO Rendezvous Messages:
A TIBCO Rendezvous Listener can be configured at the API Gateway instance level in order to consume Rendezvous messages. The following steps must be followed in order to consume Rendezvous messages. Take a look at the help pages for the steps below for more complete information.

- *TIBCO Rendezvous Daemon*
- *TIBCO Rendezvous Listener*

TIBCO Enterprise Messaging Service Integration

The API Gateway can also be configured to produce and consume messages for TIBCO Enterprise Messaging Service (EMS) product. The first step in configuring either a producer or consumer or EMS messages is to configure a TIBCO Connection. The producer or consumer can then be configured.

Producing EMS Messages:
The following steps must be configured in order to send messages to an EMS Server. Follow the links below to learn more about how to configure each of the steps involved.

- *TIBCO Enterprise Messaging Service Connection*
- *TIBCO Enterprise Messaging Service Routing Filter*

Consuming TIBCO EMS Messages:
A TIBCO EMS Consumer can be configured at the API Gateway instance level to consume messages from a queue or topic on an EMS Server. The following steps must be followed to consume EMS messages. For more details, see the following topics:

- *TIBCO Enterprise Messaging Service Connection*
- *TIBCO Enterprise Messaging Service Consumer*

Cryptographic Acceleration

Overview

The API Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation, and SSL tunneling. OpenSSL exposes an *Engine API*, which makes it possible to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. When configured appropriately, OpenSSL calls the engine's implementation of these operations instead of its own.

For example, a particular engine may provide improved implementations of the asymmetric operations RSA and DSA. This engine can then be plugged into OpenSSL so that whenever OpenSSL needs to perform either an RSA or DSA operation, it calls out to the engine's implementation of these algorithms rather than call its own.

Typically, OpenSSL engines provide a hardware implementation of specific cryptographic operations. The hardware implementation usually offers improved performance over its software-based counterpart, which is known as *cryptographic acceleration*.

Cryptographic acceleration can be configured at the process level in the API Gateway. To configure the API Gateway process to use an OpenSSL engine instead of the default OpenSSL implementation, right-click the process in the tree-view in **Policy Studio**, and select the **Cryptographic Acceleration** -> **Add OpenSSL Engine**.

General Configuration

The **OpenSSL Engine Configuration dialog:**

The dialog displays the name of the engine, the algorithms that it implements, together with any initialization and cleanup commands required by the engine. Complete the following fields:

Name:
Enter an appropriate name for the engine in this field.

Provides:
Enter a comma-separated list of cryptographic operations to be performed by the engine instead of OpenSSL. The engine must implement the listed operations, otherwise the default OpenSSL operations are used. The following operations are available:

RSA	RSA (Rivest Shamir Adleman) asymmetric algorithm
DSA	DSA (Digital Signature Algorithm) asymmetric algorithm
RAND	Random number generation
DH	Diffie-Hellman anonymous key exchange algorithm
ALL	Engine's implementation of all cryptographic algorithms

For example, if you want to configure the API Gateway to use the engine's implementation of the RSA, DSA, and DH algorithms only, enter the following in the **Provides** field:

```
RSA, DSA, DH
```

Commands:
The OpenSSL engine framework allows a number of control commands to be invoked at various stages in the loading and unloading of a specific engine library. These commands can be issued before and/or after the initialization of the en-

gine, and also before and/or after the engine is un-initialized. Control commands are based on text name-value pairs.

Typical uses for control commands include specifying the path to a driver library, logging configuration information, a password to access a protected devices, a configuration file required by the engine, and so on.

OpenSSL control commands can be added by clicking the **Add** button. The **OpenSSL Engine Command**:

Enter the name of the command in the **Name** field, and its value in the **Value** field. This command *must* be supported by the engine.

Use the **When** drop-down list to select when the command is to be run. The options available are as follows:

preInit	Command is run before the engine is initialized (before the call to `ENGINE_init()`).
postInit	Command is run after the engine is initialized (after the call to `ENGINE_init()`).
preShutdown	Command is run before the engine shuts down (before the call to `ENGINE_finish()`).
postShutdown	Command is run after the engine shuts down (after the call to `ENGINE_finish()`.

Conversations for Crypto Engines

A Hardware Security Module (HSM) protects the private keys that it holds using a variety of mechanisms, including physical tokens, passphrases, and other methods. When use of the private key is required by an agent, it must authenticate itself with the HSM, and be authorized to access this data.

For information on how the API Gateway interacts with the HSM, see the *Cryptographic Acceleration Conversation: Request-Response* topic.

Cryptographic Acceleration Conversation: Request-Response

Conversations for Crypto Engines

Hardware Security Modules (HSM) protect the private keys they hold using a variety of mechanisms, including physical tokens, passphrases, and other methods. When use of the private key is required by some agent, it must authenticate itself with the HSM, and be authorized to access this data.

Whatever the mechanism protecting the keys on the HSM, this commonly requires some interaction with the agent. The most common form of interaction required is for the agent to present a passphrase. The intent is generally that this is carried out by a real person, rather than produced mechanically by the agent. Other forms of interaction may include prompting the operator to insert a specific card into a card reader.

However, the requirement for an operator to enter a passphrase renders automated startup of services using the HSM impossible. Although weaker from a security standpoint, the server can conduct an automated dialog with a HSM when it requires access to a private key, presenting specific responses to specific requests, including feeding passphrases to it. Of course, this is futile if the dialog calls for the insertion of a physical token in a device.

The dialogue for different keys on the same device is often the same. For example, a number of keys on an nCipher HSM may require the server to present an operator passphrase for a pre-inserted card in a card-reader. The specific dialogues are therefore associated with the cryptographic engine.

Each dialogue consists of a set of expected request-generated response pairs. The expected request takes the form of a regular expression. When the cryptographic device prompts for input, the text of this prompt is compared against each expected request in the conversation, until a match is found. When matched, the corresponding generated response is delivered to the HSM.

In the simplest case, consider a HSM producing the following prompt:

```
Enter passphrase for operator card Operator1:
```

You can identify this, for example, with the following regular expression:
`"passphrase.*Operator1"`

In the configured conversation, you can make the expected response to this prompt the passphrase for the specific card, for example:
`"tellNoOne"`

The server is somewhat at the mercy of the HSM for how this dialog continues. If the HSM continues to prompt for requests, the server can only attempt to respond. You may set the maximum expected challenge setting on the conversation to indicate a maximum number of prompts to expect from the HSM, at which point the server does its best to terminate the conversation, almost certainly failing to load the affected key.

TIBCO Rendezvous Daemon

Overview

TIBCO Rendezvous® is the leading low latency messaging product for real-time high throughput data distribution applications. A message can be sent from the TIBCO daemon running on the local machine to a single TIBCO daemon running on a separate host machine or it can be broadcast to several daemons running on multiple machines. Each message has a subject associated with it, which acts as the *destination* of the message.

A listener, which is itself a TIBCO daemon, can declare an interest in a subject on a specific daemon. Whenever a message is delivered to this subject on the daemon the message is delivered to the listening daemon.

The API Gateway can act as a listener on a specific subject at a TIBCO daemon, in which case it said to be acting as a *consumer* of TIBCO messages. Similarly, it can also send messages to a TIBCO daemon, effectively acting as a *producer* of messages. In both cases, the local TIBCO daemon must be configured to talk to the TIBCO daemons running on the remote machines.

For more information on consuming and producing messages to and from TIBCO Rendezvous, please refer to the following help pages:

- *TIBCO Integration*
- *TIBCO Rendezvous Listener*
- *TIBCO Rendezvous Routing*

The remainder of this page describes how to configure a TIBCO Rendezvous Daemon. For a more detailed description of how to configure the fields on this dialog please refer to your TIBCO Rendezvous documentation.

Configuration

You can configure **TIBCO Rendezvous Daemons** under the **External Connections** tree node in the Policy Studio. Right-click the **TIBCO Rendezvous Daemons node**, and select **Add a TIBCO Rendezvous Daemon**. Configure the following fields on the **TIBCO Daemon Settings** dialog:

Name:
Enter a friendly name for this TIBCO Rendezvous Daemon. When configured, this name is available for selection when configuring a **TIBCO Rendezvous Listener** and a **TIBCO Rendezvous Connection** filter.

Service:
Communication between TIBCO Rendezvous daemons takes place using Pragmatic General Multicast (PGM) or Universal Datagram Protocol (UDP) services. The specified *service parameter* configures the local TIBCO Rendezvous daemon to use this type of service when sending or broadcasting messages to other TIBCO Rendezvous daemons who are also using this service.

You can specify the service in the following ways:

- **By Service Name:**
  If your network administrator has added an entry for TIBCO Rendezvous in a network database such as NIS (for example, `rendezvous 7500/udp`), you can enter the name of the service (for example, `rendezvous`) in this field.
- **By Port Number:**
  Alternatively, you can enter the port number on which the TIBCO Rendezvous daemon is listening (for example, `7500`).
- **Default Option:**
  If you leave this field blank, a default service name of `rendezvous` is assumed. For this reason, administrators should add an entry in the network database with this name (for example, `rendezvous 7500/udp`. This enables you to leave this field blank so that this default service is used.

Network:
If the machine on which the TIBCO Rendezvous daemon is running has more than one network interface, you can specify what interface to use for all communications with other daemons. Each TIBCO Rendezvous daemon can only communicate on a single network, meaning that separate daemons must be configured for each network you want the daemon to communicate on.

For simplicity, you can leave this field blank, in which case the primary network interface is used for communication with other daemons. For more information on how to configure different networks and multicast groups, please see the TIBCO Rendezvous documentation.

Daemon:
The value entered here tells the API Gateway where it can find the TIBCO Rendezvous daemon, which is responsible for communicating with all other daemons on the network. This daemon can be local or remote.

For local daemons you need only specify the port number that the daemon is running on (for example `6500`). Alternatively, you can leave this field blank to connect to the daemon on the default port.

To connect to a remote daemon, you must specify both the host and port number of the daemon in this field (for example `daemon_host:6500`).

TIBCO Rendezvous Listener

Overview

TIBCO Rendezvous® is the leading low latency messaging product for real-time high throughput data distribution applications. A message can be sent from the TIBCO daemon running on the local machine to a single TIBCO daemon running on a separate host machine or it can be broadcast to several daemons running on multiple machines. Each message has a subject associated with it, which acts as the *destination* of the message.

A listener, which is itself a TIBCO daemon, can declare an interest in a subject on a specific daemon. Whenever a message is delivered to this subject on the daemon the message will be delivered to the listening daemon.

The API Gateway can act as a listener on a specific subject at a TIBCO daemon, in which case it said to be acting as a *consumer* of TIBCO messages. Similarly, it can also send messages to a TIBCO daemon, effectively acting as a *producer* of messages. For more information on how to send messages to other TIBCO Rendezvous programs, see the *TIBCO Rendezvous Routing* filter.

Configuration

A TIBCO Rendezvous Listener is configured at the API Gateway instance level in the Policy Studio. To add a listener, right-click the Oracle API Gateway instance in the tree view of the Policy Studio. Select the **TIBCO** -> **Rendezvous Listener** -> **Add** option from the context menu. Configure the following fields on the **TIBCO Rendezvous Listener** dialog:

TIBCO Settings Tab:
Enter the name of the subject that you want this consumer to listen for in the **Rendezvous Subject** field. Only messages addressed with this subject are consumed by the listener.

Click the button next to the **TIBCO Rendezvous Daemon to use** field, and select a previously configured TIBCO Rendezvous Daemon to communicate with other TIBCO programs. To add a TIBCO Rendezvous Daemon, right-click the **TIBCO Rendezvous Daemons** tree node, and select **Add a TIBCO Rendezvous Daemon**. For more details, see the *TIBCO Rendezvous Daemon* topic.

Policy to Use:
When messages with the specified subject have been consumed they must be passed into a policy where they can be processed accordingly. Select the policy that you want to use to process consumed messages from the tree.

TIBCO Enterprise Messaging Service Consumer

Overview

TIBCO Enterprise Messaging Service™ (EMS) provides a distributed message bus with native support for Java Messaging Service (JMS) and TIBCO Rendezvous, along with other protocols.

In general, TIBCO EMS clients *produce* messages and send them to the TIBCO EMS Server. Similarly, TIBCO EMS clients can connect to the TIBCO EMS Server and declare an interest in a particular queue or topic on that server. In doing so, it can *consume* messages that have been produced by another TIBCO EMS client.

The API Gateway can act as a message producer by sending messages to the TIBCO EMS Server and as a message consumer by listening on a queue or topic at the server. Both configurations require a connection to the TIBCO EMS Server. For more information on consuming and producing messages to and from TIBCO EMS, please refer to the following pages:

Configuration

TIBCO EMS Consumers are added at the API Gateway instance level in the Policy Studio. To add a consumer, right-click the **Oracle API Gateway** node under **Listeners** in the Policy Studio tree view. Select the **TIBCO** -> **Enterprise Messaging Consumer Service** -> **Add** options from the context menus. The following tabs and fields should be configured on the **TIBCO Enterprise Messaging Service Consumer** dialog.

Connection Tab:
Click the button on the right, and select a previously configured TIBCO EMS Connection for this consumer to connect to. To add a TIBCO EMS Connection, right-click the **TIBCO EMS Connections** tree node, and select **Add a TIBCO EMS Connection**. For more details see the *TIBCO Enterprise Messaging Service Connection* topic.

Settings Tab:
Configure the following fields on the **Settings** tab:

Destination Type:
Select whether this consumer will read messages off a queue or topic.

Queue/Topic Name:
Enter the name of the queue or topic here.

Selector:
Enter a filter to restrict the messages that are read off the queue or topic.

Do Not Receive Local Messages:
Check this option if you do not want to consume messages that have been produced by the API Gateway. For example, if you have configured a TIBCO EMS Routing filter to place messages on to a queue and have also configured a TIBCO EMS Consumer to read messages from the same queue, you can check this option to ensure that the consumer will ignore these locally generated messages.

Extraction Method:
The option selected here determines how the API Gateway will serialize the JMS message consumed from the queue or topic so that it can be passed into the policy selected on the **Policy** tab. The following options are available:

- *Create a content.body attribute based on the SOAP over JMS draft specification:*
  If this option is selected, messages are formatted according to the SOAP over JMS [http://www.w3.org/TR/soapjms/] recommendation, and stored in the `content.body` message attribute.
- *Insert the JMS message directly into the attribute named below:*
  Select this option to simply store the JMS message directly into the attribute specified in the **Attribute Name** field below.

- *Populate the attribute below with the value inferred from message type to Java:*
  Select this option if you wish to infer the data type of the JMS message from the underlying TIBCO EMS data type. In this case a TIBCO EMS TextMessage, BytesMessage, and MapMessage, will be converted into a java.lang.String, a byte[], and a java.lang.Map, respectively, while a JMS ObjectMessage will be deserialized into the attribute specified in the **Attribute Name** field below.

Attribute Name:
Once the message has been consumed it will be stored in the Oracle message attribute specified here. The **Extraction Method** selected above will determine how the raw JMS message is deserialized to the specified attribute. The consumed message can be processed at any stage hereafter in the policy selected on the **Policy** tab by accessing this attribute. By default the message is stored in the `ems.message` attribute.

Policy Tab:
Select a previously configured policy that you want to pass messages to after consuming the messages from the queue or topic configured on the **Settings** tab.

Oracle Security Service Module Settings (10g)

Overview

An Oracle Security Service Module (SSM) integrates a secured application (in this case, the API Gateway) with an Oracle Entitlements Server (OES) 10g so that security administration (for example, roles, resources, and policies) is delegated to the Oracle Entitlements Server 10g. An SSM must be installed on the machine hosting the application to be secured by the Oracle Entitlements Server 10g. The SSM runs in-process with the secured application, which improves performance and on-the-wire security.

In the Policy Studio, select the **Settings** node in the tree, and click the **Security Service Module** tab at the bottom of the screen. The **Security Service Module** settings enable you to configure the API Gateway to act as a Java SSM. For more details on Oracle Entitlements Server 10g and SSMs, see the Oracle Entitlements Server [http://www.oracle.com/technetwork/middleware/oes/overview/index.html] website.

Important

Oracle SSM is required only for integration with Oracle OES 10g. Oracle SSM is not required for integration with Oracle OES 11g. OES 10g was previously known as BEA AquaLogic Enterprise Security (ALES). Some items, such as schema objects, paths, and so on, may still use the ALES name.

Prerequisites

Before configuring the settings on the **Security Service Module** tab, you must perform the following prerequisite tasks:

Test the SSM Installation
Because the API Gateway is running a Java SSM internally, it is recommended that the example Java SSM client that ships with the OES installation is set up and configured. This example can be found in the following directory:
`/ales32-ssm/java-ssm/examples/JavaAPIExample`
Follow the instructions in the README file in this directory to test the installation. When the testing of the `JavaAPIExample` is complete, all the configuration files for an SSM instance are located in the `/ales32-ssm/java-ssm/SSM-Name` directory, where `SSM-Name` is the name of the SSM setup when testing the example.

Configure the API Gateway Classpath
The API Gateway classpath must be updated to include the JARs and configuration files for the SSM instance. The `jvm.xml` file must be updated so that various environment variables and the `SSM-Name` are updated to reflect the installation of the Java SSM. At minimum, the following must be updated in `jvm.xml`:

```
<Environment name="BEA_HOME" value="/opt/apps/bea" >
<Environment name="INSTANCE_NAME" value="SSM-Name" >
```

For example, to modify the classpath, place the following `jvm.xml` in the `conf` directory of the API Gateway installation:

```

<!--Additional JVM settings to run with Oracle Entitlements Server BEA_HOME must be set
to the location where the SSM is installed-->

  <ConfigurationFragment>
  <!-- Environment variables -->
  <!-- change these to match location where SSM has been installed and configured -->
  <Environment name="BEA_HOME" value="/opt/apps/bea" />
  <Environment name="ALES_SHARED_HOME" value="$BEA_HOME/ales32-shared" />

  <!-- Name of the SSM running in the API Gateway, replace the "SSM-Name" with the name of
  the SSM for the API Gateway -->
  <Environment name="INSTANCE_NAME" value="SSM-Name" />
  <Environment name="INSTANCE_HOME" value="$BEA_HOME/ales32-ssm/java-ssm/instance/
  $INSTANCE_NAME" />
  <Environment name="PDP_PROXY" value="$INSTANCE_HOME/pdpproxy" />

  <!-- Location of the Java SSM libraries -->
  <!-- <ClassDir name="$BEA_HOME" /> -->
  <ClassDir name="$BEA_HOME/ales32-ssm/java-ssm/lib" />
  <ClassDir name="$BEA_HOME/ales32-ssm/java-ssm/lib/providers/ales" />

  <!-- Add location of the SSM configuration to classpath  -->
  <ClassPath name="$INSTANCE_HOME/config/" />

  <!-- Additional JVM parameters based on the %JAVA-OPTIONS% of set-env script in SSM
  instance running in API Gateway $BEA_HOME/ales32-ssm/java-ssm/instance/ssm-name/config-->
  <VMArg name="-Dwles.scm.port=7005" />
  <VMArg name="-Dwles.arme.port=8000" />
  <VMArg name="-Dwles.config.signer=Oracle Entitlements Serverdemo.oracle.com" />
  <VMArg name="-Dlog4j.configuration=file:$INSTANCE_HOME/config/log4j.properties" />
  <VMArg name="-Dlog4j.ignoreTCL=true" />
  <VMArg name="-Dwles.ssl.passwordFile=$ALES_SHARED_HOME/keys/password.xml" />
  <VMArg name="-Dwles.ssl.passwordKeyFile=$ALES_SHARED_HOME/keys/password.key" />
  <VMArg name="-Dwles.ssl.identityKeyStore=$ALES_SHARED_HOME/keys/identity.jceks" />
  <VMArg name="-Dwles.ssl.identityKeyAlias=wles-ssm" />
  <VMArg name="-Dwles.ssl.identityKeyPasswordAlias=wles-ssm" />
  <VMArg name="-Dwles.ssl.trustedCAKeyStore=$ALES_SHARED_HOME/keys/trust.jks" />
  <VMArg name="-Dwles.ssl.trustedPeerKeyStore=$ALES_SHARED_HOME/keys/peer.jks" />
  <VMArg name="-Djava.io.tmpdir=$INSTANCE_HOME/work/jar_temp" />
  <VMArg name="-Darme.configuration=$INSTANCE_HOME/config/WLESarme.properties" />
  <VMArg name="-Dales.blm.home=$INSTANCE_HOME" />
  <VMArg name="-Dkodo.Log=log4j" />
  <VMArg name="-Dwles.scm.useSSL=true" />
  <VMArg name="-Dwles.providers.dir=$BEA_HOME/ales32-ssm/java-ssm/lib/providers"/>
  <VMArg name="-Dpdp.configuration.properties.location=$PDP_PROXY/
  PDPProxyConfiguration.properties"/>
 </ConfigurationFragment>
```

Centralize All Trace Output
Oracle’s Java SSM uses log4j to output any diagnostics. You can also add these messages to the API Gateway trace output by adding the log4j that ships with the API Gateway to the following file:

```
/ales32-ssm/java-ssm/SSM-NAME/conf/log4j.properties
```

Then the `log4j.rootCategory=WARN, A1, ASIlogFile` line includes a new appender called `VordelTrace` as follows:

```
log4j.rootCategory=WARN, A1, ASIlogFile, VordelTrace
```

Add the configuration for this new appender by adding the following line to the file:

```
log4j.appender.VordelTrace=com.vordel.trace.VordelTraceAppender
```

You can now start the API Gateway so that it runs with the Java SSM classpath and the centralized trace output.

Further Information
For more details on configuring and testing SSMs, see the *Oracle SSM Installation and Configuration Guide*.

Settings

On the **Security Service Module** settings screen, configure the following fields on the **Settings** tab:

Enable Oracle Security Service Module:
Select whether to enable the API Gateway process to act as an SSM. This setting is disabled by default.

Application Configuration Name:
Enter the Application Configuration name for the SSM instance. This is the name of your runtime application used by OES (for example, for monitoring purposes).

Configuration Name:
Enter the OES Configuration name for the SSM instance to be stored in the OES Configuration Repository. Configuration names share the same name as their Policy Domain names.

Application Configuration Properties:
Click **Add** to specify optional configuration properties as name-value pairs. Enter a **Name** and **Value** in the **Properties** dialog. Repeat to specify multiple properties.

Policy Domain Name:
Enter the OES Policy Domain name for the SSM instance. Policy Domains contain policy definitions (target resource, permission set, and policy). Policy Domain names share the same name as their Configuration names.

Name Authority Definition

Configure the following field on the **Name Authority Definition** tab:

Name Authority Definition File:
Click the **Browse** button at the bottom right to configure the Name Authority Definition file for the SSM. This is an XML file that specifies the naming authority definition required for the API Gateway. For example, a specified XML file named `apigatewayNameAuthorityDefinition.xml` file should contain the following settings:

```
<AuthorityConfig>
   <AuthorityDefinition name="apigatewayResource" delimiters="/\">
     <Attribute name="protocol" type="MULTI_TOKEN" authority="URLBASE" />
   </AuthorityDefinition>

   <AuthorityDefinition name="apigatewayAction" delimiters="/">
     <Attribute name="action" type="SINGLE_VALUE_TERMINAL"/>
   </AuthorityDefinition>
</AuthorityConfig>
```

Further Information

When you have configured the settings in the **Security Service Module** screen, you can use the following filters to integrate the API Gateway with Oracle Entitlements Server 10g:

- *Oracle Entitlements Server 10g Authorization*
- *Oracle Entitlements Server 10g Authorization*

Certificates and Keys

Overview

For the API Gateway to trust X.509 certificates issued by a specific Certificate Authority (CA), you must import that CA's certificate into the API Gateway's trusted Certificate Store. For example, if the API Gateway is to trust secure communications (SSL connections or XML Signature) from an external SAML Policy Decision Point (PDP), you must import the PDP's certificate, or the issuing CA's certificate into the API Gateway's Certificate Store.

In addition to importing CA certificates, you can also import and create server certificates and private keys in the Certificate Store. Finally, you can also import and create public-private key pairs. For example, these can be used with the Secure Shell (SSH) File Transfer Protocol (SFTP) or with Pretty Good Privacy (PGP).

Viewing Certificates and Private Keys

To view the lists of certificates and private keys stored in the Certificate Store, select **Certificates and Keys** -> **Certificates** in the tree on the left of the Policy Studio. The certificates and keys are listed on the following tabs in the **Certificates** screen on the right:

- **Certificates with Keys**: Server certificates with associated private keys.
- **Certificates**: Server certificates without any associated private keys.
- **CA**: Certification Authority certificates with associated public keys.

You can search for a specific certificate or key by entering a search string in the text box at the top of each tab, which automatically filters the tree.

Configuring an X.509 Certificate

To create a certificate and private key, click the **Create/Import** button. The **Configure Certificate and Private Key** dialog is displayed. This section explains how to use the **X.509 Certificate** tab on this dialog.

Creating a Certificate
Configure the following settings to create a certificate:

- **Subject:**
  Click the **Edit** button to configure the *Distinguished Name* (DName) of the subject.
- **Alias Name:**
  This mandatory field enables you specify a friendly name (or alias) for the certificate. Alternatively, you can click **Use Subject** button to add the DName of the certificate in the text box instead of a certificate alias.
- **Public Key:**
  Click the **Import** button to import the subject's public key (usually from a PEM or DER-encoded file).
- **Version:**
  This read-only field displays the X.509 version of the certificate.
- **Issuer:**
  This read-only field displays the distinguished name of the CA that issued the certificate.
- **Choose Issuer Certificate:**
  Select this setting if you wish to explicitly specify an issuer certificate for this certificate (for example, to avoid a potential clash or expiry issue with another certificate using the same intermediary certificate). You can then click the button on the right to select an issuer certificate. This setting is not selected by default.
- **Validity Period:**
  The dates specified here define the validity period of the certificate.
- **Sign Certificate:**
  You must click this button to sign the certificate. The certificate can be self-signed, or signed by the private key be-

longing to a trusted CA whose key pair is stored in the **Certificate Store**.

Importing Certificates
You can use the following buttons to import or export certificates into the Certificate Store:

- **Import Certificate:**
  Click this button to import a certificate (for example, from a `.pem` or `.der` file).
- **Export Certificate:**
  Use this option to export the certificate (for example, to a `.pem` or `.der` file).

Configuring a Private Key

Use the **Private Key** tab to configure details of the private key. By default, private keys are stored locally in the Certificate Store. They can also be stored on a Hardware Security Module (HSM), if required.

Private Key Stored Locally:
Select the **Private key stored locally** radio button. The following configuration options are available for keys that are stored locally in the Certificate Store:

- **Private Key:**
  This read-only field displays details of the private key.
- **Import Private Key:**
  Click the **Import Private Key** button to import the subject's private key (usually from a PEM or DER-encoded file).
- **Export Private Key:**
  Click this button to export the subject's private key to a PEM or DER-encoded file.

Private key stored on HSM:
If the private key that corresponds to the public key stored in the certificate resides on a HSM, select the **Private key stored on HSM** radio button. Configure the following fields to associate a key stored on a HSM with the current certificate:

- **Engine Name:**
  Enter the name of the OpenSSL Engine to use to interface to the HSM. All vendor implementations of the OpenSSL Engine API are identified by a unique name. Please refer to your vendor's HSM or OpenSSL Engine implementation documentation to find out the name of the engine.
- **Key Id:**
  The value entered is used to uniquely identify a specific private key from all others that may be stored on the HSM. On completion of the dialog, this private key is associated with the certificate that you are currently editing. Private keys are identified by their key Id by default.
- **Use Public Key:**
  Select this option if the HSM allows identifying a specific private key based on its associated public key, instead of using the private key Id. This option is not selected by default.
- **Conversation:**
  If the HSM requires the server to provide a specific response to a specific request from the HSM, you can enter the response in this field. This enables the server to conduct an automated dialog with a HSM when it requires access to a private key. For example, in a simple case, the server response might be a specific passphrase. For more details, see the topic on *Cryptographic Acceleration Conversation: Request-Response*.

Global Options

The following global configuration options apply to both the **X.509 Certificate** and **Private Key** tabs:

- **Import Certificate + Key:**

Use this option to import a certificate and a key (for example, from a `.p12` file).

- **Export Certificate + Key:**
  Use this option to export a certificate and a key (for example, to a `.p12` file).

Click **OK** when you have finished configuring the certificate and/or private key.

Managing Certificates and Keystores

On the main **Certificates** screen, you can click the **Edit** button to edit an existing certificate. You can also click the **View** button to view the more detailed information on an existing certificate. Similarly, you can click the **Remove** button to remove a certificate from the Certificate Store.

Java Keystore
You can also export a certificate to a Java keystore. You can do this by clicking the **Keystore** button on the main **Certificates** screen. Click the browse button at beside the **Keystore** field at the top right to open an existing keystore, or click **New Keystore** to create a new keystore. Choose the name and location of the keystore file, and enter a passphrase for this keystore when prompted. Click the **Export to Keystore** button and select a certificate to export.

Similarly, you can import certificates and keys from a Java keystore into the Certificate Store. To do this, click the **Keystore** button on the main **Certificates** screen. On the **Keystore** screen, browse to the location of the keystore by clicking the button beside the **Keystore** field. The certificates/keys in the keystore are listed in the table. To import any of these keys to the Certificate Store, select the box next to the certificate or key that you want to import, and click the **Import to Trusted Certificate Store** button. If the key is protected by a password, you are prompted for this password.

You can also use the **Keystore** screen to view and remove existing entries in the keystore. You can also add keys to the keystore and to create a new keystore. Use the appropriate button to perform any of these tasks.

Configuring Key Pairs

To configure public-private key pairs in the Certificate Store, select **Certificates and Keys** -> **Key Pairs**. The **Key Pairs** screen enables you to add, edit, or delete OpenSSH public-private key pairs, which are required for the Secure Shell (SSH) File Transfer Protocol (SFTP).

Adding a Key Pair
To add a public-private key pair, click the **Add** button on the right, and configure the following settings in the dialog:

- **Alias:**
  Enter a unique name for the key pair.
- **Algorithm:**
  Enter the algorithm used to generate the key pair. Defaults to RSA.
- **Load:**
  Click the **Load** buttons to select the public key and/or private key files to use. The **Fingerprint** field is auto-populated when you load a public key.

Note

The keys must be OpenSSH keys. RSA keys are supported, but DSA keys are not supported. The keys must not be passphrase protected.

Managing OpenSSH Keys
You can use the `ssh-keygen` command provided on UNIX to manage OpenSSH keys. For example:

- The following command creates an OpenSSH key:

```
ssh-keygen -t rsa
```

- The following command converts an `ssh.com` key to an OpenSSH key:
  `ssh-keygen -i -f ssh.com.key > open.ssh.key`
- The following command removes a passphrase (enter the old passphrase, and enter nothing for the new passphrase):
  `ssh-keygen -p`
- The following command outputs the key fingerprint:
  `ssh-keygen -lf ssh_host_rsa_key.pub`

Editing a Key Pair

To edit a public-private key pair, select a key pair alias in the table, and click the **Edit** button on the right. For example, you can load a different public key and/or private key. Alternatively, double-click a key pair alias in the table to edit it.

Deleting Key Pairs

You can delete a selected key pair from the Certificate Store by clicking the **Remove** button on the right. Alternatively, click the **Remove All** button.

Configuring PGP Key Pairs

To configure Pretty Good Privacy (PGP) key pairs in the Certificate Store, select **Certificates and Keys** -> **PGP Key Pairs**. The **PGP Key Pairs** screen enables you to add, edit, or delete PGP public-private key pairs.

Adding a PGP Key Pair

To add a PGP public-private key pair, click the **Add** button on the right, and configure the following settings in the dialog:

- **Alias:**
  Enter a unique name for the PGP key pair.
- **Load:**
  Click the **Load** buttons to select the public key and/or private key files to use.

Note

The PGP keys added must not be passphrase protected.

Managing PGP Keys

You can use the freely available GNU Privacy Guard (GnuPG) tool to manage PGP key files (available from http://www.gnupg.org/). For example:

- The following command creates a PGP key:
  `gpg --gen-key`
  For more details, see http://www.seas.upenn.edu/cets/answers/pgp_keys.html [http://www.seas.upenn.edu/cets/answers/pgp_keys.html]
- The following command enables you to view the PGP key:
  `gpg -a --export`
- The following command exports a public key to a file:
  `gpg --export -u 'UserName' -a -o public.key`
- The following command exports a private key to a file:
  `gpg --export-secret-keys -u 'UserName' -a -o private.key`
- The following command lists the private keys:
  `gpg --list-secret-keys`

Editing a PGP Key Pair

To edit a PGP key pair, select a key pair alias in the table, and click the **Edit** button on the right. For example, you can load a different public key and/or private key. Alternatively, double-click a key pair alias in the table to edit it.

Deleting PGP Key Pairs

You can delete a selected PGP key pair from the Certificate Store by clicking the **Remove** button on the right. Alternatively, click the **Remove All** button.

API Gateway Users

Overview

The API Gateway User Store contains the configuration data for managing API Gateway user information. This topic introduces the concepts of API Gateway Users, Groups, and Attributes. It explains how to manage these components on the **Users** screen in the Policy Studio.

Note

API Gateway Users provide access to the messages and services protected by the API Gateway. Whereas Admin Users provide access to the API Gateway configuration management features available in the Policy Studio and API Gateway Manager. For more details, see the *Managing Admin Users* topic.

Users

API Gateway Users specify the user identity in the User Store. This includes details such as the user name, password, and X.509 certificate. API Gateway Users must be a member of at least one User Group. In addition, Users can specify optional Attributes, and inherit Attributes at the Group level.

To view all existing Users, select the **Users and Groups** -> **Users** node in the Policy Studio tree. The Users are listed in the table on the main panel of the Policy Studio. You can find a specific User by entering a search string in the **Filter** field.

Adding Users

You can create API Gateway Users on the **Users** page in the Policy Studio. Click the **Add** button on the right to view the **Add User** dialog.

Adding User Details
To specify the new user details, complete the following fields on the **General** tab:

- **User Name**
  Enter a name for the new user.
- **Password**
  Enter a password for the new user.
- **Confirm Password**
  Re-enter the user's password to confirm.
- **X.509 Cert**
  Click the **X.509 Cert** button to load the user's certificate from the **Certificate Store**.

Adding User Attributes
You can specify optional User Attributes on the **Attributes** tab, which is explained in the next section.

Attributes

You can specify Attributes at the User level and at the Group level on the **Attributes** tab. Attributes specify user configuration data (for example, attributes used to generate SAML attribute assertions).

Adding Attributes
The **Attributes** tab enables you to configure user attributes as simple name-value pairs. The following are examples of user attributes:

- `role=admin`
- `email=niall@oracle.com`
- `dept=eng`
- `company=oracle`

You can add user attributes by clicking the **Add** button. Enter the attribute name, type, and value in the fields provided. The `Encrypted` type refers to a string value that is encrypted using a well-known encryption algorithm or cipher.

Groups

API Gateway User Groups are containers that encapsulate one or more Users. You can specify Attributes at the Group level, which are inherited by all Group members. If a User is a member of more than one Group, that User inherits Attributes from all Groups (the superset of Attributes across the Groups of which the User is a member).

To view all existing Groups, select the **Users and Groups** -> **Groups** node in the Policy Studio tree. The User Groups are listed in the table on the main panel of the Policy Studio. You can find a specific Group by entering a search string the **Filter** field.

Adding Groups

You can create User Groups on the **Groups** page in the Policy Studio. Click the **Add** button on the right to view the **Add Group** dialog.

Adding Group Details

To specify the new group details, complete the following fields on the **General** tab:

- **Group Name**
  Enter a name for the new group.
- **Members**
  Click the **Add** button to display the **Add Group Member** dialog, and select the members to add to the group.

Adding Group Attributes

You can specify optional Attributes at the Group level on the **Attributes** tab. For more details, see the Attributes section.

Updating Users or Groups

To edit details for a specific User or Group, select it in the list, and click the **Edit** button on the right. Enter the updated details in the **Edit User** or **Edit Group** dialog.

To delete a specific User or Group, select it in the list, and click the **Remove** button on the right. Alternatively, to delete all Users or Groups, click the **Remove All** button. You are prompted to confirm all deletions.

Global Schema Cache

Overview

The Schema Cache contains XML Schemas that can be used globally by **Schema Validation** filters. You can import XML Schemas from XML Schema files or from WSDL files. WSDL files often contain XML Schemas that define the elements that appear in SOAP messages. To facilitate this, the Policy Studio can import WSDL files from the file system, from a URL, or from a UDDI registry.

When the XML Schema has been imported into the cache and selected in a **Schema Validation** filter, the API Gateway can retrieve the schema from the cache instead of fetching it from its original location. This improves the runtime performance of the filter, and also ensures that an administrator has complete control over the schemas used to validate messages.

In the Policy Studio navigation tree, you can access the global Schema Cache by selecting **Resources** -> **XML Schemas**. The list of schemas present in the Schema Cache is shown in the tree. You can view or edit the contents of any of these schemas by clicking the schema node. The schema contents are displayed in the tab on the right.

At any point, you can manually modify the contents of the schema in the tab on the right. To save the modified contents to the cache, right-click, and elect **Save**.

Adding Schemas to the Cache

To add an XML Schema to the cache, right-click the **Schemas** node in the tree, and select **Add Schema**. Alternatively, click the **Add Schema** link at the top of the **Schemas** screen on the right. The **Load Schema** dialog enables you to load a schema from an XML Schema file directly or from a WSDL file.

Select the **From XML Schema** radio button to load the schema directly from a schema file, and click **Next**. On the next screen, enter or browse to the location of the schema file using the field provided. You can also enter a full URL to pull the schema from a web location. Click the **Finish** button to import the schema into the cache. Alternatively, if you wish to load the schema file from a WSDL file, select the **From WSDL** radio button on the **Select Schema Source** screen, and click **Next**.

The WSDL file can be located from the file system, from a URL, or from a UDDI registry. Select the appropriate option and enter or browse to the location of the WSDL file in the fields provided. If you wish to retrieve the WSDL file from a UDDI registry, click the **WSDL from UDDI** radio button, and click the **Browse UDDI** button. The **Browse UDDI Server for WSDL** dialog enables you to connect to a UDDI and search it for a particular WSDL file. For more information on how to configure this dialog, see the *Retrieving WSDL Files from a UDDI Registry* topic.

Testing WSDL Files for WSI Compliance

Before loading the schema from a WSDL file, you can check the WSDL file for compliance with the WS-I Basic Profile. The Basic Profile consists of a set of assertions and guidelines on how to ensure maximum interoperability between different implementations of Web Services. For example, there are recommendations on the SOAP style to use (`document/literal`), how schema information is included in WSDL files, and how message parts are defined to avoid ambiguity for consumers of WSDL files.

The Policy Studio uses the Java version of the WS-Interoperability Testing Tools to test imported WSDL files for compliance with the recommendations in the Basic Profile. A report is generated showing which recommendations have passed and which have failed. While you can still import a WSDL file that does not comply with the Basic Profile, there is no certainty that consumers of the Web Service can use it without encountering problems.

Important

Before you run the WS-I compliance test, you must ensure that the Java version of the Interoperability Testing Tools is installed on the machine on which the Policy Studio is running. You can download these tools from www.ws-i.org [http://www.ws-i.org].

To configure the location of the WS-I testing tools, select **Window** -> **Preferences** from the Policy Studio main menu. In the **Preferences** dialog, select the **WS-I Settings**, and browse to the location of the WS-I testing tools. You must specify the *full path* to these tools (for example, `C:\Program Files\WSI_Test_Java_Final_1.1\wsi-test-tools`). For more details on configuring WS-I settings, see the *Policy Studio Preferences* topic.

Running the WS-I Compliance Test
To run the WS-I compliance test on a WSDL file, perform the following steps:

1. Select **Tools** -> **Run WS-I Compliance Test** from the Policy Studio main menu.
2. In the **Run WS-I Compliance Test** dialog, browse to the **WSDL File** or specify the **WSDL URL**.
3. Click **OK**. The WS-I Analysis tools run in the background in Policy Studio.

The results of the compliance test are displayed in your browser in a **WS-I Profile Conformance Report**. The overall result of the compliance test is displayed in the **Summary** section. The results of the WS-I compliance tests are grouped by type in the **Artifact: description** section. For example, you can access details for a specific port type, operation, or message by clicking the appropriate link in the **Entry List** table. Each **Entry** displays the results for the relevant WS-I Test Assertions.

Organizing Schemas with Schema Containers

If you intend to add large numbers of schemas to validate different types of requests, it makes sense to organize these types of schemas into different groups. For example, if you have a set of schemas that defines types used in requests for a StockQuote Web Service and another set of schemas used to validate requests for a PurchaseOrder Web Service, it makes sense to organize each set of schemas into separate groups (for example, StockQuote Schemas and PurchaseOrder Schemas).

The Policy Studio enables you to add *Schema Containers* for this purpose. To add a Schema Container, right-click the **Schemas** tree node, and select the **Add Schema Container** menu option. Enter a descriptive name for the container in the field provided on the **Schema Container** dialog.

You can add related schemas under this container by right-clicking the container, and selecting the **Add Schema** menu option. You can then load the schema directly from an XML Schema file, or indirectly from a WSDL file in the usual manner.

Furthermore, you can create containers within containers to further organize your schemas. Right-click an existing container, and select the **Add Schema Container** menu option.

A useful feature of Schema Containers is the ability to copy and paste schemas from one container to another. For example, you can use this to copy schemas to a **Test Schemas** container where you can modify them and test them against incoming requests. To do this, right-click a schema, and select the **Copy** menu option. To copy the schema to another container, right-click the destination container and select the **Paste** option.

Important

Note the following:

- Only one schema is allowed per target namespace in any one container. This is because schemas are keyed in the cache using the `targetNamespace` in the schema. Therefore, all elements referenced by the imported schema must be in the same container as the schema. The Policy Studio automatically stores imported and included schemas in the same container as the top-level schema. However, deleting schemas from a container that contains elements referenced by other schemas in the container prevents the set of schemas from successfully validating incoming requests.

If a schema has no `targetNamespace` defined, it is keyed using the full path to the file on the file system.

- If you add a schema that *includes* other schemas (using the `<include>` element) to the Schema Cache, the included schemas are added inline to the top-level parent schema. Because schemas are keyed by target namespace, and because all included schemas *must* belong to the same namespace as the parent schema, it makes sense to inline the included schemas.

Schema Validation

The **Schema Validation** filter is used to validate XML messages against schemas stored in the cache or in the **Web Services Repository**. This filter is found in the **Content Filtering** category of filters in the Policy Studio. For more information on configuring this filter, see the *Schema Validation* topic.

External Connections

Overview

The API Gateway can leverage your existing Identity Management infrastructure, thus avoiding the need to maintain separate silos of user information. For example, if you already have a database full of user credentials, the API Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, the API Gateway can authorize users, lookup user attributes, and validate certificates against third-party Identity Management servers.

You can add a connection to an external system as a global *External Connection* in the Policy Studio so that it can be re-used across all filters and policies. For example, if you create a policy that authenticates users against an LDAP directory, and then validates an XML signature by retrieving a public key from the same LDAP directory, it makes sense to create a global External Connection for that LDAP directory. You can then *select* the LDAP Connection in both the authentication and XML signature verification filters, rather than having to reconfigure them in both filters.

You can also use External Connections in cases where you want to configure a group of related URLs. This is most useful when you want to round-robin between a number of related URLs to ensure high availability. When the API Gateway is configured to use a *URL Connection Set* (instead of a single URL), it round-robins between the URLs in the set.

You can configure External Connections by right-clicking the appropriate node (for example, **Database Connections**) under the **External Connections** node in the Policy Studio tree. This topic introduces the different types of External Connection and shows where to obtain more details.

Authentication Repository Profiles

The API Gateway can authenticate users against external databases and LDAP repositories, in addition to its own local user store. You can also use a number of bespoke authentication connectors to enable the API Gateway to authenticate against specific third-party Identity Management products.

Connection details for these authentication repositories are configured at a global level, making them available for use across authentication (and authorization) filters. This saves the administrator from reconfiguring connection details in each filter.

For example, the available authentication repository types include the following:

- CA SiteMinder Repositories
- Database Repositories
- Entrust GetAccess Repositories
- LDAP Repositories
- Local Repositories (for example, Local User Store)
- Oracle Access Manager Repositories
- Oracle Entitlements Server Repositories
- RADIUS Repositories
- RSA Access Manager Repositories
- Tivoli Repositories

For details how to configure the various authentication repository types, see the *Authentication Repository* topic.

Connection Sets

Connection Sets are used by the API Gateway to round-robin between groups of external servers (for example, RSA Access Manager). You can use reuse these global groups when configuring connections to external servers in the Policy Studio. For this reason, Connection Sets are available under the **External Connections** node according to the filter from which they are available. For example, Connection Sets under the **RSA ClearTrust Connection Sets** node are available in the **RSA Access Manager** filter.

At runtime, the API Gateway can round-robin between the servers in the group to ensure that if one of the servers becomes unavailable, the API Gateway can use one of the other servers in the group.

To add a Connection Set for a particular category of filters, right-click the appropriate node under the **Connection Sets** node under the **External Connections** node. Select **Add a Connection Set** to display the **Connection Group** dialog. For more details, see the *Configuring Connection Groups* topic.

Database Connections

The API Gateway typically connects to databases to authenticate or authorize users using the API Gateway's numerous Authentication and Authorization filters. Similarly, the API Gateway can retrieve user attributes from a database (for example, which can then be used to generate SAML attribute assertions later in the policy). You can configure database connections globally under the **External Connections** node, making them available to the various filters that require a database connection. This means that an administrator can reuse the same database connection details across multiple authentication, authorization, and attribute-based filters.

The API Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented using *Jakarta DBCP (Database Connection Pools)*. The settings in the **Advanced** section of the **Configure Database Connection** dialog are used internally by the API Gateway to initialize the connection pool. The table at the end of this section shows how the fields correspond to specific configuration DBCP settings.

To configure details for a global database connection, right-click the **External Connections** -> **Database Connections** node. Select the **Add a Database Connection** menu option, and configure the fields on the **Configure Database Connection** dialog. For details on configuring these fields, see the *Database Connection* topic.

ICAP Servers

The Internet Content Adaptation Protocol (ICAP) is a lightweight HTTP-based protocol used to optimize proxy servers, which frees up resources and standardizes how features are implemented. For example, ICAP is typically used to implement features such as virus scanning, content filtering, ad insertion, or language translation in the HTTP proxy cache.

When an ICAP Server is configured under the **External Connections** node, you can then select it in multiple ICAP filters. For details on how to configure an ICAP Server, see the *Configuring ICAP Servers* topic.

JMS Services

The Java Message Service (JMS) is a Java message-oriented middleware API for sending messages between two or more clients. When a JMS Service is configured under the **External Connections** node, it is available for selection in multiple JMS-related configuration screens. This enables you to share JMS configuration across multiple filters.

For more details on configuring JMS services, see the *Messaging System* topic.

Kerberos Connections

You can configure global **Kerberos Clients**, **Kerberos Services**, and **Kerberos Principals** under the **External Connections** node. When a Kerberos item is configured, it is available for selection in all Kerberos-related configuration screens that require this item. This enables you to share Kerberos configuration items across multiple filters.

For more details, see the following topics:

- *Kerberos Clients*
- *Kerberos Services*
- *Kerberos Principals*

LDAP Connections

In the same way that database connections can be configured globally in the Policy Studio (and then reused across individual filters), LDAP connections are also managed globally in the Policy Studio. LDAP connections are used by authentication, authorization, and attribute filters. Filters that require a public key (from a public-private key pair) can also retrieve the key from an LDAP source.

When a filter that uses an LDAP directory is run for the first time, it binds to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually the connection details include the username and password of an administrator user who has read access to all users in the LDAP directory for whom you wish to retrieve attributes or authenticate.

For details on how to configure a global LDAP connection, see the topic on *Configuring LDAP Directories*.

OCSP Connections

The API Gateway can use OCSP (Online Certificate Status Protocol) to validate a certificate against an OCSP responder or group of responders. OCSP requests for certificate validation can be signed by a key from the **Certificate Store**, and the response from the OCSP responder can be optionally validated.

An OCSP Connection typically comprises a *group* of OCSP Responder URLs, together with options to sign OCSP requests and validate OCSP responses. To configure a global OCSP Connection, right-click the **OCSP Connection** node under the **External Connections** node. Select the **Add an OCSP Connection** option to display the **Certificate Validation - OCSP** dialog. For more details, see the *OCSP Certificate Validation* topic.

Note

When an OCSP Connection is added in this manner, it is available for selection in the **OCSP Certificate Validation** filter, which is found under the **Certificate** category of filters in the Policy Studio.

Proxy Servers

You can configure proxy servers under the **External Connections** node, which can then be specified in the **Connection** and **Connect To URL** filters. When configured, the filter connects to the proxy server, which routes the message to the destination server.

To configure a proxy server, click the **External Connections** node, and select **Proxy Servers** -> **Add a Proxy Server**. For details on how to configure the settings the **Proxy Server Settings** dialog, see the *Proxy Servers* topic.

RADIUS Clients

The Remote Authentication Dial In User Service (RADIUS) protocol provides centralized authentication and authorization for clients connecting to remote services.

To configure a client connection to a remote server over the RADIUS protocol, click the **External Connections** node, and select **RADIUS Clients** -> **Add a RADIUS Client**. For details on how to configure the settings the **RADIUS Client** dialog, see the *RADIUS Clients* topic.

For details on how to configure a RADIUS Authentication Repository, see the *Authentication Repository* topic.

SiteMinder

To add a CA SiteMinder connection, right-click the **SiteMinder/SOA Security Manager** node under the **External Connections** node, and select **Add a SiteMinder Connection** to display the **SiteMinder Connection Details** dialog. For details on configuring the fields on this dialog, see the *SiteMinder/SOA Security Manager Connection* topic.

SMTP Servers

You can configure a Simple Mail Transfer Protocol (SMTP) server as a global configuration item under the **External Connections** node. The **SMTP** filter in the **Routing** category can then reference this SMTP server. To configure an SMTP server, right-click the **External Connections** -> **SMTP Servers** node, and select **Add an SMTP Server**. For more details, see the topic on *SMTP Servers*.

SOA Security Manager

To add a CA SiteMinder connection, right-click the **External Connections** -> **SiteMinder/SOA Security Manager** node, and select **Add a SOA Security Manager Connection** to display the **SOA Security Manager Connection Details** dialog. For details on configuring the fields on this dialog, see the *SiteMinder/SOA Security Manager Connection* topic.

Syslog Servers

You can configure Syslog Servers globally, and then select them as a customized logging destination for a Process. Right-click the **External Connections** -> **Syslog Servers** node, and select the **Add a Syslog Server** menu option. Complete the following fields on the **Syslog Server** dialog:

Name:
Enter a name for the Syslog server.

Host:
Specify the host on which the Syslog daemon is running.

Facility:
Select the Syslog facility that you want to log to.

For details on how to configure the API Gateway process to log to this remote Syslog Server, see the topic on *Audit Log Settings*.

TIBCO

You can add connections to the following TIBCO products:

- TIBCO Enterprise Messaging Service (EMS)
- TIBCO Rendezvous Daemon

To add a TIBCO EMS connection, right-click the **External Connections** -> **TIBCO EMS Connection** node in the tree, and select **Add a TIBCO EMS Connection**. For details on configuring the fields on the dialog, see the *TIBCO Enterprise Messaging Service Connection* topic.

To add a TIBCO Rendezvous Daemon, right-click the **External Connections** -> **TIBCO Rendezvous Daemon** node in the tree, and select **Add a TIBCO Rendezvous Daemon**. For details on configuring the fields on the dialog, see the *TIBCO Rendezvous Daemon* topic.

Tivoli

You can create a connection to an IBM Tivoli server to enable integration between the API Gateway and Tivoli Access Manager. Tivoli connections can then be used by the API Gateway's Tivoli filter to delegate authentication and authorization decisions to Tivoli Access Manager, and to leverage existing Tivoli Access Manager policies.

To add a Tivoli connection, right-click the **External Connections** -> **Tivoli Connections** node in the tree, and select **Add a Tivoli Connection**. For details on configuring the fields on the **Tivoli Configuration** dialog, see the *Tivoli Integration* topic.

URL Connection Sets

URL Connection Sets are used by API Gateway filters to round-robin between groups of external servers (for example, Entrust GetAccess, OCSP, SAML PDP, or XKMS). These global groups can then be reused when configuring these filters in the Policy Studio. For this reason, URL Connection Sets are available under the **External Connections** node in the tree, according to the filters from which they are available. For example, URL sets under the **OCSP URL Sets** node are available in the **OCS Certificate Validation** filter, while sets under the **XKMS URL Sets** are only available from the **XKMS Certificate Validation** filter.

At runtime, the API Gateway can round-robin between the servers in the group to ensure that if one of the servers becomes unavailable, the API Gateway can use one of the other servers in the group.

To add a URL Connection Set for a particular category of filters, right-click the appropriate node under the **External Connections** -> **URL Connection Sets** node in the tree. Select the **Add a URL Set** option to display the **URL Group** dialog. For more details, see the *Configuring URL Groups* topic.

XKMS Connections

The API Gateway can also validate certificates against an XKMS (XML Key Management Service) responder or group of responders. An XKMS Connection consists of a group of XKMS responders to validate certificates against, coupled with the signing key to use for signing requests to each of the responders in the group.

To add a global XKMS Connection, right-click the **External Connections** -> **XKMS Connection** node in the tree, and select the **Add an XKMS Connection** option to display the **Certificate Validation - XKMS** dialog. For more details, see the *XKMS Certificate Validation* topic.

All global XKMS Connections are available for selection when configuring the **Certificate Validation - XKMS** filter. This saves the administrator from reconfiguring XKMS connection details across multiple filters.

Global Caches

Overview

In cases where a Web Service is serving the same request (and generating the same response) over and over again, it makes sense to use a caching mechanism. When a cache is employed, a unique identifier for the request is cached together with the corresponding response for this request. If an identical request is received, the response can be retrieved from the cache instead of forcing the Web Service to reprocess the identical request and generate the same response. The use of caching in this way helps divert unnecessary traffic from the Web Service and makes it more responsive to new requests.

For example, assume you have deployed a Web Service that returns a list of cities in the USA from an external database, which is then used by a variety of Web-based applications. Because the names and quantity of cities in the USA are relatively constant, if the Web Service is handling hundreds or thousands of requests every day, this represents a serious waste of processing time and effort, especially considering that the database that contains the relatively fixed list of city names is hosted on a separate machine to the service.

If you assume that the list of cities in the database does not change very often, it makes sense to use Oracle API Gateway to cache the response from the Web Service that contains the list of cities. Then when a request for this Web Service is identified by the API Gateway, the cached response can be returned to the client. This approach results in the following performance improvements:

- The API Gateway does not have to route the message on to the Web Service, therefore saving the processing effort required, and perhaps more importantly, saving the time it takes for the round trip.
- The Web Service does not have to waste processing power on generating the same list over and over again, therefore making it more responsive to requests for other services.
- Assuming a naive implementation of database retrieval and caching, the Web Service does not have to query the database (over the network) and collate the results over and over again for every request.

The caching mechanism used in the API Gateway offers full control over the size of the cache, the lifetime of objects in the cache, whether objects are cached to disk or not, and even whether caches can be replicated across multiple instances of API Gateways. The following sections describe how to configure both local and distributed caches in the API Gateway, concluding with a detailed example of how to configure a policy to cache responses .

Local Caches

Local caches are used where a single API Gateway instance has been deployed. In such cases, you do not need to replicate caches across multiple running instances of the API Gateway.

Adding a Local Cache
In the Policy Studio tree, you can add a local cache by selecting the **Libraries** -> **Caches** node, and clicking the **Add** button at the bottom right of the screen. Select **Add Local Cache** from the menu. You can configure the following fields on the **Configure Local Cache** dialog:

Cache Name:
Enter a name for the cache.

Maximum Elements in Memory:
Enter the maximum number of objects that can be in memory at any one time.

Maximum Elements on Disk:
Sets the maximum number of objects that can be stored in the disk store at any one time. A value of zero indicates an unlimited number of objects.

Eternal:

If this option is selected, objects stored in the caches never expire and timeouts have no effect.

Overflow to Disk:
Select this option if you want the cache to overflow to disk when the number of objects in memory has reached the amount set in the **Maximum Elements in Memory** field above.

Note

The following fields are optional:

Time to Idle:
Determines the maximum amount of time (in seconds) between accesses that an object can remain idle before it expires. A value of zero indicates that objects can idle for infinity, which is the default value. If the **Eternal** field is selected, this setting is ignored.

Time to Live:
Sets the maximum time between when an object is created and when it expires. The default value is zero, which means that the object can live for infinity. If the **Eternal** field is selected, this setting is ignored.

Persist to Disk:
If selected, the disk store is persisted between JVM restarts. This option is disabled by default.

Disk Expiry Interval:
Configures the number of seconds between runs of the disk expiry thread. The default is 120 seconds.

Disk Spool Buffer Size:
Indicates the size of memory (in MBs) to allocate the disk store for a spool buffer. Writes are made to this memory and then asynchronously written to disk. The default size is 30 MB. If you get `OutOfMemory` exceptions, you may consider lowering this value. However, if you notice poor performance, you should increase the value.

Eviction Policy:
Select the eviction policy that the cache uses to evict objects from the cache. The default policy is **Least Recently Used**. However, you can also use **First in First Out** and **Less Frequently Used**.

Distributed Caches

If you have deployed several API Gateways throughout your network, you need to employ a distributed cache. In this scenario, each API Gateway has its own local copy of the cache but registers a cache event listener that replicates messages to the other caches so that put, remove, expiry, and delete events on a single cache are duplicated across all other caches.

Adding a Distributed Cache
You can add a distributed cache by selecting the **Libraries** -> **Caches** tree node, and clicking the **Add** button at the bottom right of the screen. Select **Add Distributed Cache** from the menu, and configure the following fields on the **Configure Distributed Cache** dialog:

Note

Many of the settings for the distributed cache are identical to those for the local cache. For details on how to configure these fields, see the Local Caches section. The following information refers to fields that are not displayed on both dialogs.

Event Listener Class Name:
Enter the name of the listener factory class that enables this cache to register listeners for cache events, such as put, remove, delete, and expire.

Properties Separator:
Specify the character to use to separate the list of properties.

Properties:
Specify the properties to pass to the `RMICacheReplicatorFactory`. The following properties are available:

- `replicatePuts=true | false`
  Determines whether new elements placed in a cache are replicated to other caches. Default is `true`.
- `replicateUpdates=true | false`
  Determines whether new elements that override (update) existing elements with the same key in a cache are replicated. Default is `true`.
- `replicateRemovals=true`
  Determines whether element removals are replicated. Default is `true`.
- `replicateAsynchronously=true | false`
  Determines whether replications are asynchronous (true) or synchronous (false). Default is `false`.
- `replicateUpdatesViaCopy=true | false`
  Determines whether new elements are copied to other caches (true) or a remove message is sent (false). Default is `true`.
- `asynchronousReplicationIntervalMillis=[number of ms]`
  The asynchronous replicator runs at a set interval of milliseconds. The default is 1000 and the minimum is 10. This property is only applicable if `replicateAsynchronously=true`.

Cache Bootstrap Class Name:
Specifies a `BootstrapCacheLoader` factory that the cache can call on initialization to pre-populate itself. The `RMIBootstrapCacheLoader` bootstraps caches in clusters where `RMICacheReplicators` are used.

Properties Separator:
The character entered here is used to separate the list of properties listed in the field below.

Properties:
The properties listed here are used to initialize the `RMIBootstrapCacheLoaderFactory`. The following properties are recognized:

- `bootstrapAsynchronously=true | false`
  Determines whether the bootstrap happens in the background after the cache has started (true), or if bootstrapping must complete before the cache is made available (false). Default is `true`.
- `maximumChunkSizeBytes=[integer]`
  Caches can potentially grow larger than the memory limits on the JVM. This property enables the bootstrapper to fetch elements in chunks. The default chunk size is 5000000 (5 MB).

Distributed Cache Settings

In a distributed cache, there is no master cache controlling all caches in the group. Instead, each cache is a peer in the group and needs to know where all the other peers in the group are located. *Peer Discovery* and *Peer Listeners* are two essential parts of any distributed cache system.

Editing Distributed Cache Settings

You can configure distributed cache settings by selecting the **Settings** node in the Policy Studio tree, and clicking the **Cache Settings** tab at the bottom of the screen on the right. You can configure the following fields:

Peer Provider Class:
By default, the built-in peer discovery class factory is used:
`net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory`

Properties Separator:

Specify the token used as the separator for the list of properties in the next field.

Properties:
The properties listed here specify whether the peer discovery mechanism is automatic or manual. If the automatic mechanism is used, each peer uses TCP multicast to establish and maintain a multicast group. This is the default option because it requires minimal configuration and peers can be automatically added and removed from the group. Each peer pings the group every second. If a peer has not pinged any of the other peers after 5 seconds, it is dropped from the group, while a new peer is admitted to the group if it starts pinging the other peers. To use automatic peer discovery, ensure that the `peerDiscovery` setting is set to `automatic`. You can specify the multicast address and port using the `multicastGroupAddress` and `multicastGroupPort` settings. You can specify the time to live for multicast datagrams using the `timeToLive` setting.

Alternatively, you can configure a manual peer discovery mechanism, whereby each peer definitively lists the peers that it wants to communicate with. This should only be used in networks where there are problems propagating multicast datagrams. To use a manual peer discovery mechanism, make sure the `peerDiscovery` setting is set to `manual`. The list of RMI URLs of the other peers in the group must also be specified, for example:
`rmiUrls=//server2:40001/sampleCache1|//server2:40001/sampleCache2` .

Peer Listener Class:
The peer listener class specified is responsible for listening for messages from peers in the group.

Properties Separator:
Specify the token used to separate the list of properties.

Properties:
The properties entered configure the way the listener behaves. Valid properties are as follows:

- **hostname (optional)**
  Hostname of the machine on which the listener is listening.

  Note

  By default, this is set to `localhost`, which maps to the local loopback address of `127.0.0.1`, which is not addressable from another machine on the network. If you intend this cache to be used over the network, you should change this address to the IP address of the network interface on which the listener is listening.

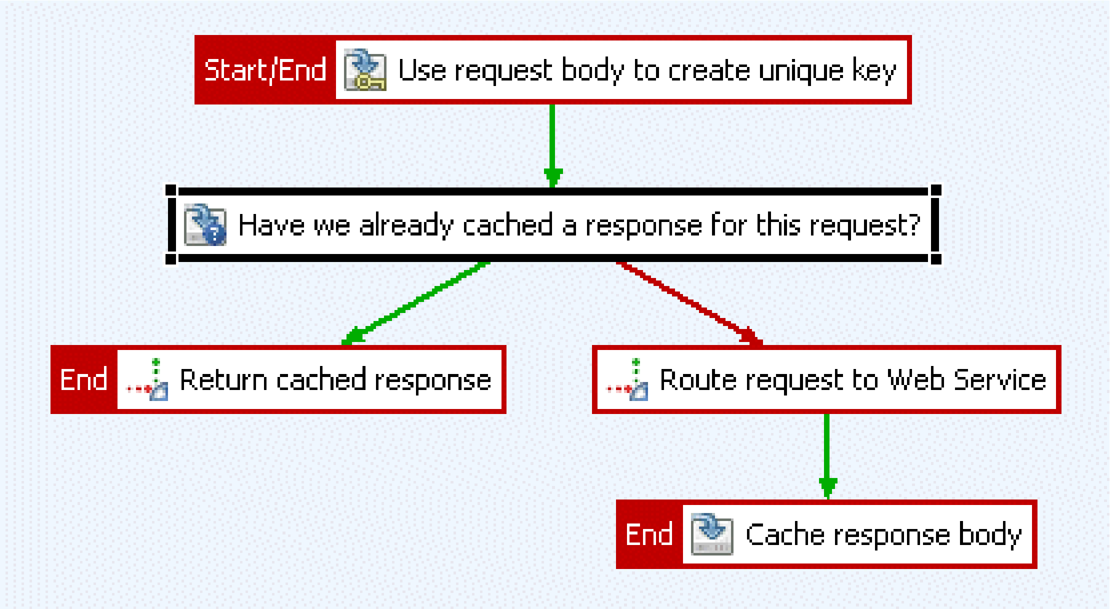
- **port (mandatory)**
  Specify the port on which the listener is listening, which by default is 4001. This setting is mandatory.
- **socketTimeoutMillis (optional)**
  Enter the number of seconds that client sockets wait when sending messages to this listener until they give up. The default is 2000 ms.

Notify replicators of removal of items during refresh:
A server refresh automatically purges all items from the cache (for example, when configuration updates are deployed to the API Gateway). If this checkbox is selected, the contents of each peer in the group are also purged. This avoids a situation where a single peer is refreshed (and has its contents purged), but the other peers in the group are not purged. If this option is not selected, the refreshed peer attempts to bootstrap itself to the other peers in the group, resulting in the cache items becoming replicated in the refreshed cache. This effectively negates the effect of the server refresh and may result in inconsistent behavior.

Example of Caching Response Messages

This simple example shows how to construct a policy that caches responses from the Web Service. It uses the request body to identify identical successive requests. In other words, if the API Gateway receives two successive requests with an identical message body, it returns the corresponding response from the cache instead of routing the request to the Web Service.

The following diagram illustrates the complete policy:


Start/End
Use request body to create unique key
Have we already cached a response for this request?
End
Return cached response
Route request to Web Service
End
Cache response body

The logic of the policy is summarized as follows:

1. The purpose of the first filter is to configure what part of the request you want to use to identify unique requests. This example uses the request body as the unique key, which is then used to look up the appropriate response message from the cache.
2. The second filter looks up the request body in the response cache to see if it contains the request body. If it does, the response message that corresponds to this request is returned to the client.
3. If it does not, the request is routed to the Web Service, which processes it (by connecting to a database over the network and running a SQL statement) and returns a response to the API Gateway.
4. The API Gateway then returns the response to the client and caches it in the response cache.
5. When the next identical request is received by the API Gateway, the corresponding response is located in the responses cache and returned immediately to the client.

You must configure the following caching filters to achieve this policy. For convenience, the routing filters are not included in this example because the configuration options depend on your target Web Service.

Create Key Filter:
This filter is used to decide what part of the request is used for a request to be considered unique. Different parts of the request can be identified internally using message attributes (for example, `content.body` contains the request body). The following fields must be configured for this filter:

- **Name:** Use request body to create unique key
- **Attribute Name:** `content.body`
- **Output attribute name:** `message.key`

Is Cached?:
This filter looks up the cache to see if a response has been stored for the current request. It looks up the cache using the `message.key` attribute by default. The `message.key` attribute contains a hash of the request message, and can be used as the key for objects in the cache. If the key is found in the cache, the value of the key (cached response for this request) is written to the `content.body` attribute, which can be returned to the client using the **Reflect** filter. You must

configure the following fields:

- **Name:** Is a response for this request already cached?
- **Cache containing key:** Response Cache (assuming you have created a cache of this name)
- **Attribute Containing Key:** `message.key`
- **Overwrite attribute name if found:** `content.body`

Reflect:
If the **Is Cached?** filter passes, it retrieves the response from the cache and stores it in the `content.body` message attribute. The **Reflect** filter is used to return the cached response to the client.

Routing:
If a response for this request could not be located in the cache, the API Gateway routes the request to the Web Service, and waits for a response. For more details on how to route messages, see the *Getting Started with Routing Configuration* tutorial.

Cache Attribute:
When the response has been received from the Web Service, it should be cached for future use. The **Cache Attribute** filter is used to configure the key used to look up the cache and which aspect of the response message is stored as the key value in the cache.

Note

This example specifies the value of the `content.body` attribute to cache. Because this filter is configured *after* the routing filters, this attribute contains the response message. The value entered in the **Attribute Key** field should match that entered in the **Attribute containing key** field in the **Is Cached?** filter. You must configure the following fields:

- **Name:** Cache response body
- **Cache to use:** Response Cache
- **Attribute key:** `message.key`
- **Attribute name to store:** `content.body`

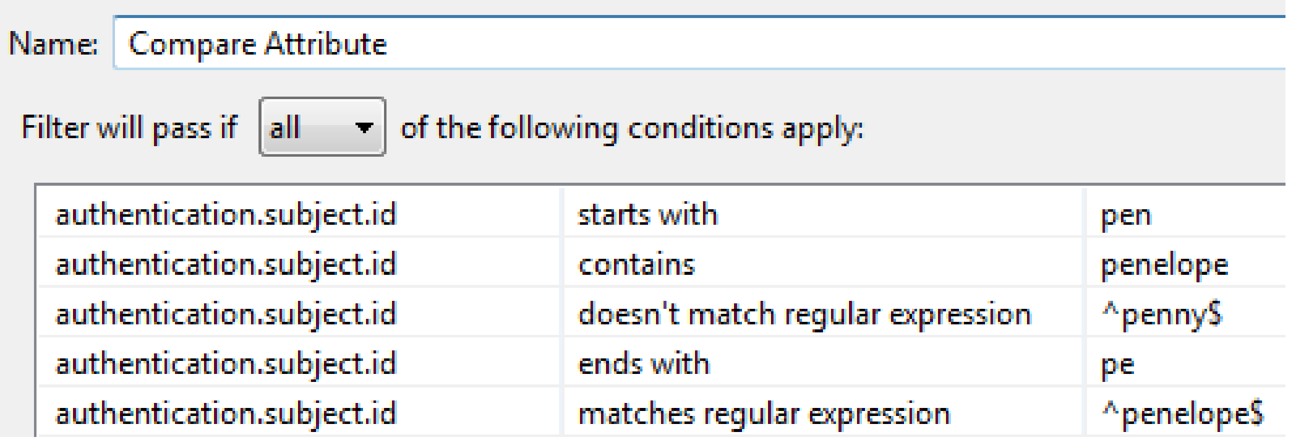
For more information on these filters, see the following topics:

Filter	***Topic***
Create Key	*Create Key*
Is Cached?	*Is Cached?*
Cache Attribute	*Cache Attribute*
Remove Cached Attribute	*Removed Cached Attribute*

Compare Attribute

Overview

The **Compare Attribute** filter enables you to compare the value of a specified message attribute on the API Gateway white board with the values specified in the filter. For example, the following filter only passes if the `authentication.subject.id` message attribute has a value of `penelope`:



Name: Compare Attribute

Filter will pass if all of the following conditions apply:

authentication.subject.id	starts with	pen
authentication.subject.id	contains	penelope
authentication.subject.id	doesn't match regular expression	^penny$
authentication.subject.id	ends with	pe
authentication.subject.id	matches regular expression	^penelope$

Configuration

Configure the following fields:

Name:
Enter an appropriate name for this filter.

Filter will pass if:
Select `all` or one of the conditions to apply in the drop-down list. Defaults to `all`. Click the **Add** button at the bottom right to add a rule condition. In the **Attribute filter rule** dialog, perform the following steps:

1. Enter a message attribute name in the **Attribute** text box on the left (for example, `http.request.verb` or `my.customer.attribute`).
2. Select one of the following rule conditions from the drop-down list:
   - `contains`
   - `doesn't contain`
   - `doesn't match regular expression`
   - `ends with`
   - `is`
   - `is not`
   - `matches regular expression`
   - `starts with`
3. Enter a value to compare with in the text box on the right (for example, `POST`). Alternatively, you can enter a selector that is expanded at runtime (for example, `${http.request.uri}`). For more details on selectors, see *Selecting Configuration Values at Runtime*.
4. Click **OK**.

Finally, to edit or delete an existing rule condition, select it in the table, and click the appropriate button.

Extract REST Request Attributes

Overview

This filter extracts the values of query string parameters and/or HTTP headers from a REST request and stores them in separate message attributes. The REST request can be an HTTP GET or POST request. This filter is found in the **Attributes** category in the Policy Studio. For details on how to create a REST request, see the *Create REST Request* filter.

Example REST Request
The following example shows an incoming REST request with query string and HTTP headers:

```
POST /services?name=Niall&location=Dublin&location=Pembroke%20St HTTP/1.1
Host: mail.google.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.15)
Gecko/20110303 Firefox/3.6.15
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Using this example, the **Extract REST Request Attributes** filter generates and populates the following attributes:

```
http.header.Host = mail.google.com
http.header.User-Agent = Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.15)
Gecko/20110303 Firefox/3.6.15
http.header.Accept = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
http.header.Accept-Language = en-gb,en;q=0.5
http.header.Accept-Encoding = gzip,deflate
http.header.Accept-Charset = ISO-8859-1,utf-8;q=0.7,*;q=0.7

http.querystring.name = Niall
http.querystring.location.1 = Dublin
http.querystring.location.2 = Pembroke St
```

Note

For multi-valued query string parameters (for example, `location`), each value is given an incremental index. For example, the multi-valued `location` parameter results in the creation of the `http.querystring.location.1` and `http.querystring.location.2` message attributes.

The purpose of this filter is to extract all parameters from an incoming REST request and store them in separate message attributes so that they can be validated easily, without needing to iterate through the set of `http.headers`.

Configuration

Configure the following fields on the **Extract REST Request Attributes** screen:

Name:
Enter an appropriate name for this filter.

Request Querystring:
Select whether to extract the values of query string parameters from an HTTP POST or GET request. These are simple name-value pairs (for example, `Name=Joe Bloggs`). This setting is selected by default.

HTTP Headers:
Select whether to extract the values of HTTP headers from an HTTP POST or GET request. This is selected by default.

Extract WSS Timestamp

Overview

You can use the **Extract WSS Timestamp** filter to extract a WSS Header Timestamp from a message. The timestamp is stored in a specified message attribute so that it can be processed later in a policy. This filter requires the WSS Header block to have been extracted previously. For more details, see the *Extract WSS Header* filter.

Typically, the **Validate Timestamp** filter is used to retrieve the timestamp from the specified message attribute and validate it. The **Validate Timestamp** filter is available from the **Content Filtering** filter category. For more details, see the *Validate Timestamp* filter.

Configuration

Configure the following fields on the **Extract WSS Timestamp** filter configuration screen:

Name:
Enter an appropriate name for this filter.

Message Attribute to Contain the Timestamp:
When the API Gateway extracts the WSS Header Timestamp from the message at runtime, it stores the timestamp in the specified message attribute. If you wish to validate the timestamp later in the policy, you *must* specify this message attribute in the configuration screen for the **Validate Timestamp** filter.

Extract WSS UsernameToken

Overview

You can use the **Extract WSS Username Token** filter to extract a WS-Security Username Token from a message if it exists. The extracted Username Token is stored in the `wss.usernameToken` message attribute.

If you want to process the Username Token later in the policy, you can specify this message attribute in the configuration screen for the processing filter. For example, if you want to sign the Username Token, you can simply specify the `wss.usernameToken` message attribute in the **What to Sign** section of the **Sign Message** filter. Open the **Message Attribute** tab on the **What to Sign** screen, and specify this attribute to sign the Username Token.

Configuration

Configure the following field on the **Extract WSS Username Token** filter configuration screen:

Name:
Enter an appropriate name for the filter. Remember that the WS-Security Username Token is stored in the `wss.usernameToken` message attribute.

Extract WSS Header

Overview

The **Extract WSS Header** filter extracts a WS-Security `<Header>` block from a message. The extracted security header is stored in the `authentication.ws.wsblockinfo` message attribute.

If you want to process this security header later in the policy, you can specify this message attribute in the configuration screen for the specific processing filter. For example, if you want to sign the security header, you can specify the `authentication.ws.wsblockinfo` message attribute in the **What to Sign** section of the **Sign Message** filter. Open the **Message Attribute** tab on the **What to Sign** screen, and specify this attribute to sign the security header.

Configuration

Configure the following fields on the **Extract WSS Header** filter configuration screen:

Name:
Enter an intuitive name for this filter (for example, **Extract Current Actor WSS Header**).

Actor or Role:
Specify the name of the SOAP Actor or Role of the WS-Security header that you want to extract. Remember, the WS-Security header is stored in the `authentication.ws.wsblockinfo` message attribute.

Remove enclosing WS-Security element:
This option removes the enclosing `<wsse:Security>` element from the message.

Get Cookie

Overview

An HTTP cookie is data sent by a server in an HTTP response to a client. The client can then return an updated cookie value in subsequent requests to the server. For example, this enables the server to store user preferences, manage sessions, track browsing habits, and so on.

The **Get Cookie** filter is used to read the `Cookie` and `Set-Cookie` HTTP headers. The `Cookie` header is used when a client sends a cookie to a server. The `Set-Cookie` header is used when the server instructs the client to store a cookie.

For more details, see the topic on the *Create Cookie* filter.

Configuration

Configure the following fields on the **Get Cookie Filter Configuration** screen:

Filter Name:
Enter an appropriate name to display for this filter.

Cookie Name:
Enter a regular expression that matches the name of the cookie. This value can use wildcards. Defaults to `.*`.

Remove all Cookie Headers from Message after retrieval:
When this setting is selected, all `Cookie` and `Set-Cookie` headers are removed from the message after retrieving the target cookie. This setting is not selected by default.

Attribute Storage

When a cookie is retrieved, it is stored in the appropriate API Gateway message attribute. The following message attributes are used to store cookies:

Cookie Header Type	***Message Attribute Name***
`Cookie`	`cookie.`*`cookie_name`*`.value` (for example, `cookie.mytest.value`)
`Set-Cookie`	`cookie.`*`cookie_name`*`.`*`cookie_attribute_name`* (for example, `cookie.mytest.header`)

Set-Cookie Attribute List
The `Set-Cookie` HTTP header includes the following cookie attributes (reflected in the `Set-Cookie` message attribute name):

Cookie Attribute Name	***Description***
`header`	The HTTP header name.
`value`	The value of the cookie.
`domain`	The domain name for this cookie.
`path`	The path on the server to which the browser returns this cookie.
`maxage`	The maximum age of the cookie in days, hours, minutes, and/or seconds.

Cookie Attribute Name	***Description***
`secure`	Whether sending this cookie is restricted to a secure protocol. This setting is not selected by default, which means that it can be sent using any protocol.
`HTTPOnly`	Whether the browser should use cookies over HTTP only. This setting is not selected by default.

Retrieve Attribute from Database

Overview

The API Gateway can retrieve user attributes from a specified database, or write user attributes to a specified database. It can do this by running an SQL query on the specified database, or by invoking a stored procedure call.

General Configuration

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database

Configure the following fields on the **Database** tab:

Database Location:
The API Gateway searches the selected database for the user's attributes. Click the button on the right to select the database to search. To use an existing database connection (for example, `Default Database Connection`), select it in the tree. To add a database connection, right-click the **Database Connections** tree node, and select **Add DB connection**. Alternatively, you can add database connections under the **External Connections** node in the Policy Studio tree view. For more information on configuring database connections, see the *Database Connection* topic.

Database Statements:
The **Database Statements** table lists the currently configured SQL queries or stored procedure calls. These queries and calls retrieve certain user attributes from the database selected in the **Database Location** field. You can edit and delete existing queries by selecting them from the drop-down list and clicking the **Edit** and **Delete** buttons. For more information on how to configure a **Database Query**, see the *Database Query* topic.

Advanced

On the **Advanced** tab, configure the following fields in the **User Attribute Extraction** section:

Place query results into user attribute list:
Select whether to place the database query results in a user attribute list using this setting (selected by default). When selected, the query results are placed in the `attribute.lookup.list` message attribute.

Associate attributes with user ID returned by selector
When the **Place query results into message attribute list** setting is selected, you can select or enter a user ID to associate with the user attributes. For example, if the user name is stored as `admin` in the database, you must select the message attribute containing the value `admin`. The API Gateway then looks up the database using this name. By default, the user ID is stored in the `${authentication.subject.id}` message attribute.

Configure the following fields on the **Attribute Naming** section:

Enable legacy attribute naming for retrieved attributes:
Specifies whether to enable legacy naming of retrieved message attributes (unselected by default). Prior to version 7.1, retrieved attributes were stored in message attributes in the following format:

```
user.<retrieved_attribute_name>
```

For example, `${user.email}`, `${user.role}`, and so on. If the retrieved attribute was multi-valued, you would access the values using `${user.email.1}` or `${user.email.2}`, and so on.

In version 7.1 and later, by default, you can now query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}`, or `${user.email[1]}`, and so on). You can also access other previously unreachable fields in the retrieved attribute (for example, `${user.email.attKey}` or `${user.email.namespace}`). Select this setting if you wish to use the legacy format for attribute naming.

Prefix for message attribute names:
You can specify an optional prefix for message attribute names. The default prefix is `user.`

Attribute name for stored procedure out parameters:
You can also specify an attribute name for stored procedure out parameters. The default prefix is `out.param.value.`

Case for attribute names:
You can specify whether attribute names are in lower case or upper case. The default is lower case.

Retrieve Attributes from Directory Server

Overview

The API Gateway can leverage an existing directory server by querying it for user profile data. The **Retrieve from Directory Server** filter can lookup a user, retrieve that user's attributes, and set them to the `attribute.lookup.list` message attribute, which stores a map of name-value pairs.

General Configuration

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database

Configure the following fields on the **Database** tab:

LDAP Directory:
The API Gateway queries the selected LDAP directory for user attributes. An LDAP connection is retrieved from a pool of connections at runtime. Click the button on the right to select the LDAP directory to query. If you wish to use an existing LDAP directory, (for example, `Sample Active Directory Connection`), you can select it in the tree. To add an LDAP directory, right-click the **LDAP Connections** tree node, and select **Add an LDAP Connection**. Alternatively, you can add LDAP connections under the **External Connections** node in the Policy Studio tree view. For more details on how to configure LDAP connections, see the topic on *Configuring LDAP Directories*.

The **Retrieve Unique User Identity** section enables you to select the user whose profile the API Gateway looks up in the directory server. The user ID can be taken from a message attribute or looked up from an LDAP directory.

From Message Attribute:
Select this option if the user ID is stored in a message attribute. A user's credentials are stored in the `authentication.subject.id` message attribute after authenticating to the API Gateway, so this is the most likely attribute to enter in this field. Typically, this contains the Distinguished Name (DName) or username of the authenticated user. The name extracted from the selected message attribute is used to query the directory server.

From LDAP Search:
In cases where you have not already obtained the user's identity and the `authentication.subject.id` attribute has not been pre-populated by a prior authentication filter, you must configure the API Gateway to retrieve the user's identity from an LDAP search. Click the **Configure Directory Search** button to configure the search criteria to use to retrieve the user's unique DName from the LDAP repository.

The **Retrieve Attributes** section instructs the API Gateway to search the LDAP tree to locate a specific user profile. When the appropriate profile is retrieved, the API Gateway extracts the specified user attributes.

Base Criteria:
This value specifies where the API Gateway should begin searching the LDAP directory. You can enter a selector representing the value of a message attribute, which is expanded at runtime. The two most likely message attributes to specify are the authenticated user's ID and Distinguished Name. The corresponding selector values are available in the drop-down list:

- `${authentication.subject.id}`
- `${authentication.subject.dname}`

However, you can enter selectors representing other message attributes using the same syntax. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Search Filter:
This is the name given by the particular LDAP directory to the *User* class. This depends on the type of LDAP directory configured. You can also use a selector to represent the value of a message attribute. For example, you can use the `user.role` attribute to store the user class. The syntax for using the selector representing this attribute is as follows:

```
(objectclass=${user.role})
```

Search Scope:
If the API Gateway retrieves a user profile node from the LDAP tree, the option selected here dictates the level that the API Gateway searches the node to. The available options are:

- Object level
- One level
- Sub-tree

Select the **Unique Result** option to force the API Gateway to retrieve a unique user profile from the LDAP directory. This is useful in cases where the LDAP search has returned several profiles.

The **Attribute Name** table lists the attributes the API Gateway retrieves from the user profile. If no attributes are listed, the API Gateway extracts all user attributes. In both cases, retrieved attributes are set to the `attribute.lookup.list` message attribute. Click **Add** to add the name of an attribute to extract from the returned user profile. Enter the attribute name to extract from the profile in the **Attribute Name** field of the **Attribute Lookup** dialog.

Important

- If the search returns results for more that one user, and the **Unique Result** option is enabled, an error is generated. If this option is not enabled, all attributes are merged.
- If an attribute is configured that does not exist in the repository, no error is generated.
- If no attributes are configured, all attributes present for the user are retrieved.

Advanced

Configure the following fields on the **Advanced** tab:

Enable legacy attribute naming for retrieved attributes:
Specifies whether to enable legacy naming of retrieved message attributes (unselected by default). Prior to version 7.1, retrieved attributes were stored in message attributes in the following format:

```
user.<retrieved_attribute_name>
```

For example, `${user.email}`, `${user.role}`, and so on. If the retrieved attribute was multi-valued, you would access the values using `${user.email.1}` or `${user.email.2}`, and so on.

In version 7.1 and later, by default, you can now query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}` or `${user.email[1]}`, and so on). You can also access other previously unreachable fields in the retrieved attribute (for example, `${user.email.attKey}` or `${user.email.namespace}`). Select this setting if you wish to use the legacy format for attribute naming.

Prefix for message attribute names:
You can specify an optional prefix for message attribute names. The default prefix is `user.`

Retrieve Attribute from HTTP Header

Overview

The **Retrieve from HTTP Header** attribute retrieval filter can be used to retrieve the value of an HTTP header and set it to a message attribute. For example, this filter can retrieve an X.509 certificate from a `USER_CERT` HTTP header, and set it to the `authentication.cert` message attribute. This certificate can then be used by the filter's successors. The following HTTP request shows an example of such a header:

```
POST /services/getEmployee HTTP/1.1
Host: localhost:8095
Content-Length: 21
SOAPAction: HelloService
USER_CERT:  MIIEZDCCA0 ...9aKD1fEQgJ
```

You can also retrieve a value from a named query string parameter and set this to the specified message attribute. The following example shows a request URL that contains a query string:

```
http://hostname.com/services/getEmployee?first=john&last=smith
```

In the above example, the query string is `first=john&last=smith`. As is clear from the example, query strings consist of attribute name-value pairs. Each name-value pair is separated by the `&` character.

Configuration

The following fields are available on the **Retrieve from HTTP Header** filter configuration screen:

Name:
Enter an appropriate name for this filter.

HTTP Header Name:
Enter the name of the HTTP header contains the value that we want to set to the message attribute.

Base64 Decode:
Check this box if the extracted value should be Base64 decoded before it is set to the message attribute.

Use Query String Parameters:
Select this setting if the API Gateway should attempt to extract the **HTTP Header Name** from the query string parameters instead of from the HTTP headers.

Attribute ID:
Finally, select the attribute used to store the value extracted from the request.

Insert SAML Attribute Assertion

Overview

A Security Assertion Markup Language (SAML) attribute assertion contains information about a user in the form of a series of attributes. Having collated a certain amount of information about a user, the API Gateway can generate a SAML attribute assertion, and insert it into the downstream message.

A *SAML Attribute* (see example below) is generated for each entry in the `attribute.lookup.list` attribute. Other filters from the **Attributes** filter group can be used to insert user attributes into the `attribute.lookup.list` attribute.

It may be useful to refer to the following example of a SAML attribute assertion when configuring this filter:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <soap:Header>
  <wsse:Security>
   <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                   ID="Id-0000010a3c4ff12c-0000000000000002"
                   IssueInstant="2006-03-27T15:26:12Z" Version="2.0">
    <saml:Issuer Format="urn:oasis ... WindowsDomainQualifiedName">
      TestCA
    </saml:Issuer>
    <saml:Subject>
     <saml:NameIdentifier Format="urn:oasis ... WindowsDomainQualifiedName">
      TestUser
     </saml:NameIdentifier>
    </saml:Subject>
    <saml:Conditions NotBefore="2005-03-27T15:20:40Z"
                     NotOnOrAfter="2028-03-27T17:20:40Z"/>
    <saml:AttributeStatement>
     <saml:Attribute Name="role" NameFormat="http://www.oracle.com">
      <saml:AttributeValue>admin</saml:AttributeValue>
     </saml:Attribute>
     <saml:Attribute Name="email" NameFormat="http://www.oracle.com">
      <saml:AttributeValue>joe@oracle.com</saml:AttributeValue>
     </saml:Attribute>
     <saml:Attribute Name="dept" NameFormat="">
      <saml:AttributeValue>engineering</saml:AttributeValue>
     </saml:Attribute>
    </saml:AttributeStatement>
   </saml:Assertion>
  </wsse:Security>
 </soap:Header>

 <soap:Body>
  <product>
   <name>API Gateway</name>
   <company>Oracle</company>
   <description>Web Services Security</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

General Configuration

Configure the following field:

Name:

Enter an appropriate name for the filter.

Assertion Details

Configure the following fields on the **Assertion Details** tab:

Issuer Name:
Select the certificate containing the Distinguished Name (DName) that you want to use as the Issuer of the SAML assertion. This DName is included in the SAML assertion as the value of the `Issuer` attribute of the `<saml:Assertion>` element. For an example, see the sample SAML assertion above.

Expire In:
Specify the lifetime of the assertion in this field. The lifetime of the assertion lasts from the time of insertion until the specified amount of time has elapsed.

Drift Time:
The **Drift Time** is used to account for differences in the clock times of the machine hosting the API Gateway (that generate the assertion) and the machines that consume the assertion. The specified time is subtracted from the time at which the API Gateway generates the assertion.

SAML Version:
You can create SAML 1.0, 1.1, and 2.0 attribute assertions. Select the appropriate version from the drop-down list.

Important

SAML 1.0 recommends the use of the `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` XML Signature Canonicalization algorithm. When inserting signed SAML 1.0 assertions into XML documents, it is quite likely that subsequent signature verification of these assertions will fail. This is due to the side effect of the algorithm including inherited namespaces into canonical XML calculations of the inserted SAML assertion that were not present when the assertion was generated.

For this reason, Oracle recommend that SAML 1.1 or 2.0 is used when signing assertions as they both uses the exclusive canonical algorithm `http://www.w3.org/2001/10/xml-exc-c14n#`, which safeguards inserted assertions from such changes of context in the XML document. Please see section 5.4.2 of the `oasis-sstc-saml-core-1.0.pdf` and section 5.4.2 of `sstc-saml-core-1.1.pdf` documents, both of which are available at `http://www.oasis-open.org`.

Assertion Location

The options on the **Assertion Location** tab specify where the SAML assertion is inserted in the message. By default, the SAML assertion is added to the WS-Security block with the current SOAP actor/role. The following options are available:

Append to Root or SOAP Header:
Appends the SAML assertion to the message root for a non-SOAP XML message, or to the SOAP Header for a SOAP message. For example, this option may be suitable for cases where this filter may process SOAP XML messages or non-SOAP XML messages.

Add to WS-Security Block with SOAP Actor/Role:
Adds the SAML assertion to the WS-Security block with the specified SOAP actor (SOAP 1.0) or role (SOAP 1.1). By default, the assertion is added with the current SOAP actor/role only, which means the WS-Security block with no actor. You can select a specific SOAP actor/role when available from the drop-down list.

XPath Location:
If you wish to insert the SAML assertion at an arbitrary location in the message, you can use an XPath expression to specify the exact location in the message. You can select XPath expressions from the drop-down list. The default is the `First WSSE Security Element`, which has an XPath expression of `//wsse:Security`. You can add, edit, or remove expressions by clicking the relevant button. For more details, see the *Configuring XPath Expressions* topic.

You can specify exactly how the SAML assertion is inserted using the following options:

- **Append to node returned by XPath expression** (the default)
- **Insert before node returned by XPath expression**
- **Replace node returned by XPath expression**

Insert into Message Attribute:
Specify a message attribute to store the SAML assertion from the drop-down list (for example, `saml.assertion`). Alternatively, you can also enter a custom message attribute in this field (for example, `my.test.assertion`). The SAML assertion can then be accessed downstream in the policy.

Subject Confirmation Method

The settings on the **Subject Confirmation Method** tab determine how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the `<SubjectConfirmation>` block can be used to authenticate either the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical `<SubjectConfirmation>` block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

The following configuration fields are available on the **Subject Confirmation Method** tab:

Method:
The value selected here determines the value of the `<ConfirmationMethod>` element. The following table shows the available methods, their meanings, and their respective values in the `<ConfirmationMethod>` element:

Method	***Meaning***	***Value***
Holder Of Key	The API Gateway includes the key used to prove that the API Gateway is the holder of the key, or includes a reference to the key.	`urn:oasis:names:tc:SAML:1.0:cm:holder-of-key`
Bearer	The subject of the assertion is the bearer of the assertion.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`
SAML Artifact	The subject of the assertion is the user that presented a SAML Artifact to the API Gateway.	`urn:oasis:names:tc:SAML:1.0:cm:artifact`
Sender Vouches	Use this confirmation method to assert that the API Gateway is acting on behalf of the authenticated end-user. No other information relating to the context	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	*Meaning*	*Value*
	of the assertion is sent. It is recommended that both the assertion **and** the SOAP Body must be signed if this option is selected. These message parts can be signed by using the *XML Signature Generation* filter.	

Note

You can also leave the **Method** field blank, in which case no `<ConfirmationMethod>` block is inserted into the assertion.

Holder-of-Key Configuration:
When you select `Holder-of-Key` as the SAML subject confirmation in the **Method** field, you must configure how information about the key is to be included in the message. There are a number of configuration options available depending on whether the key is a symmetric or asymmetric key.

Asymmetric Key:
If you want to use an asymmetric key as proof that the API Gateway is the holder-of-key entity, you must select the **Asymmetric Key** radio button, and then configure the following fields on the **Asymmetric** tab:

- **Certificate from Store:**
  If you want to select a key that is stored in the Certificate Store, select this option and click the **Signing Key** button. On the **Select Certificate** screen, select the box next to the certificate that is associated with the key that you want to use.
- **Certificate from Message Attribute:**
  Alternatively, the key may have already been used by a previous filter in the policy (for example, to sign a part of the message). In this case, the key is stored in a message attribute. You can specify this message attribute in this field.

Symmetric Key:
If you want to use a symmetric key as proof that the API Gateway is the holder of key, select the **Symmetric Key** radio button, and configure the fields on the **Symmetric** tab:

- **Generate Symmetric Key, and Save in Message Attribute:**
  If you select this option, the API Gateway generates a symmetric key, which is included in the message before it is sent to the client. By default, the key is saved in the `symmetric.key` message attribute.
- **Symmetric Key in Message Attribute:**
  If a previous filter (for example, a **Sign Message** filter) has already used a symmetric key, you can to reuse this key as proof that the API Gateway is the holder-of-key entity. You must enter the name of the message attribute in the field provided, which defaults to `symmetric.key`.
- **Encrypt using Certificate from Certificate Store:**
  When a symmetric key is used, you must assume that the recipient has no prior knowledge of this key. It must, therefore, be included in the message so that the recipient can validate the key. To avoid meet-in-the-middle style attacks, where a hacker could eavesdrop on the communication channel between the API Gateway and the recipient and gain access to the symmetric key, the key must be encrypted so that only the recipient can decrypt the key. One way of doing this is to select the recipient's certificate from the Certificate Store. By encrypting the symmetric key with the public in the recipient's certificate, the key can only be decrypted by the recipient's private key, to which only the recipient has access. Select the **Signing Key** button and then select the recipient's certificate on the **Select Certificate** dialog.
- **Encrypt using Certificate from Message Attribute:**

Alternatively, if the recipient's certificate has already been used (perhaps to encrypt part of the message) this certificate is stored in a message attribute. You can enter the message attribute in this field.

- **Symmetric Key Length:**
  Enter the length (in bits) of the symmetric key to use.
- **Key Wrap Algorithm:**
  Select the algorithm to use to encrypt (*wrap*) the symmetric key.

Key Info:
The **Key Info** tab must be configured regardless of whether you have elected to use symmetric or asymmetric keys. It determines how the key is included in the message. The following options are available:

- **Do Not Include Key Info:**
  Select this option if you do not wish to include a `<KeyInfo>` section in the SAML assertion.
- **Embed Public Key Information:**
  If this option is selected, details about the key are included in a `<KeyInfo>` block in the message. You can include the full certificate, expand the public key, include the distinguished name, and include a key name in the `<KeyInfo>` block by selecting the appropriate boxes. When selecting the **Include Key Name** field, you must enter a name in the **Value** field, and select the **Text Value** or **Distinguished Name Attribute** radio button, depending on the source of the key name.
- **Put Certificate in Attachment:**
  Select this option to add the certificate as an attachment to the message. The certificate is then referenced from the `<KeyInfo>` block.
- **Security Token Reference:**
  The Security Token Reference (STR) provides a way to refer to a key contained in a SOAP message from another part of the message. It is often used in cases where different security blocks in a message use the same key material, and it is considered an overhead to include the key more than once in the message.
  When this option is selected, a `<wsse:SecurityTokenReference>` element is inserted into the `<KeyInfo>` block. It references the key material using a `URI` to point to the key material and a `ValueType` attribute to indicate the type of reference used. For example, if the STR refers to an encrypted key, you should select `EncryptedKey` from the drop-down list, whereas if it refers to a `BinarySecurityToken`, select `X509v3` from the dropdown. Other options are available to enable more specific security requirements.

Advanced

The settings on the **Advanced** tab include the following fields.

Select Required Layout Type:
WS-Policy and SOAP Message Security define a set of rules that determine the layout of security elements that appear in the WS-Security header in a SOAP message. The SAML assertion is inserted into the WS-Security header according to the layout option selected here. The available options correspond to the WS-Policy Layout assertions of `Strict`, `Lax`, `LaxTimestampFirst`, and `LaxTimestampLast`.

Indent:
Select this method to ensure that the generated signature is properly indented.

Security Token Reference:
The generated SAML attribute assertion can be encapsulated in a `<SecurityTokenReference>` block. The following example demonstrates this:

```
<soap:Header>
 <wsse:Security
     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
     soap:actor="oracle">
  <wsse:SecurityTokenReference>
   <wsse:Embedded>
    <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
```

```
                    ID="Id-0000010a3c4ff12c-0000000000000002"
                    IssueInstant="2006-03-27T15:26:12Z" Version="2.0">
     <saml:Issuer Format="urn:oasis ... WindowsDomainQualifiedName">
       TestCA
     </saml:Issuer>
     <saml:Subject>
      <saml:NameID Format="urn:oasis ... WindowsDomainQualifiedName">
       TestUser
      </saml:NameID>
     </saml:Subject>
     <saml:Conditions NotBefore="2005-03-27T15:20:40Z"
                      NotOnOrAfter="2028-03-27T17:20:40Z"/>
     <saml:AttributeStatement>
      <saml:Attribute Name="role" NameFormat="http://www.oracle.com">
       <saml:AttributeValue>admin</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="email" NameFormat="http://www.oracle.com">
       <saml:AttributeValue>joe@oracle.com</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="attrib1" NameFormat="">
       <saml:AttributeValue xsi:nil="true"/>
       <saml:AttributeValue>value1</saml:AttributeValue>
      </saml:Attribute>
     </saml:AttributeStatement>
    </saml:Assertion>
   </wsse:Embedded>
  </wsse:SecurityTokenReference>
 </wsse:Security>
</soap:Header>
```

To add the SAML assertion to a `<SecurityTokenReference>` block like in this example, select the **Embed SAML assertion within Security Token Reference** option. Otherwise, select **No Security Token Reference**.

Retrieve Attributes with JSON Path

Overview

JSON Path is an XPath like query language for JSON (JavaScript Object Notation) that enables you to select nodes in a JSON document. The **Retrieve Attributes with JSON Path** filter enables you to retrieve specified message attributes from a JSON message using JSON Path expressions.

For more details on JSON Path, see http://code.google.com/p/jsonpath/.

Configuration

Configure the following fields on the **Retrieve Attributes with JSON Path** filter screen:

Name:
Enter an appropriate name for this filter.

Extract attributes using the following JSON Path expressions:
Specify the list of attributes for the API Gateway to retrieve using appropriate JSON Path expressions. All attribute values are stored in the `attribute.lookup.list` message attribute.

To add an attribute to the list, click the **Add** button, and enter the following values in the dialog:

- **Attribute name:**
  Enter the message attribute name that you wish to extract using JSON Path (for example, `bicycle.price`).
- **JSON Path Expression:**
  Enter the JSON Path expression that you wish to use to extract the message attribute (for example, `$.store.bicycle.price`). The Policy Studio prompts if you enter an unsupported JSON Path expression.
- **Unmarshal as:**
  Enter the data type to unmarshal the message attribute value as (defaults to `java.lang.String`).
- **Fail if JSON Path Fails:**
  Select whether the filter should fail if the specified JSON Path expression fails. This option is not selected by default.

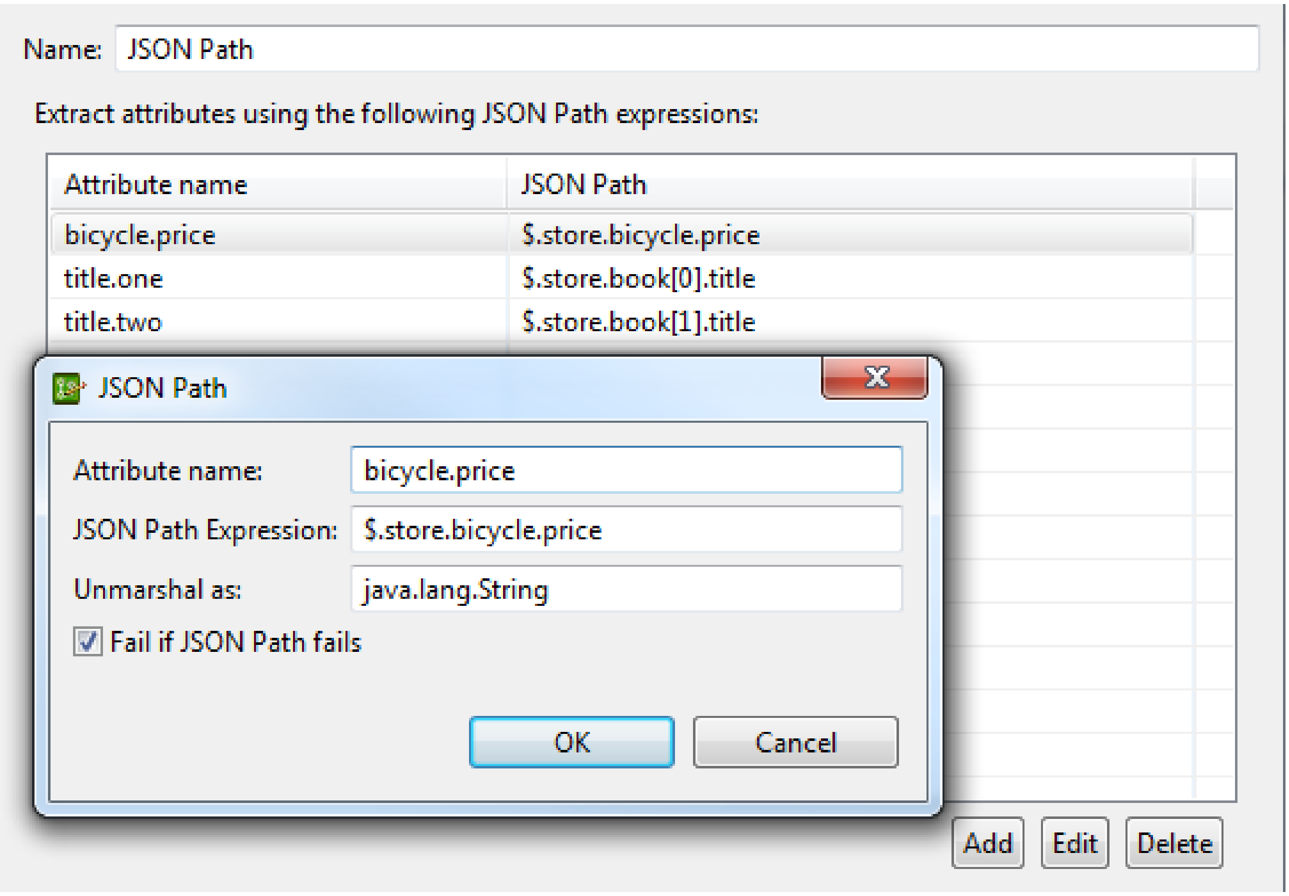
Note

If no attributes are specified, the API Gateway retrieves all the attributes in the message and sets them to the `attribute.lookup.list` attribute.

JSON Path Examples

The following are some examples of using the **Retrieve Attributes with JSON Path** filter to retrieve data from a JSON message.

Retrieving Attributes
The following example retrieves three different data items from the JSON message and stores them in the specified message attributes as strings:



Name: JSON Path

Extract attributes using the following JSON Path expressions:

Attribute name	JSON Path
bicycle.price	$.store.bicycle.price
title.one	$.store.book[0].title
title.two	$.store.book[1].title

JSON Path

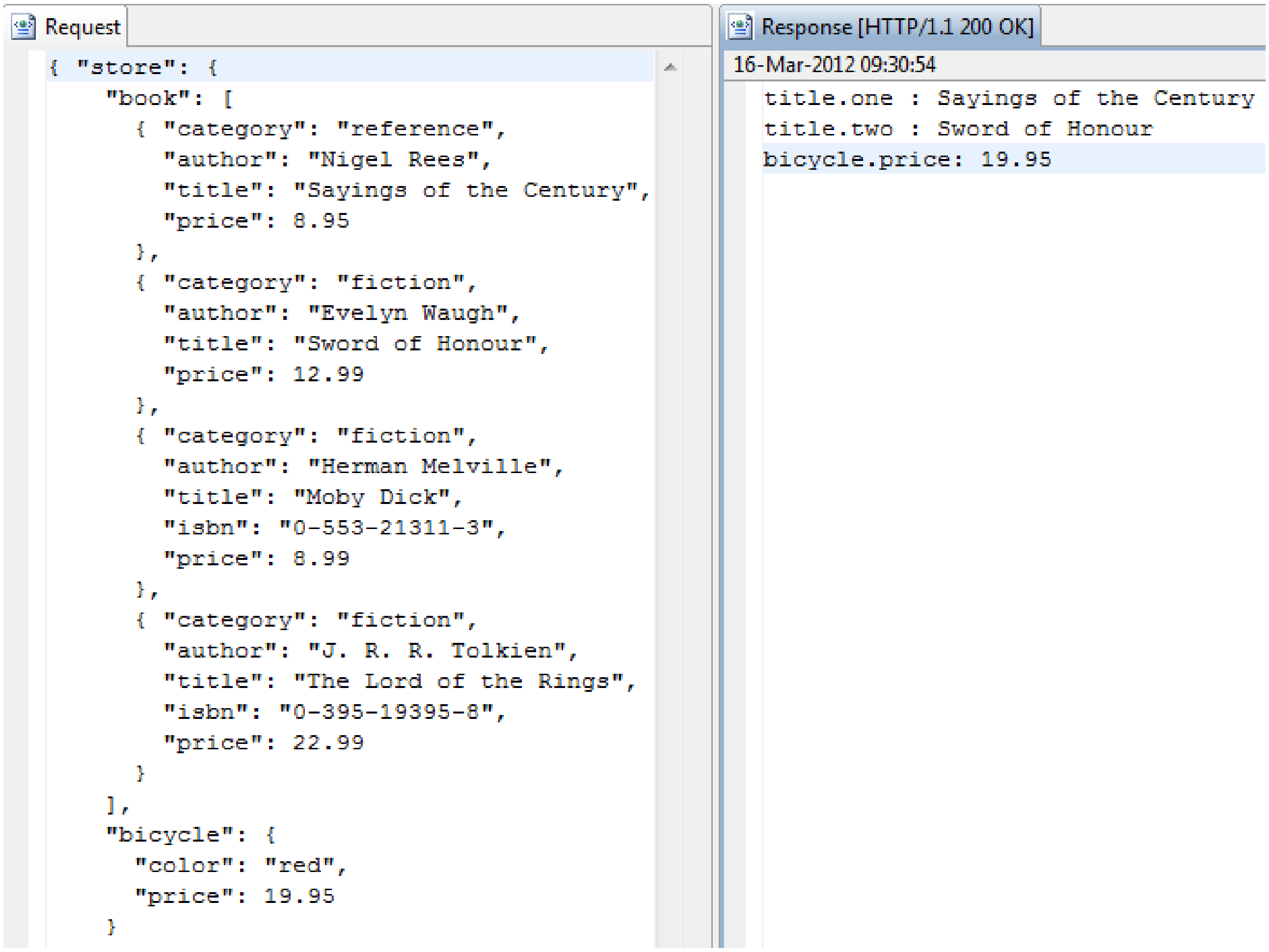
Attribute name: bicycle.price

JSON Path Expression: $.store.bicycle.price

Unmarshal as: java.lang.String

☑ Fail if JSON Path fails

When the extracted attributes are added to the `content.body` message attribute, the following example shows the corresponding request and response message in Oracle API Gateway Explorer:

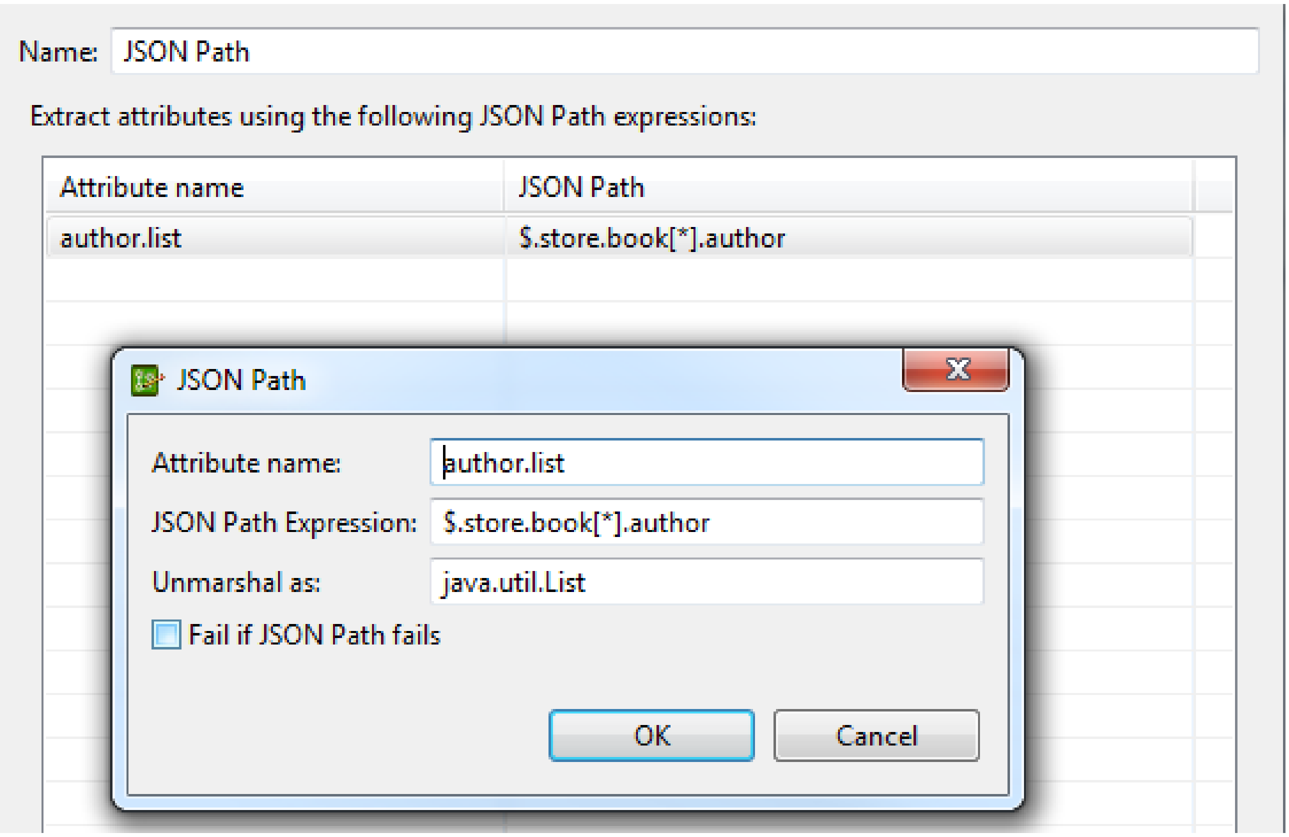


```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
```

```
title.one : Sayings of the Century
title.two : Sword of Honour
bicycle.price: 19.95
```

Retrieving Multiple Attributes in a List

The following example retrieves all the authors from the JSON message and stores them in the specified message attribute as a `List`:



Name: JSON Path

Extract attributes using the following JSON Path expressions:

Attribute name	JSON Path
author.list	$.store.book[*].author

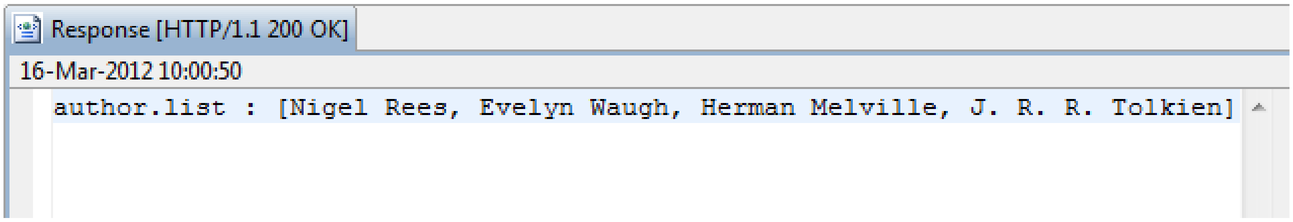
Attribute name: author.list

JSON Path Expression: $.store.book[*].author

Unmarshal as: java.util.List

□ Fail if JSON Path fails

The following example shows the corresponding request and response in Oracle API Gateway Explorer:



16-Mar-2012 10:00:50

```
author.list : [Nigel Rees, Evelyn Waugh, Herman Melville, J. R. R. Tolkien]
```

Retrieve Attribute from Message

Overview

The **Retrieve from Message** filter uses XPath expressions to extract the value of an XML element or attribute from the message and set it to an internal message attribute. The XPath expression can also return a `NodeList`, and the `NodeList` can be set to the specified message attribute.

Configuration

The following fields are available on the **Retrieve from Message** filter configuration screen:

Name:
Enter an appropriate name for this filter.

Attribute Location:
Configure an XPath expression to retrieve the desired content.

Click the **Add** button to add an XPath expression. You can add and remove existing expressions by clicking the **Edit** and **Remove** buttons respectively.

Extract the Content of the Node:
When this option is selected, the content of the XML element or attribute in the message is extracted and set to the specified message attribute.

Serialize All Nodes in the NodeList:
This option saves all nodes retrieved from the XPath expression to the specified message attribute as a `String` (for example, `<node1>test</node1>`). This option is useful for extracting `<Signature>`, `<Security>`, and `<UsernameToken>` blocks, as well as proprietary blocks of XML from messages.

Save as List of Nodes:
This option saves the nodes retrieved from the XPath expression to the specified message attribute as a Java `List`, where each item is of type `Node`. For example, if the XPath returns `<node1>test</node1>`, there is one node in the `List` (`<node1>`). The child text node (`test`) is accessible from that node, but is not saved as an entry in the `List` at the top-level.

Attribute ID:
The API Gateway sets the value of the message attribute selected here to the value extracted from the message. You can also enter a user-defined message attribute.

Retrieve Attribute from SAML Attribute Assertion

Overview

A SAML (Security Assertion Markup Language) attribute assertion contains information about a user in the form of a series of attributes. The **Retrieve from SAML Attribute Assertion** can retrieve these attributes and store them in the `attribute.lookup.list` message attribute.

The following SAML attribute assertion contains 3 attributes, "role", "email", and "dept". The **Retrieve from SAML Attribute Assertion** will store all 3 attributes and their values in the `attribute.lookup.list` message attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <soap:Header>
  <wsse:Security>
   <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                   ID="Id-0000010a3c4ff12c-0000000000000002"
                   IssueInstant="2006-03-27T15:26:12Z" Version="2.0">
    <saml:Issuer Format="urn:oasis ... WindowsDomainQualifiedName">
      TestCA
    </saml:Issuer>
    <saml:Subject>
     <saml:NameIdentifier Format="urn:oasis ... WindowsDomainQualifiedName">
      TestUser
     </saml:NameIdentifier>
    </saml:Subject>
    <saml:Conditions NotBefore="2005-03-27T15:20:40Z"
                     NotOnOrAfter="2028-03-27T17:20:40Z"/>
    <saml:AttributeStatement>
     <saml:Attribute Name="role" NameFormat="http://www.oracle.com">
      <saml:AttributeValue>admin</saml:AttributeValue>
     </saml:Attribute>
     <saml:Attribute Name="email" NameFormat="http://www.oracle.com">
      <saml:AttributeValue>joe@oracle.com</saml:AttributeValue>
     </saml:Attribute>
     <saml:Attribute Name="dept" NameFormat="">
      <saml:AttributeValue>engineering</saml:AttributeValue>
     </saml:Attribute>
    </saml:AttributeStatement>
   </saml:Assertion>
  </wsse:Security>
 </soap:Header>

 <soap:Body>
  <product>
   <name>API Gateway</name>
   <company>Oracle</company>
   <description>Web Services Security</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

Details

The following fields are available on the **Details** configuration tab:

Name:
Enter a name for this filter here.

SOAP Actor/Role:
If you expect the SAML assertion to be embedded within a WS-Security block, you can identify this block by specifying the SOAP Actor or Role of the WS-Security header that contains the assertion.

XPath Expression:
Alternatively, if the assertion is not contained within a WS-Security block, you can enter an XPath expression to locate the attribute assertion. XPath expressions can be added by selecting the **Add** button. Expressions can be edited and deleted by selecting an XPath expression and clicking the **Add** and **Delete** buttons respectively.

SAML Namespace:
Select the SAML namespace that must be used on the SAML assertion in order for this filter to succeed. If you do not wish to check the namespace, select the "Do not check version" option from the dropdown.

SAML Version:
Enter the SAML Version that the assertion must adhere to by entering the major version in the 1st field, followed by the minor version in the 2nd field. For example, for SAML version 2.0, enter "2" in the 1st field and "0" in the 2nd field.

Drift Time:
When the API Gateway receives a SAML attribute assertion, it first checks to make sure that it has not expired. The lifetime of the assertion is specified using the "NotBefore" and "NotOnOrAfter" attributes of the `<Conditions>` element in the assertion itself. The API Gateway makes sure that the time at which it validates the assertion is between the "NotBefore" and "NotOnOrAfter" times.

The **Drift Time** is used to account for differences in the clock time of the machine that generated the assertion and the machine hosting the API Gateway. The time specified here will be subtracted from the time at which the API Gateway attempts to validate the assertion.

Trusted Issuers

You can use the table on this tab to select the issuers that you consider trusted. In other words, this filter will only accept assertions that have been issued by the SAML Authorities selected here.

Click the **Add** button to display the **Trusted Issuers** screen. Select the Distinguished Name of a SAML Authority whose certificate has been added to the Certificate Store and click the **OK** button. Repeat this step to add more SAML Authorities to the list of trusted issuers.

Subject Configuration

The API Gateway can perform some very basic authentication checks on the subject or sender of the assertion using the options available on the **Subject** tab. The API Gateway can compare the subject of the assertion (i.e. the `<NameIdentifier>`) to one of the following values:

- **Subject of the Authentication Filter:**
  Select this option if the user specified in the `<NameIdentifier>` element must match the user that authenticated to the API Gateway. The subject of the authentication event is stored in the `authentication.subject.id` message attribute.
- **A User-Specified Value:**
  This option can be used if the `<NameIdentifier>` must match a user-specified value. Select this radio button and enter the value in the field provided.
- **No Authentication:**
  If the **Neither of the above** radio button is selected, the API Gateway will not attempt to match the `<NameIdentifier>` to any value.

Lookup Attributes

The **Lookup Attributes** tab is used to determine what attributes the API Gateway should extract from the SAML attribute assertion. Extracted attributes and their values will be set to the `attribute.lookup.list` message attribute.

The table lists the attributes that the API Gateway will extract from the assertion and set to the `attribute.lookup.list`.

Alternatively, check the **Extract all of the attributes from the SAML assertion** checkbox to configure the API Gateway to extract all attributes from the assertion. All attributes will be set to the `attribute.lookup.list` message attribute.

To configure a specific attribute to lookup in the message, click the **Add** button to display the **Attribute Lookup** dialog. Enter the value of the "Name" attribute of the `<Attribute>` element in the **Name** field. Enter the value of the "NameFormat" attribute of the `<Attribute>` element in the **Namespace** field.

SAML PDP Attributes

Overview

The API Gateway can request information about an authenticated end-user in the form of user *attributes* from a SAML PDP (Policy Decision Point) using the SAML Protocol (SAMLP). In such cases, the API Gateway presents evidence to the PDP in the form of some user credentials, such as the Distinguished Name of a client's X.509 certificate.

The PDP looks up its configured user store and retrieves attributes associated with that user. The attributes are inserted into a SAML attribute assertion and returned to the API Gateway in a SAMLP response. The assertion and/or SAMLP response is usually signed by the PDP.

When the API Gateway receives the SAMLP response, it performs a number of checks on the response, such as validating the PDP signature and certificate, and examining the assertion. It can also insert the SAML attribute assertion into the original message for consumption by a downstream Web Service.

Request Configuration

This section describes how the API Gateway should package the SAMLP request before sending it to the SAML PDP.

SAML PDP URL Sets
You can configure a group of SAML PDPs to which the API Gateway connects in a round-robin fashion if one or more of the PDPs are unavailable. This is known as a SAML PDP URL Set. You can configure a SAML PDP URL Set using this screen or under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring URL Groups*.

You can configure the following general fields:

- **SAML PDP URL Set:**
  Click the button on the right, and select a previously configured SAML PDP URL Set from the tree. To add a URL Set, right-click the **SAML PDP URL Sets** tree node, and select **Add a URL Set**. Alternatively, you can configure a SAML PDP URL Set under the **External Connections** node in the Policy Studio tree.
- **SOAPAction:**
  Enter the SOAP Action required to send SAML Protocol requests to the PDP. Click the **Use Default** button to use the following default SOAP Action as specified by the SAML Protocol:
  http://www.oasis-open.org/committees/security
- **SAML Version:**
  Select the SAML version to use in the SAMLP request.
- **Signing Key:**
  If the SAMLP request is to be signed, click the **Signing Key** button, and select the appropriate signing key from the Certificate Store.

SAML Subject:
These details describe the *subject* of the SAML assertion. Complete the following fields:

- **Subject Attribute:**
  Select the message attribute that contains the name of an authenticated username. By default, the `authentication.subject.id` message attribute is selected, which contains the username of the authenticated user.
- **Subject Format:**
  Select the format of the message attribute selected in the **Subject Attribute** field above.

Note

There is no need to select a format here if the **Subject Attribute** field is set to `authentication.subject.id`

Subject Confirmation:
The settings on the **Confirmation Method** tab determine how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the `<SubjectConfirmation>` block can be used to authenticate either the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical `<SubjectConfirmation>` block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

You must configure the following fields on the **Subject Confirmation** tab:

Method:
The selected value determines the value of the `<ConfirmationMethod>` element. The following table shows the available methods, their meanings, and their respective values in the `<ConfirmationMethod>` element:

Method	***Meaning***	***Value***
Holder Of Key	A `<SubjectConfirmation>` is inserted into the SAMLP request. The `<SubjectConfirmation>` contains a `<dsig:KeyInfo>` section with the certificate of the user selected to sign the SAMLP request. The user selected to sign the SAMLP request must be the authenticated subject (`authentication.subject.id`). Select the **Certificate is included** if the signer's certificate is to be included in the `SubjectConfimration` block. Alternatively, select the **Only key name is included** radio button if only the key name is to be included. Select the user whose private key is used to sign part of the message in the **User Name** drop-down list on the **Sign Request** tab.	`urn:oasis:names:tc:SAML:1.0:cm:holder-of-key`

Method	***Meaning***	***Value***
Bearer	A `<SubjectConfirmation>` is inserted into the SAMLP request.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`
SAML Artifact	A `<SubjectConfirmation>` is inserted into the SAMLP request.	`urn:oasis:names:tc:SAML:1.0:cm:artifact`
Sender Vouches	A `<SubjectConfirmation>` is inserted into the SAMLP request. The SAMLP request must be signed by a user.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

If the **Method** field is left blank, no `<ConfirmationMethod>` block is inserted into the assertion.

Include Certificate:
Select this option if you wish to include the SAML subject's certificate in the `<KeyInfo>` section of the `<SubjectConfirmation>` block.

Include Key Name:
Alternatively, if you do not want to include the certificate, you can select this option to only include the key name in the `<KeyInfo>` section.

Attributes:
You can list a number of user attributes to include in the SAML attribute assertion that is generated by the API Gateway. If no attributes are explicitly listed in this section, the API Gateway inserts all attributes associated with the user (all user attributes in the `attribute.lookup.list message attribute`) in the assertion.

To add a specific attribute to the SAML attribute assertion, click the **Add** button. A user attribute can be configured using the **Attribute Lookup** dialog.

Enter the name of the attribute that is added to the assertion in the **Attribute Name** field. Enter the namespace that is associated with this attribute in the **Namespace** field.

You can edit and remove previously configured attributes using the **Edit** and **Remove** buttons.

Response Configuration

The fields on this tab relate to the SAMLP Response returned from the SAML PDP. The following fields are available:

SOAP Actor/Role:
If the SAMLP response from the PDP contains a SAML attribute assertion, the API Gateway can extract it from the response and insert it into the downstream message. The SAML assertion is inserted into the WS-Security block identified by the specified SOAP actor/role.

Drift Time:
The SAMLP request to the PDP is time stamped by the API Gateway. To account for differences in the times on the machines running the API Gateway and the SAML PDP the specified time is subtracted from the time at which the API Gateway generates the SAMLP request.

Retrieve Attribute from User Store

Overview

The **User Store** stores a user's profile, including attributes relating to that user. After a user has successfully authenticated to the API Gateway, the **Retrieve From User Store** filter can retrieve attributes belonging to that user from the **User Store**. All attributes that are retrieved are set to the `attribute.lookup.list`.

General Configuration

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database

Configure the following fields on the **Database** tab:

User ID:
Select or enter the name of the message attribute that contains the name of the user to look up in the **User Store**. For example, if the user name is stored as `admin`, you must select the message attribute containing the value `admin`. The API Gateway then looks up the user the **User Store** using this name.

Attributes:
Enter the list of attributes that the API Gateway should retrieve if it successfully looks up the user identified by the message attribute specified in the **User ID** field. All attribute values are stored in the `attribute.lookup.list` message attribute.

If no user attributes are specified, the API Gateway retrieves all the user's registered attributes and sets them to the `attribute.lookup.list` attribute.

You can add attributes by selecting the **Add** button. Similarly, you can edit and remove existing attributes by selecting the **Edit** and **Remove** buttons.

Advanced

Configure the following fields on the **Advanced** tab:

Enable legacy attribute naming for retrieved attributes:
Specifies whether to enable legacy naming of retrieved message attributes (unselected by default). Prior to version 7.1, retrieved attributes were stored in message attributes in the following format:

```
user.<retrieved_attribute_name>
```

For example, `${user.email}`, `${user.role}`, and so on. If the retrieved attribute was multi-valued, you would access the values using `${user.email.1}` or `${user.email.2}`, and so on.

In version 7.1 and later, by default, you can now query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}` or `${user.email[1]}`, and so on). You can also access other previously unreachable fields in the retrieved attribute (for example, `${user.email.attKey}` or `${user.email.namespace}`). Select this setting if you wish to use the legacy format for attribute naming.

Prefix for message attribute names:
You can specify an optional prefix for message attribute names. The default prefix is `user.`

Attribute Authentication

Overview

In cases where user credentials are passed to the API Gateway in a non-standard way, these credentials can be copied into API Gateway message attributes, and then authenticated against a specified authentication repository, such as the API Gateway User Store, an LDAP directory, or a database. For example, assume that username and password credentials are passed to the API Gateway in the following XML message:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <ns:User xmlns:ns="http://www.user.com">
      <ns:Username>1</ns:Username>
      <ns:Password>2</ns:Password>
    </ns:User>
  </s:Body>
</s:Envelope>
```

In this example, the standard methods of passing credentials, such as HTTP Basic/Digest authentication, SAML assertions, WS-Security Username tokens, are bypassed, and the client sends the username and password as parameters in a simple SOAP message. When the API Gateway receives this message, it can extract the value of the `<Username>` and `<Password>` elements using an XPath expression configured in the **Retrieve Attributes from Message** filter. This filter uses an XPath expression to retrieve the value of an element or attribute, and can then store this value in the specified message attribute.

In this example, you can configure an instance of this filter to retrieve the value of the `<Username>` attribute, and store it in the `authentication.subject.id` message attribute. Similarly, you can configure another filter to retrieve the value of the `<Password>`, and store it in the `authentication.subject.password` message attribute.

The **Attribute Authentication** filter can then use the username and password values stored in these message attributes to authenticate the user against the specified authentication repository.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for this filter.

Username:
Specify the API Gateway message attribute that contains the username of the user to be authenticated. The default attribute is the `authentication.subject.id` attribute, which is typically used to store a username.

Password:
Enter the API Gateway message attribute that contains the password of the user to authenticate. The default message attribute is the `authentication.subject.password` attribute, which is typically used to store a password.

Credential Format:
Select the format of the credential stored in the API Gateway message attribute specified in the **Username** field above. By default, `User Name` is selected.

Repository Name:
Select an existing repository to authenticate the user against from the drop-down list. Alternatively, you can configure a new authentication repository by clicking the **Add** button. For more details on configuring the various types of repository supported by the API Gateway, see the *Authentication Repository* tutorial.

Authenticate API Key

Overview

API keys are supplied by client users and applications calling REST APIs to track and control how the APIs are used (for example, to meter access and prevent abuse or malicious attack). The **Authenticate API Key** filter enables you to securely authenticate an API key with the API Gateway. API keys include a key ID that identifies the client responsible for the API service request. This key ID is not a secret, and must be included in each request. API keys can also include a confidential secret key used for authentication, which should only be known to the client and to the API service. You can use the **Authenticate API Key** filter to specify where to find the API key ID and secret key in the request message, and to specify timestamp and expiry options.

An example use case for this filter would be a client accessing a REST API service to invoke specific methods (for example, `startVM()` or `stopVM()`). To invoke these methods, you are required to provide your API key ID and secret key to the API Gateway. You can keep the secret key private by sending the request over HTTPS. Alternatively, you can use the secret key to generate an HMAC digital signature. This means that the secret key is not sent in the request, but is inferred instead, because the message must have been signed using the required secret key. When the API service receives the request, it uses the API key ID to look up the corresponding secret key, and uses it to validate the signature and confirm the request sender.

The API Gateway supports the following API key types:

- Simple API keys including an key ID only. The API key ID is included in all requests to authenticate the client.
- Amazon Web Services style API keys including a key ID and a secret key, which are used together to securely authenticate the client. The API key ID is included in all requests to identify the client. The secret key is known only to the client and the API Gateway.

For more details on authenticating Amazon Web Services API keys, see
http://s3.amazonaws.com/doc/s3-developer-guide/RESTAuthentication.html

General Settings

Configure the following fields on this tab:

Name:
Enter a suitable name for this filter in your policy.

KPS Alias:
Enter the alias name of the Key Property Store (KPS) used to store the API keys. For more details, see *Key Property Stores*. Defaults to the example `ClientRegistry` supplied with the API Gateway. For details on storing API keys in the Oracle API Manager, see *Configuring and Managing OAuth 2.0*.

Field Containing Secret:
Enter the name of the field in the KPS that contains the secret. Defaults to `secretKey`.

API Key Settings

Configure the following fields on this tab:

Where to find API key:
To specify where to find the API key in the request message, select one of the following options:

- **API key is located in:**
  Select one of the following from the list:
  - `Query String`

  - `Header`
  - `Parameter`

  The default option is `Query String`. Enter the name in the text box. Defaults to `KeyId`.

- **API key is in Authorization header with format:**
  Select one of the following Authorization headers from the list:
  - `Amazon AWS s3 Authorization Header - "AWS apiKey + ":" + base64(signature)"`
  - `HTTP Basic Authentication Header - "Basic base64(apiKey:secret)"`

  Defaults to the `Amazon AWS s3 Authorization Header`.
- **API key can be found using the following selector:**
  Enter the selector value that specifies the location of the API key. For details on selectors, see *Selecting Configuration Values at Runtime*. Defaults to `${http.client.getCgiArgument("KeyId")}`.

Where to find Secret key:
To specify where to find the secret key in the request message, select the **Extract Secret** setting, and select one of the following options:

- **Secret key is in:**
  Select one of the following from the list:
  - `Query String`
  - `Header`
  - `Parameter`

  The default option is `Query String`. Enter the name in the text box. Defaults to `SecretKey`.
- **Secret key is in Authorization header with format:**
  Select the Authorization header from the list. Defaults to `HTTP Basic Authentication Header - "Basic base64(apiKey:secret)"`.
- **Secret key can be inferred from signature:**
  The client can use the secret key to generate a digital signature that is included in the request. When the API Gateway receives the request, it uses the API key ID to identify the client and look up the corresponding secret key in the Oracle API Manager. The secret key is then used to validate the signature and authenticate the client. To specify the signature format, select one of the following from the list:
  - `Amazon AWS s3 Authorization Header Authentication - "AWS apiKey + ":" + base64(signature)"`
  - `Amazon AWS s3 REST Authentication - "?Signature=<base64(signature)> &Expires=<seconds since epoch>&AWSAccessKeyId=<aws-id>"`

  Defaults to `Amazon AWS s3 Authorization Header Authentication`.
- **Secret key can be found using the following selector:**
  Enter the selector value that specifies the location of the secret key. For details on selectors, see *Selecting Configuration Values at Runtime*. Defaults to `${http.client.getCgiArgument("SecretKey")}`.

Authenticate API key and secret:
Select whether to authenticate both the API key ID and the secret key. This means that the client must supply the API key ID and the secret key in the request message. This setting is selected by default.

Advanced

Configure the following fields on this tab:

Validate Timestamp:
Select whether to validate the API key timestamp using the settings specified below. This setting is unselected by default.

Timestamp is located in:
To specify where the timestamp is located in the request message, select one of the following from the list:

- `Header`
- `Parameter`
- `Query String`

The default option is `Header`. Enter the name in the text box. Defaults to `Date`.

Timestamp format is:
To specify the timestamp format, select one of the following from the list:

- `Simple Date Format`
- `Milliseconds since epoch`
- `Seconds since epoch`

The default option is `Simple Date Format`. Enter the format in the text box. Defaults to `EEE, dd MMM yyyy HH:mm:ss zzz`.

Timestamp Drift +/-:
You can specify a drift time in milliseconds to allow differences in the clock times between the machine on which the API key was generated and the machine on which the API Gateway is running. Defaults to `+-60000` milliseconds (one minute).

Validate Expires:
Select whether to validate the API key expiry details using the settings specified below. This setting is unselected by default.

Expires is located in:
To specify the location of the expiry details in the request message, select one of the following from the list:

- `Query String`
- `Header`
- `Parameter`

The default option is `Query String`. Enter the name in the text box. Defaults to `Expires`.

Expires format is:
To specify the format of the expiry details, select one of the following from the list:

- `Milliseconds since epoch`
- `Seconds since epoch`
- `Simple Date Format`

The default option is `Milliseconds since epoch`. Enter the format in the text box.

Timestamp Drift +/-:
You can specify a drift time in milliseconds to allow differences in the clock times between the machine on which the API key was generated and the machine on which the API Gateway is running. Defaults to `60000` milliseconds (one minute).

CA SOA Security Manager Authentication

Overview

CA SOA Security Manager can authenticate end-users and authorize them to access protected Web resources. When the API Gateway receives a message containing user credentials, it can forward the message to CA SOA Security Manager where the passed credentials are extracted from the message to authenticate the end-user. When the message has been passed to CA SOA Security Manager, it can authenticate the user by the following methods:

- **XML Document Credential Collector:**
  Gathers credentials from the message and maps them to fields within a user directory.
- **XML Digital Signature:**
  Validates the X.509 certificate contained within an XML-Signature on the message.
- **WS-Security:**
  Extracts user credentials from WS-Security tokens contained in the message.
- **SAML Session Ticket:**
  Consumes a SAML session ticket from an HTTP header, SOAP envelope, or session cookie to authenticate the end-user.

By delegating the authentication decision to CA SOA Security Manager, the API Gateway acts as a Policy Enforcement Point (PEP). It *enforces* the decisions made by the CA SOA Security Manager, which acts a Policy Decision Point (PDP).

Please refer to the Authentication Methods section of the CA SOA Security Manager Policy Configuration Guide for more information on these authentication methods.

Enter a name for the filter in the **Name:** field before configuring the Agent and Message Details sections described below.

Prerequisites

CA SOA Security Manager integration requires CA TransactionMinder SDK version 6.0 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Agent Configuration

Name:
Enter a name for this authentication filter in the field provided.

Agent Name:
To act as a PEP for the CA SOA Security Manager, the API Gateway must have been set up as a *SOA Agent* with the Policy Server. For more details on how to do this, see the *CA SOA Security Manager Agent Configuration Guide*.

Click the button on the right to select a previously configured agent to connect to SOA Security Manager. This name *must* correspond with the name of an agent previously configured in the SOA Security Manager **Policy Server**. At runtime, the API Gateway connects as this agent to a running instance of SOA Security Manager.

To add an agent, right-click the **SiteMinder/SOA Security Manager Connections** tree node, and select **Add a SOA Security Manager Connection**. Alternatively, you can add SOA Security Manager connections under the **External Connections** node in the Policy Studio tree view. For details on how to configure SOA Security Manager connections, see the section called “SOA Security Manager Connection Details Only”.

Message Details Configuration

While authenticating the user against CA SOA Security Manager, the user can also be authorized for a specified action on a particular resource. Configure the following fields in the **Message Details** section:

Resource:
Enter the name of the resource for which you want to ensure that the user has access to. By default, the `http.request.uri` message attribute is used, which contains the relative path on which the request was received by the API Gateway.

Action:
Specify the action that the user is attempting to perform on the specified resource. The API Gateway will check the user's entitlements in CA SOA Security Manager to ensure that the user is allowed to perform this action on the resource entered above. By default, the `http.request.verb` message attribute is used, which stores the HTTP verb used by the client when sending up the message.

Protocol:
Enter the protocol used by the client to access the requested resource. Users can have different access rights depending on their roles in the organization. For example, managers may be allowed to FTP to a given resource, but more junior employees are only allowed to GET a resource using HTTP. Defaults to `http`.

Headers:
In order to carry out further authorization checks on the message, it is possible to forward the HTTP headers associated with the client message to the CA SOA Security Manager. By default, the `http.headers` message attribute is used to ensure that the original client headers are send to the CA SOA Security Manager.

XmlToolkit.properties File

The `XmlToolkit.properties` file contains default properties used by the SOA agent, such as the URL of the CA SOA Manager, an identifier for the SOA agent, and an indication to the SOA Manager if it should perform fine-grained resource identification or not. The `XmlToolkit.properties` file can be found in the `/lib/modules/soasm` directory of your API Gateway installation.

```
#Wed Jul 18 15:02:16 BST 2007
WSDMResourceIdentification=yes
WS_UT_CREATION_EXPIRATION_MINUTES=60
```

The following properties are available:

- `WSDMResourceIdentification:`
  This value cannot be configured from the Policy Studio, and so can only be set directly in the properties file. If this property is set to `no` (or if the properties file cannot be found) only a coarse-grained resource identification is performed on the requested URL. If this value is set to `yes`, a fine-grained resource identification including the reques-

ted URL, Web Service name, and SOAP operation (`[url]/[web service name]/[soap operation]`).

- `WS_UT_CREATION_EXPIRATION_MINUTES:`
  Specifies the WS-Username Token age limit restriction in minutes. This setting helps prevent against replay attacks. The default token age limit is 60 minutes. See the section below for more information on modifying this setting.

Configuring the Username and Password Digest Token Age Restriction:
By default, the WS-Security authentication scheme imposes a 60 minute restriction on the age of Username and Password Digest Tokens to protect against replay attacks.

You can configure a different value for the token age restriction for the API Gateway by setting the `WS_UT_CREATION_EXPIRATION_MINUTES` parameter in the `XmlToolkit.properties` file for that API Gateway. To configure the API Gateway to use a non-default age restriction for Username and Password Token authentication, complete the following steps:

1. Navigate to the `INSTALL_DIR/system/lib/modules/soasm` directory, where `INSTALL_DIR` points to the root of your API Gateway installation.
2. Open the `XmlToolkit.properties` file in a text editor.
3. Add the following line, where `token_age_limit` specifies the token age limit in minutes:
   `WS_UT_CREATION_EXPIRATION_MINUTES=token_age_limit`
4. Save and close the `XmlToolkit.properties` file.
5. Restart the API Gateway.

Important

It is important to note the following:

- The properties file is written to the `/lib/modules/soasm` directory when a SOA Security Manager Authentication or Authorization filter is loaded at startup, or on server refresh (for example, when a configuration update is deployed), but only if the file does not already exist in this location.
- If the properties file already exists in the `/lib/modules/soasm` directory, the **WSDMResourceIdentification** property is *not* overwritten. In other words, the user is allowed to manually set this property independently of the Policy Studio.
- If the **WSDMResourceIdentification** property does not exist, it is given a default value of `yes` and written to the file.

HTML Form-based Authentication

Overview

HTML Form-based Authentication enables users to supply their user name and password details in an HTML form, and submit them to login to a system. Using HTML form-based authentication, normal HTTP authentication features such as HTTP Basic or HTTP Digest are not used. Instead, the user name and password are typically sent as HTML `<FORM>` data in an HTTP `POST` over SSL.

When the **HTML Form-based Authentication** filter is configured, the API Gateway can authenticate the user details specified in the HTML form against a user profile stored in the API Gateway local repository, a database, or an LDAP directory. The **HTML Form-based Authentication** filter also enables you to specify how HTTP sessions are managed (for example, session expiry, and applicable API Gateway domain or relative path).

General Settings

These settings enable you to configure general details such as the names of the HTML form fields, format of user credentials, and repository to validate credentials against. Complete the following settings:

Name:
Enter an appropriate name for the filter.

Username:
Enter the name of the HTML form field in which the user enters their username. Defaults to `username`.

Password:
Enter the name of the HTML form field in which the user enters their password. Defaults to `password`.

Format of Authentication Credentials:
You must specify the format of the user credentials presented by the client because the API Gateway has no way of telling one credential format from another. Select one of the following from the list:

- `User Name`
- `Distinguished Name`

The selected format is then used internally by the API Gateway when performing authorization lookups against third-party Identity Management servers.

Validate Credentials against this Repository:
This specifies the name of the Authentication Repository where all user profiles are stored. This can be in the API Gateway's local repository, a database, or an LDAP directory. Select a pre-configured **Repository Name** from the drop-down list (for example, `Local User Store`).

You can add a new repository by right-clicking the appropriate node under **External Connections** -> **Authentication Repository Profiles** (for example, **Database Repositories**), and selecting **Add a new Repository**. For more details, see the *Authentication Repository* tutorial.

Session Settings

The session settings enable you to configure how HTTP sessions between the HTML form client and the API Gateway are managed. Complete the following settings:

Create a session:
Select whether to create an HTTP session. This setting is selected by default.

Expiry of session in milliseconds:
Enter the period of time in milliseconds before the session expires. Defaults to `600000` (10 minutes).

Session applicable for this domain:
Enter the API Gateway domain name to which the session applies (for example, `dmz`).

Session applicable for this path:
Enter the API Gateway relative path to which the session applies. Defaults to `/`.

Session sent over SSL only:
Select whether the session is sent over an SSL connection only. This setting is not selected by default.

HTTP Basic Authentication

Overview

A client can authenticate to the API Gateway with a username and password combination using *HTTP Basic Authentication*. When an **HTTP Basic Authentication** filter is configured, the API Gateway requests the client to present a username and password combination as part of the *HTTP Basic challenge-response* mechanism.

With HTTP Basic Authentication, the client's username and password are concatenated, base64-encoded, and passed in the `Authorization` HTTP header as follows:

```
Authorization: Basic dm9yZGVsOnZvcmRlbA==
```

The API Gateway can then authenticate this user against a user profile stored in the API Gateway's local repository, a database, or an LDAP directory. The realm presented in the challenge for HTTP Basic Authentication is the realm currently specified in the system settings. See the *Default Settings* topic for more information.

Configuration

The information specified on this screen informs the API Gateway where it can find user profiles for authentication purposes. The API Gateway can look up user profiles in the API Gateway's local repository, in a database, or in an LDAP directory. You can add Users to the local repository using the **Users** interface. For more details, see the *API Gateway Users* tutorial.

To configure the **HTTP Basic Authentication** filter, complete the following settings:

Name
Enter an appropriate name for the filter.

Credential Format
The username presented to the API Gateway during the HTTP Basic handshake can be of many formats, usually username or Distinguished Name (DName). Because the API Gateway has no way of inherently telling one format from another (for example, the client's username could be a DName), you must specify the format of the credential presented by the client. This format is then used internally by the API Gateway when performing authorization lookups against third-party Identity Management servers.

Allow Client Challenge
HTTP Basic Authentication can use the following approaches:

- **Direct Authentication**
  The client sends up the `Authorization` HTTP Basic Authentication header in its first request to the server.
- **Challenge-Response Handshake**
  The client does *not* send the `Authorization` header when sending its request to the server (it does not know that the server requires HTTP Basic Authentication). The server responds with an *HTTP 401 response code*, instructing the client to authenticate to the server by sending the `Authorization` header. The client then sends a second request, this time including the `Authorization` header and the relevant username and password.

The first case is used mainly for machine-to-machine transactions in which there is no human intervention. The second case is typical of situations where a browser is talking to a Web server. When the browser receives the HTTP 401 response to its initial request, it displays a dialog to enable the user to enter the username and password combination.

If you wish to force clients to always send the HTTP Basic `Authorization` header to the API Gateway, deselect the **Allow client challenge** checkbox. If you wish to allow clients to engage in the HTTP Basic Authentication challenge-response handshake with the API Gateway, ensure this feature is enabled by selecting this option.

Allow Retries
Select this option to allow the user to retry their username/password in the browser when an HTTP 401 response code is received (for example, if authentication fails, or is not yet provided). The number of times that the browser displays the username/password dialog when an HTTP 401 is received is controlled by the browser (usually three times). This setting is not selected by default.

Remove HTTP Authentication Header
Select this option to remove the HTTP `Authorization` header from the downstream message. If this option is not selected, the incoming `Authorization` header is forwarded on to the destination Web Service.

Repository Name
This specifies the name of the Authentication Repository where all user profiles are stored. This can be the API Gateway's local repository, a database, or an LDAP directory. Select a pre-configured **Repository Name** from the drop-down list.

You can add a new repository in the tree on the left under the **External Connections** node. Right-click the appropriate node under **Authentication Repository Profiles** (for example, **Database Repositories**), and select **Add a new Repository**. For more details, see the *Authentication Repository* tutorial.

HTTP Digest Authentication

Overview

A client can authenticate to the API Gateway with a username and password digest using *HTTP Digest Authentication*. When an **HTTP Digest Authentication** filter is configured, the API Gateway requests the client to present a username and password digest as part of the *HTTP Digest challenge-response* mechanism. The API Gateway can then authenticate this user against a user profile stored in the API Gateway database.

The realm presented in the challenge for HTTP Digest Authentication is the realm currently specified in the system settings. For more details, see the *Default Settings* topic.

Configuration

The information specified on this screen informs the API Gateway where it can find user profiles for authentication purposes. The API Gateway can lookup user profiles in the API Gateway's local repository, in a database, or in an LDAP directory. Users can be added to the local repository using the **Users** interface. For more details, see the *API Gateway Users* tutorial.

To configure the **HTTP Digest Authentication** filter, complete the following settings:

Name
Enter an appropriate name for the filter.

Credential Format
The username presented to the API Gateway during the HTTP Digest handshake can be of many formats, usually username or Distinguished Name (DName). Because the API Gateway has no way of inherently telling one format from the other (for example, the client's username could be a DName), it is necessary to specify the format of the credential presented by the client. This format is then used internally by the API Gateway when performing authorization lookups against third party Identity Management servers.

Session Timeout
As part of the *HTTP Digest Authentication* protocol, the API Gateway must generate a *nonce* (number used once) value, and send it to the client. The client uses this nonce to create the digest of the username and password. However, it should only be allowed a certain amount of time to do so. The **Session Timeout** field specifies the length of time (in milliseconds) for which the nonce is valid.

Allow Retries
Select this option to allow the user to retry their username/password in the browser when an HTTP 401 response code is received (for example, if authentication fails, or is not yet provided). The number of times that the browser displays the username/password dialog when an HTTP 401 is received is controlled by the browser (usually three times). This setting is not selected by default.

Remove HTTP Authentication Header
Select this option to remove the HTTP `Authorization` header from the downstream message. If this option is not selected, the incoming `Authorization` header is forwarded on to the destination Web Service.

Repository Name
This specifies the name of the Authentication Repository where all user profiles are stored. This can be in the API Gateway's local repository, in a database, or in an LDAP directory. Select a pre-configured **Repository Name** from the drop-down list.

You can add a new repository in the tree on the left under the **External Connections** node. Right-click the appropriate node under **Authentication Repository Profiles** (for example, **Database Repositories**), and select **Add a new Repository**. For more details, see the *Authentication Repository* tutorial.

HTTP Header Authentication

Overview

You can use the **HTTP Header** filter in cases where the API Gateway receives end-user authentication credentials in an HTTP header. A typical scenario would see the end-user (or message originator) authenticating to an intermediary. The intermediary authenticates the end-user, and to propagate the end-user credentials to the destination Web Service, the intermediary inserts the credentials into an HTTP header and forwards them onwards.

When the API Gateway receives the message, it performs the following tasks:

- Authenticate the sender of the message (the intermediary)
- Extract the *end-user* identity from the token in the HTTP header for use in subsequent Authorization filters

Important

In the case outlined above, the API Gateway does *not* attempt to re-authenticate the end-user. It trusts that the intermediary has already authenticated the end-user, and so the API Gateway does not authenticate the user again. However, it is good practice to authenticate the message sender (the intermediary). Any subsequent Authorization filters use the end-user credentials that were passed in the HTTP header.

Configuration

The following configuration fields are available on this screen:

Name:
Enter an appropriate name for this filter in the **Name** field.

HTTP Header Name:
Enter the name of the HTTP Header that contains the end-user credentials.

HTTP Header Type:
Select the type of credentials that are passed in the named HTTP Header. The following types are supported:

1. X.509 Distinguished Name
2. Certificate
3. Username

IP Address

Overview

You can configure the API Gateway to allow or deny machines, or groups of machines, access to resources based on IP address. The main table on the screen shows the IP addresses from which the API Gateway accepts or denies messages depending on what is configured.

The **IP Address** Authentication filter uses the value stored in the `http.request.clientaddr` message attribute to determine whether or not to allow or deny access. This message attribute contains the remote host address from the TCP socket used in the connection between the client and the API Gateway.

Configuration

The following fields must be configured:

Name:
Enter a name for the filter.

IP Addresses:
You can add IP addresses by clicking the **Add** button, which displays the **Add IP Filter** dialog. Enter an **IP Address** and **Subnet Mask** to indicate a network to filter.

Messages sent from hosts belonging to this network will be accepted or rejected based on what is configured in the section below. A **Subnet Mask** of `255.255.255.255` can be used to filter specific IP addresses. For more details, see Configuring Subnet Masks.

Important

If requests are made across a proxy, portal, or other such intermediary, the API Gateway filters on the IP address of the intermediary. Therefore, you should enter the IP address of the intermediary on this screen, and not that of the user/client machine.

You can edit and remove existing IP addresses by selecting the **Edit** and **Remove** buttons.

Access:
Depending on whether the **Allow Access** or **Deny Access** radio button is checked, the IP addresses listed in the table are allowed or denied access to the Web Service.

Configuring Subnet Masks

An IP address is normally represented by a string of 4 numbers separated by periods (for example, `192.168.0.20`. Each number is normally represented as the decimal equivalent of an eight-bit binary number, which means that each number may take any value between 0 (all eight bits cleared) and 255 (all eight bits set).

A *subnet mask* (or netmask) is also a set of four number blocks separated by periods, each of which has a value in the range 0-255. Every IP address consists of two parts: the network address and the host number. The netmask is used to determine the size of these two parts. The positions of the bits set in the netmask represent the space reserved for the network address, while the bits that are cleared represent the space reserved for the host number. The netmask determines the range of IP addresses.

The following examples illustrate how netmasks work in practice:

Example 1: Specifying a Range of IP Addresses:
You only want to allow requests from the following IP addresses:

192.168.0.16, 192.168.0.17, 192.168.0.18, and 192.168.0.19.
Use the following address/netmask combination to cover the 4 IP addresses listed above:
192.168.0.16/255.255.255.252

In more detail, the binary representation of the netmask is as follows:
11111111.11111111.11111111.11111100
The top 30 bits of the netmask indicate the network and the last 2 bits refer to the host on the network. These last 2 bits allow 4 different addresses as shown in the worked example below.

When the API Gateway receives a request from a certain IP address, the API Gateway performs a logical AND on the client IP address and the configured netmask. It also does a logical AND with the IP address entered in the IP Address filter and the configured subnet mask. If the AND-ed binary values are the same, the request from the IP address can be considered in the same network range as that configured in the filter.

The following worked example illustrates the mechanics of the IP address filtering. It assumes that you have entered the following in the IP Address and Netmask fields in the IP Address filter:

Field	***Value***
IP Address	192.168.0.16
Net Mask	255.255.255.252

```
Step 1:  AND the IP address and Netmask configured in the IP Address Filter:
11000000.10100000.00000000.00010000 (192.168.0.16)
AND
11111111.11111111.11111111.11111100 (255.255.255.252)
=========================================
11000000.10100000.00000000.00010000

Step 2: Request is received from 192.168.0.18:
11000000.10100000.00000000.00010010 (192.168.0.18)
AND
11111111.11111111.11111111.11111100 (255.255.255.252)
=========================================
11000000.10100000.00000000.00010000
===> AND-ed value is equal to the result for 192.168.0.16.
===> Therefore the client IP address is inside the configured range.

Step 3: Request is received from 192.168.0.20:
11000000.10100000.00000000.00010100 (192.168.0.20)
AND
11111111.11111111.11111111.11111100 (255.255.255.252)
=========================================
11000000.10100000.00000000.00010100
===> AND-ed value is NOT equal to the result for 192.168.0.16.
===> Therefore the client IP address is NOT inside the configured range.
```

Example 2: Specifying an Exact IP Address:
You can also specify an exact IP address by using a netmask of 255.255.255.255. When this netmask is used, only requests from this client IP address is allowed or blocked, depending on what is configured in the filter. This example assumes that the following details have been configured in the IP Address filter:

Field	***Value***
IP Address	192.168.0.36
Net Mask	255.255.255.255

```
Step 1:  AND the IP address and Netmask configured in the IP Address Filter:
11000000.10100000.00000000.00100100 (192.168.0.36)
AND
11111111.11111111.11111111.11111111 (255.255.255.255)
=========================================
11000000.10100000.00000000.00100100

Step 2: Request is received from client with IP address of 192.168.0.37:
11000000.10100000.00000000.00100101 (192.168.0.37)
AND
11111111.11111111.11111111.11111111 (255.255.255.255)
=========================================
11000000.10100000.00000000.00100101
===> AND-ed value is NOT equal to the result for 192.168.0.36
===> Therefore the client IP address is NOT inside the configured range.
```

SAML Authentication

Overview

A Security Assertion Markup Language (SAML) *authentication assertion* is issued as proof of an authentication event. Typically, an end-user authenticates to an intermediary, who generates a SAML authentication assertion to prove that it has authenticated the user. The intermediary inserts the assertion into the message for consumption by a downstream Web Service.

When the API Gateway receives a message containing a SAML authentication assertion, it does not attempt to authenticate the end-user again. Instead, it authenticates the sender of the assertion (the intermediary) to ensure that only the intermediary could have issued the assertion, and then validates the authentication details contained in the assertion. Therefore, the API Gateway performs the following tasks in this scenario:

- Authenticates the sender of the message (the intermediary)
- Extracts the end-user's identity from the authentication assertion and validates the authentication details

The **SAML Authentication** filter performs the second task. A separate authentication filter must be placed before this filter in the policy to authenticate the sender of the assertion. The end-user's identity is used in any subsequent authorization filters.

The following sample SOAP message contains a SAML authentication assertion:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
 <soap-env:Header xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <wsse:Security>
   <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
                        AssertionID="oracle-1056477425082"
                        Id="oracle-1056477425082"
                        IssueInstant="2003-06-24T17:57:05Z"
                        Issuer="CN=Sample User,....,C=IE"
                        MajorVersion="1"
                        MinorVersion="0">
      <saml:Conditions
            NotBefore="2003-06-20T16:20:10Z"
            NotOnOrAfter="2003-06-20T18:20:10Z"/>
      <saml:AuthenticationStatement
                        AuthenticationInstant="2003-06-24T17:57:05Z"
            AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
        <saml:SubjectLocality IPAddress="192.168.0.32"/>
        <saml:Subject>
          <saml:NameIdentifier
            Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
                    sample
          </saml:NameIdentifier>
        </saml:Subject>
      </saml:AuthenticationStatement>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
                       id="Sample User">
      <dsig:SignedInfo>
        .....
      </dsig:SignedInfo>
      <dsig:SignatureValue>
        rpa/......0g==
      </dsig:SignatureValue>
      <dsig:KeyInfo>
        .....
      </dsig:KeyInfo>
    </dsig:Signature>
   </saml:Assertion>
  </wsse:Security>
 </soap-env:Header>
<soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
  </soap-env:Body>
</soap-env:Envelope>
```

General Settings

Configure the following field:

Name:
Enter an appropriate name for the filter.

Details

Configure the following fields on the **Details** tab:

SOAP Actor/Role:
If you expect the SAML assertion to be embedded in a WS-Security block, you can identify this block by specifying the SOAP Actor or Role of the WS-Security header that contains the assertion.

XPath Expression:
Alternatively, if the assertion is not contained in a WS-Security block, you can enter an XPath expression to locate the authentication assertion. You can configure XPath expressions using the **Add Edit** and **Delete** buttons.

SAML Namespace:
Select the SAML namespace that must be used on the SAML assertion for this filter to succeed. If you do not wish to check the namespace, select the `Do not check version` option from the drop-down list.

SAML Version:
Specify the SAML Version that the assertion must adhere to by entering the major version in the first field, and the minor version in the second field. For example, for SAML 2.0, enter 2 in the first field and `0` in the second field.

Drift Time:
The *drift time*, specified in seconds, is used when checking the validity dates on the authentication assertion. The drift time allows for differences between the clock times of the machine on which the assertion was generated and the machine hosting the API Gateway.

Remove Enclosing WS-Security Element on Successful Validation:
Select this checkbox if you wish to remove the WS-Security block that contains the SAML assertion after the assertion has been successfully validated.

Trusted Issuers

You can use the table on this tab to select the issuers that you consider trusted. In other words, this filter only accepts assertions that have been issued by the SAML Authorities selected here.

Click the **Add** button to display the **Trusted Issuers** screen. Select the Distinguished Name of a SAML Authority whose certificate has been added to the Certificate Store, and click **OK**. Repeat this step to add more SAML Authorities to the list of trusted issuers.

SAML PDP Authentication

Overview

The API Gateway can request an authentication decision from a Security Assertion Markup Language (SAML) Policy Decision Point (PDP) for an authenticated client using the SAML Protocol (SAMLP). In such cases, the API Gateway presents evidence to the PDP in the form of some user credentials, such as the Distinguished Name of a client's X.509 certificate.

The PDP decides whether to authenticate the end-user. It then creates an authentication assertion, signs it, and returns it to the API Gateway in a SAML Protocol response. The API Gateway can then perform a number of checks on the response, such as validating the PDP signature and certificate, and examining the assertion. It can also insert the SAML authentication assertion into the message for consumption by a downstream Web Service.

Request Configuration

This section describes how the API Gateway should package the SAMLP request before sending it to the SAML PDP.

SAML PDP URL Sets
You can configure a group of SAML PDPs to which the API Gateway connects in a round-robin fashion if one or more of the PDPs are unavailable. This is known as a SAML PDP URL Set. You can configure a SAML PDP URL Set using this screen or under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring URL Groups*.

You can configure the following general fields:

- **SAML PDP URL Set:**
  Click the button on the right, and select a previously configured SAML PDP URL Set in the tree. To add a URL Set, right-click the **SAML PDP URL Sets** tree node, and select **Add a URL Set**. Alternatively, you can configure a SAML PDP URL Set under the **External Connections** node in the Policy Studio tree.
- **SOAPAction:**
  Enter the SOAP Action required to send SAML Protocol requests to the PDP. Click the **Use Default** button to use the following default SOAP Action as specified by the SAML Protocol:
  http://www.oasis-open.org/committees/security
- **SAML Version:**
  Select the SAML version to use in the SAMLP request.
- **Signing Key:**
  If the SAMLP request is to be signed, click the **Signing Key** button, and select the appropriate signing key from the Certificate Store.

SAML Subject:
The specified details describe the *subject* of the SAML assertion. Complete the following fields:

- **Subject Attribute:**
  Select the message attribute that contains the name of an authenticated user name. By default, the `authentication.subject.id` message attribute is selected, which contains the user name of the authenticated user.
- **Subject Format:**
  Select the format of the message attribute selected in the **Subject Attribute** field above. You do not need to select a format if the **Subject Attribute** field is set to `authentication.subject.id`

Subject Confirmation:
The settings on the **Confirmation Method** tab determine how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the

<SubjectConfirmation> block can be used to authenticate the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical <SubjectConfirmation> block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

You must configure the following fields on the **Subject Confirmation** tab:

Method:
The selected value determines the value of the <ConfirmationMethod> element. The following table shows the available methods, their meanings, and their respective values in the <ConfirmationMethod> element:

Method	***Meaning***	***Value***
Holder Of Key	Inserts a <SubjectConfirmation> into the SAMLP request. The <SubjectConfirmation> contains a <dsig:KeyInfo> section with the certificate of the user selected to sign the SAMLP request. The user selected to sign the SAMLP request must be the authenticated subject (authentication.subject.id). Select the **Include Certificate** option if the signer's certificate is to be included in the SubjectConfimration block. Alternatively, select the **Include Key Name** option if only the key name is to be included.	urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
Bearer	Inserts a <SubjectConfirmation> into the SAMLP request.	urn:oasis:names:tc:SAML:1.0:cm:bearer
SAML Artifact	Inserts a <SubjectConfirmation> into the SAMLP request.	urn:oasis:names:tc:SAML:1.0:cm:artifact
Sender Vouches	Inserts a <SubjectConfirmation> into the SAMLP request. A user must sign the SAMLP request.	urn:oasis:names:tc:SAML:1.0:cm:bearer

If the **Method** field is left blank, no <ConfirmationMethod> block is inserted into the assertion.

Include Certificate:
Select this option if you wish to include the SAML subject's certificate in the <KeyInfo> section of the

`<SubjectConfirmation>` block.

Include Key Name:
Alternatively, if you do not want to include the certificate, select this option to only include the key name in the `<KeyInfo>` section.

Response Configuration

This tab configures the SAMLP Response returned from the SAML PDP. The following fields are available:

SOAP Actor/Role:
If the SAMLP response from the PDP contains a SAML authentication assertion, the API Gateway can extract it from the response and insert it into the downstream message. The SAML assertion is inserted into the WS-Security block identified by the specified SOAP actor/role.

Drift Time:
The SAMLP request to the PDP is timestamped by the API Gateway. To account for differences in the times on the machines running the API Gateway and the SAML PDP the specified time is subtracted from the time at which the API Gateway generates the SAMLP request.

Insert SAML Authentication Assertion

Overview

After successfully authenticating a client, the API Gateway can insert a SAML (Security Assertion Markup Language) authentication assertion into the SOAP message. Assuming all other security filters in the policy are successful, the assertion will eventually be consumed by a downstream Web Service.

It may be useful to refer to the following example of a signed SAML authentication assertion when configuring this filter:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
<wsse:Security>
 <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
      AssertionID="oracle-1056477425082"
      Id="oracle-1056477425082"
      IssueInstant="2003-06-24T17:57:05Z"
      Issuer="CN=Sample User,....,C=IE"
      MajorVersion="1"
      MinorVersion="0">
    <saml:Conditions
      NotBefore="2003-06-20T16:20:10Z"
      NotOnOrAfter="2003-06-20T18:20:10Z"/>
    <saml:AuthenticationStatement
      AuthenticationInstant="2003-06-24T17:57:05Z"
      AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
      <saml:SubjectLocality IPAddress="192.168.0.32"/>
      <saml:Subject>
        <saml:NameIdentifier
      Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
                  sample
        </saml:NameIdentifier>
      </saml:Subject>
    </saml:AuthenticationStatement>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
                     id="Sample User">
    <dsig:SignedInfo>
      .....
    </dsig:SignedInfo>
    <dsig:SignatureValue>
      rpa/......0g==
    </dsig:SignatureValue>
    <dsig:KeyInfo>
      .....
    </dsig:KeyInfo>
  </dsig:Signature>
 </saml:Assertion>
</wsse:Security>
</soap-env:Header>
<soap-env:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
</ns1:getTime>
</soap-env:Body>
</soap-env:Envelope>
```

General Configuration

Configure the following field:

Name:

Enter an appropriate name for the filter.

Assertion Details

Configure the following fields on the **Assertion Details** tab:

Issuer Name:
Select the certificate containing the Distinguished Name (DName) that you want to use as the Issuer of the SAML assertion. This DName is included in the SAML assertion as the value of the `Issuer` attribute of the `<saml:Assertion>` element. For an example, see the sample SAML assertion above.

Expire In:
Specify the lifetime of the assertion in this field. The lifetime of the assertion lasts from the time of insertion until the specified amount of time has elapsed.

Drift Time:
The **Drift Time** is used to account for differences in the clock times of the machine hosting the API Gateway (that generate the assertion) and the machines that consume the assertion. The specified time is subtracted from the time at which the API Gateway generates the assertion.

SAML Version:
You can create SAML 1.0, 1.1, and 2.0 attribute assertions. Select the appropriate version from the drop-down list.

Important

SAML 1.0 recommends the use of the `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` XML Signature Canonicalization algorithm. When inserting signed SAML 1.0 assertions into XML documents, it is quite likely that subsequent signature verification of these assertions will fail. This is due to the side effect of the algorithm including inherited namespaces into canonical XML calculations of the inserted SAML assertion that were not present when the assertion was generated.

For this reason, Oracle recommend that SAML 1.1 or 2.0 is used when signing assertions as they both uses the exclusive canonical algorithm `http://www.w3.org/2001/10/xml-exc-c14n#`, which safeguards inserted assertions from such changes of context in the XML document. Please see section 5.4.2 of the `oasis-sstc-saml-core-1.0.pdf` and section 5.4.2 of `sstc-saml-core-1.1.pdf` documents, both of which are available at `http://www.oasis-open.org`.

Assertion Location

The options on the **Assertion Location** tab specify where the SAML assertion is inserted in the message. By default, the SAML assertion is added to the WS-Security block with the current SOAP actor/role. The following options are available:

Append to Root or SOAP Header:
Appends the SAML assertion to the message root for a non-SOAP XML message, or to the SOAP Header for a SOAP message. For example, this option may be suitable for cases where this filter may process SOAP XML messages or non-SOAP XML messages.

Add to WS-Security Block with SOAP Actor/Role:
Adds the SAML assertion to the WS-Security block with the specified SOAP actor (SOAP 1.0) or role (SOAP 1.1). By default, the assertion is added with the current SOAP actor/role only, which means the WS-Security block with no actor. You can select a specific SOAP actor/role when available from the drop-down list.

XPath Location:
If you wish to insert the SAML assertion at an arbitrary location in the message, you can use an XPath expression to specify the exact location in the message. You can select XPath expressions from the drop-down list. The default is the `First WSSE Security Element`, which has an XPath expression of `//wsse:Security`. You can add, edit, or remove expressions by clicking the relevant button. For more details, see the *Configuring XPath Expressions* topic.

You can also specify how exactly the SAML assertion is inserted using the following options:

- **Append to node returned by XPath expression** (the default)
- **Insert before node returned by XPath expression**
- **Replace node returned by XPath expression**

Insert into Message Attribute:
Specify a message attribute to store the SAML assertion from the drop-down list (for example, `saml.assertion`). Alternatively, you can also enter a custom message attribute in this field (for example, `my.test.assertion`). The SAML assertion can then be accessed downstream in the policy.

Subject Confirmation Method

The settings on the **Subject Confirmation Method** tab determine how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the `<SubjectConfirmation>` block can be used to authenticate the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical `<SubjectConfirmation>` block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

The following configuration fields are available on the **Subject Confirmation Method** tab:

Method:
The selected value determines the value of the `<ConfirmationMethod>` element. The following table shows the available methods, their meanings, and their respective values in the `<ConfirmationMethod>` element:

Method	***Meaning***	***Value***
Holder Of Key	The API Gateway includes the key used to prove that the API Gateway is the holder of the key, or it includes a reference to the key.	`urn:oasis:names:tc:SAML:1.0:cm:holder-of-key`
Bearer	The subject of the assertion is the bearer of the assertion.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`
SAML Artifact	The subject of the assertion is the user that presented a SAML Artifact to the API Gateway.	`urn:oasis:names:tc:SAML:1.0:cm:artifact`
Sender Vouches	Use this confirmation method to assert that the API Gateway is acting on behalf of the authenticated end-user. No	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	***Meaning***	***Value***
	other information relating to the context of the assertion is sent. It is recommended that both the assertion **and** the SOAP Body must be signed if this option is selected. These message parts can be signed by using the *XML Signature Generation* filter.	

Note

You can also leave the **Method** field blank, in which case no `<ConfirmationMethod>` block is inserted into the assertion.

Holder-of-Key Configuration:
When you select `Holder-of-Key` as the SAML subject confirmation in the `Method` field, you must configure how information about the key is included in the message. There are a number of configuration options available depending on whether the key is a symmetric or asymmetric key.

Asymmetric Key:
If you want to use an asymmetric key as proof that the API Gateway is the holder-of-key entity, you must select the **Asymmetric Key** radio button and then configure the following fields on the **Asymmetric** tab:

- **Certificate from Store:**
  If you want to select a key that is stored in the Certificate Store, select this option and click the **Signing Key** button. On the **Select Certificate** screen, select the box next to the certificate that is associated with the key that you want to use.
- **Certificate from Selector Expression:**
  Alternatively, the key may have already been used by a previous filter in the policy (for example, to sign a part of the message). In this case, the key can be retrieved using the selector expression entered in this field. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Symmetric Key:
If you want to use a symmetric key as proof that the API Gateway is the holder of key, select the **Symmetric Key** radio button, and configure the fields on the **Symmetric** tab:

- **Generate Symmetric Key, and Save in Message Attribute:**
  If you select this option, the API Gateway generates a symmetric key, which is included in the message before it is sent to the client. By default, the key is saved in the `symmetric.key` message attribute.
- **Symmetric Key Selector Expression:**
  If a previous filter (for example, a **Sign Message** filter) has already used a symmetric key, you can reuse this key as proof that the API Gateway is the holder-of-key entity. Enter the name of the selector expresion (for example, message attribute) in the field provided, which defaults to `${symmetric.key}`. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.
- **Encrypt using Certificate from Certificate Store:**
  When a symmetric key is used, you must assume that the recipient has no prior knowledge of this key. It must, therefore, be included in the message so that the recipient can validate the key. To avoid meet-in-the-middle style attacks, where a hacker could eavesdrop on the communication channel between the API Gateway and the recipient

and gain access to the symmetric key, the key must be encrypted so that only the recipient can decrypt the key. One way of doing this is to select the recipient's certificate from the Certificate Store. By encrypting the symmetric key with the public in the recipient's certificate, the key can only be decrypted by the recipient's private key, to which only the recipient has access. Select the **Signing Key** button, and select the recipient's certificate on the **Select Certificate** dialog.

- **Encrypt using Certificate from Message Attribute:**
  Alternatively, if the recipient's certificate has already been used (perhaps to encrypt part of the message) this certificate is stored in a message attribute. You can enter this message attribute in this field.
- **Symmetric Key Length:**
  Enter the length (in bits) of the symmetric key to use.
- **Key Wrap Algorithm:**
  Select the algorithm to use to encrypt (*wrap*) the symmetric key.

Key Info:
The **Key Info** tab must be configured regardless of whether you have elected to use symmetric or asymmetric keys. It determines how the key is included in the message. The following options are available:

- **Do Not Include Key Info:**
  Select this option if you do not wish to include a <KeyInfo> section in the SAML assertion.
- **Embed Public Key Information:**
  If this option is selected, details about the key are included in a `<KeyInfo>` block in the message. You can include the full certificate, expand the public key, include the distinguished name, and include a key name in the `<KeyInfo>` block by selecting the appropriate boxes. When selecting the **Include Key Name** field, you must enter a name in the **Value** field, and then select the **Text Value** or **Distinguished Name Attribute** radio button, depending on the source of the key name.
- **Put Certificate in Attachment:**
  Select this option to add the certificate as an attachment to the message. The certificate is then referenced from the `<KeyInfo>` block.
- **Security Token Reference:**
  The Security Token Reference (STR) provides a way to refer to a key contained within a SOAP message from another part of the message. It is often used in cases where different security blocks in a message use the same key material and it is considered an overhead to include the key more than once in the message.
  When this option is selected, a `<wsse:SecurityTokenReference>` element is inserted into the `<KeyInfo>` block. It references the key material using a `URI` to point to the key material and a `ValueType` attribute to indicate the type of reference used. For example, if the STR refers to an encrypted key, you should select `EncryptedKey` from the dropdown, whereas if it refers to a `BinarySecurityToken`, you should select `X509v3` from the dropdown. Other options are available to enable more specific security requirements.

Advanced

Select Required Layout Type:
WS-Policy and SOAP Message Security define a set of rules that determine the layout of security elements that appear in the WS-Security header within a SOAP message. The SAML assertion will be inserted into the WS-Security header according to the layout option selected here. The available options correspond to the WS-Policy Layout assertions of `Strict`, `Lax`, `LaxTimestampFirst`, and `LaxTimestampLast`.

Insert SAML Attribute Statement:
You can insert a SAML attribute statement into the generated SAML authentication assertion. If you select this option, a SAML attribute assertion is generated using attributes stored in the `attribute.lookup.list` message attribute and subsequently inserted into the assertion. The `attribute.lookup.list` attribute must have been populated previously by an attribute lookup filter for the attribute statement to be generated successfully.

Indent:
Select this method to ensure that the generated signature is properly indented.

Security Token Reference:
The generated SAML authentication assertion can be encapsulated within a `<SecurityTokenReference>` block. The following example demonstrates this:

```
<soap:Header>
 <wsse:Security
     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
     soap:actor="oracle">
  <wsse:SecurityTokenReference>
   <wsse:Embedded>
    <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
                    AssertionID="Id-00000109fee52b06-0000000000000012"
                    IssueInstant="2006-03-15T17:12:45Z"
                    Issuer="oracle" MajorVersion="1" MinorVersion="0">
     <saml:Conditions NotBefore="2006-03-15T17:12:39Z"
                      NotOnOrAfter="2006-03-25T17:12:39Z"/>
      <saml:AuthenticationStatement
          AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
          AuthenticationInstant="2006-03-15T17:12:45Z">
       <saml:Subject>
        <saml:NameIdentifier Format="Oracle-Username-Password">
            admin
        </saml:NameIdentifier>
       <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>
            urn:oasis:names:tc:SAML:1.0:cm:artifact
        </saml:ConfirmationMethod>
       </saml:SubjectConfirmation>
      </saml:Subject>
     </saml:AuthenticationStatement>
    </saml:Assertion>
   </wsse:Embedded>
  </wsse:SecurityTokenReference>
 </wsse:Security>
</soap:Header>
```

To add the SAML assertion to a `<SecurityTokenReference>` block as in the example above, select the **Embed SAML assertion within Security Token Reference** option. Otherwise, select **No Security Token Reference**.

Insert Timestamp

Overview

In any secure communications protocol, it is crucial that secured messages do not have an indefinite life span. In secure Web Services transactions, a WS-Utility (WSU) Timestamp can be inserted into a WS-Security Header to define the lifetime of the message in which it is placed. A message containing an expired timestamp should be rejected immediately by any Web Service that consumes the message.

Typically, the timestamp contains `Created` and `Expires` times, which combine to define the lifetime of the timestamp. The following shows an example `Timestamp`:

```
<wsu:Timestamp xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
  <wsu:Created>2009-03-16T16:32:22Z</wsu:Created>
  <wsu:Expires>2009-03-16T16:42:22Z</wsu:Expires>
</wsu:Timestamp>
```

Because the WS-Utility Timestamp is inserted into the WS-Security header block, it is also referred to as a WSS Timestamp. For example, see the *Extract WSS Timestamp* filter.

Configuration

Complete the following fields to configure the API Gateway to insert a timestamp into the message:

Name:
Enter an intuitive name for the filter.

Actor:
The timestamp is inserted into the WS-Security header identified by the SOAP Actor selected here.

Expires In:
Configure the lifetime of the timestamp (and hence the message into which the timestamp is inserted) by specifying the expiration time of the assertion. The expiration time is expressed in days, hours, minutes, and/or seconds.

Layout Type:
In cases where the timestamp must adhere to a particular layout as mandated by the WS-Policy `<Layout>` assertion, you must select the appropriate layout type. A Web Service that enforces a WS-Policy may reject the message if the layout of security elements in the SOAP header is incorrect. Therefore, you must ensure that you select the correct layout type.

Insert WS-Security Username Token

Overview

When a client has been successfully authenticated, the API Gateway can insert a *WS-Security Username Token* into the downstream message as proof of the authentication event. The `<wsse:UsernameToken>` token enables a user's identity to be inserted into the XML message so that it can be propagated over a chain of Web Services.

A typical example would see a user authenticating to the API Gateway using HTTP Digest Authentication. After successfully authenticating the user, the API Gateway inserts a WS-Security Username Token into the message and digitally signs it to prevent anyone from tampering with the token.

The following example shows the format of the `<wsse:UsernameToken>` token:

```
<wsse:UsernameToken wsu:Id="oracle"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
  <wsu:Created>2006.01.13T-10:42:43Z</wsu:Created>
  <wsse:Username>oracle</wsse:Username>
  <wsse:Nonce EncodingType="UTF-8">
      KFIy9LgzhmDPNiqU/B9ZiWKXfEVNvFyn6KWYP+1zVt8=
  </wsse:Nonce>
  <wsse:Password Type="wsse:PasswordDigest">
      CxWj1OMnYj7dddMnU/DrOhyY3j4=
  </wsse:Password>
</wsse:UsernameToken>
```

This topic explains how to configure the API Gateway to insert a WS-Security Username Token after successfully authenticating a user.

General Configuration

To configure general settings, complete the following fields:

Name:
Enter an appropriate name for the filter.

Actor:
The Username Token is inserted into the WS-Security block identified by the specified SOAP *Actor*.

Credential Details

To configure the credential details, complete the following fields:

Username:
Enter the name of the user included in the Username Token. By default, the `authentication.subject.id` message attribute is stored, which contains the name of an authenticated user.

Include Nonce:
Select this option if you wish to include a nonce in the Username Token. A nonce a random number that is typically used to help prevent replay attacks.

Include Password:
Select this option if you wish to include a password in the Username Token.

Password:
If the **Include Password** checkbox is selected, the API Gateway inserts the user's password into the generated WS-Security Username Token. It can insert **Clear** or **SHA1 Digest** version of the password, depending on which radio button

you select. Oracle recommends the digest form of the password to avoid potential eavesdropping.

You can either explicitly enter the password for this user in the **Password** field, or use a message attribute by selecting the **Wildcard** option, and entering the message attribute in the field provided. By default, the `authentication.subject.password` attribute is used, which contains the password used by the user to authenticate to the API Gateway.

Advanced

To configure advanced settings, complete the following field:

Indent:
Select this option to add indentation to the generated `UsernameToken` and `Signature` blocks. This makes the security tokens more human-readable.

Kerberos Client Authentication

Overview

You can configure the API Gateway to act as a Kerberos Client by obtaining a service ticket for a specific Kerberos Service. The service ticket makes up part of the Kerberos client-side token that is injected into a SOAP message and then sent to the Service. If the service can validate the token, the client will be authenticated successfully.

You can also configure a **Connection** filter (from the Routing category) to authenticate to a Kerberos Service by inserting a client-side Kerberos token into the `Authorization` HTTP header.

Therefore, you should use the **Connection** filter if you wish to send the client-side Kerberos token in an HTTP header to the Kerberos Service. You should use the **Kerberos Client Authentication** filter if you wish to send the client-side Kerberos token in a `BinarySecurityToken` block in the SOAP message. For more information on authenticating to a Kerberos Service using a client-side Kerberos token, see the topic on the *Connection* filter.

The **Kerberos Client Authentication** filter is available from the Authentication category of filters. Drag and drop this filter on to the policy canvas to configure the filter. The sections below describe how to configure the fields on this filter screen.

Kerberos Client

The fields configured on this tab determine how the Kerberos Client obtains a service ticket for a specific Kerberos Service. The following fields must be configured:

Kerberos Client:
The role of the Kerberos Client selected in this field is twofold. First, it must obtain a Kerberos Ticket Granting Ticket (TGT) and second, it uses this TGT to obtain a service ticket for the **Kerberos Service Principal** selected below. The TGT is acquired at server startup, refresh (for example, when a configuration update is deployed), and when the TGT expires.

Click the button on the right, and select a previously configured Kerberos Client in the tree. To add a Kerberos Client, right-click the **Kerberos Clients** tree node, and select **Add Kerberos Client**. Alternatively, you can add Kerberos Clients under the **External Connections** node in the Policy Studio tree view. For more details, see the *Kerberos Clients* topic.

Kerberos Service Principal:
The Kerberos Client selected above must obtain a service ticket from the Kerberos Ticket Granting Server (TGS) for the Kerberos Service Principal selected in this field. The Service Principal can be used to uniquely identify the Service in the Kerberos realm. The TGS grants a ticket for the selected Principal, which the client can then send to the Kerberos Service. The Principal in the ticket must match the Kerberos Service's Principal for the client to be successfully authenticated.

Click the button on the right, and select a previously configured Kerberos Principal in the tree (for example, the default `HTTP/host Service Principal`). To add a Kerberos Principal, right-click the **Kerberos Principals** tree node, and select **Add Kerberos Principal**. Alternatively, you can add Kerberos Principals under the **External Connections** node in the Policy Studio tree view. For more details, see the topic on *Kerberos Principals*.

Kerberos Standard:
When using the **Kerberos Client Authentication** filter to insert Kerberos tokens into SOAP messages in order to authenticate to Kerberos Services, it can do so according to 2 different standards:

- Web Services Security Kerberos Token Profile 1.1
- WS-Trust for Simple and Protected Negotiation Protocol (SPNEGO)

When using the Kerberos Token Profile, the client-side Kerberos token is inserted into a `BinarySecurityToken` block within the SOAP message. The Kerberos session key may be used to sign and encrypt the SOAP message using the signing and encrypting filters. When this option is selected, the fields on the **Kerberos Token Profile** tab must be configured.

When the WS-Trust for SPNEGO standard is used, a series of requests and responses occur between the Kerberos Client and the Kerberos Service in order to establish a secure context. Once the secure context has been established (using WS-Trust and WS-SecureConversation), a further series of requests and responses are used to produce a shared secret key that can be used to sign and encrypt "real" requests to the Kerberos Service.

If the **WS-Trust for SPNEGO** option is selected it will not be necessary to configure the fields on the **Kerberos Token Profile** tab. However, the **Kerberos Client Authentication** filter must be configured as part of a complicated policy that is set up to handle the multiple request and response messages that are involved in setting up the secure context between the Kerberos Client and Service.

Kerberos Token Profile

The fields on this tab need only be configured if the **Kerberos Token Profile** option has been selected on the **Kerberos Client** tab. This tab allows you to configure where to insert the `BinarySecurityToken` within the SOAP message.

Where to Place BinarySecurityToken:
It is possible to insert the `BinarySecurityToken` inside a named WS-Security Actor/Role within the SOAP message or else an XPath expression can be specified to indicate where the token should be inserted.

Select the **WS-Security Element** radio button if you wish to insert the token into a WS-Security element within the SOAP Header element. You can either select the default "Current actor/role only" option from the dropdown or enter a named actor/role in the field provided. The `BinarySecurityToken` will be inserted into a WS-Security block for the actor/role specified here.

Alternatively, you should select the **XPath Location** option if you want to use an XPath expression to specify where the `BinarySecurityToken` is to be inserted. Click the **Add** button to add a new XPath expression or select an XPath and click the **Edit** or **Delete** buttons to edit or delete an existing XPath expression. Take a look at the *Configuring XPath Expressions* help page for more information on how to configure XPath expressions.

Note

You can insert the `BinarySecurityToken` *before* or *after* the node pointed to by the XPath expression. Select the **Append** or **Before** radio buttons depending on where you want to insert the token relative to the node pointed to by the XPath expression.

BinarySecurityToken Value Type:
Currently, the only supported `BinarySecurityToken` type is the "GSS_Kerberosv5_AP_REQ" type. The selected type will be specified in the generated `BinarySecurityToken`.

Kerberos Service Authentication

Overview

The API Gateway can act as a Kerberos Service to consume Kerberos tokens sent from a client in either the HTTP header or in the message itself. The client must have obtained a ticket from the Ticket Granting Server (TGS) for this Service.

The **Kerberos Service Authentication** filter is available from the Authentication category of filters. Drag and drop this filter on to the Policy Editor to configure this filter. The sections below describe how to configure the fields on this filter screen.

Kerberos Service

The **Kerberos Service** selected in this field is responsible for consuming the client's Kerberos token. The client must have obtained a ticket for the service's Principal name to be able to use the service.

Click the button on the right, and select a previously configured Kerberos Service in the tree. To add a Kerberos Service, right-click the **Kerberos Services** tree node, and select **Add Kerberos Service**. Alternatively, you can add Kerberos Services under the **External Connections** node in the Policy Studio tree view. For more details, see the *Kerberos Services* topic.

Kerberos Standard

Complete the following fields on this tab.

Kerberos Standard:
You must first select one of the following Kerberos standards:

- Kerberos Token Profile
- WS-Trust for SPNEGO
- SPNEGO over HTTP

Note

The Kerberos Service Authentication filter is used to consume the Kerberos client-side token regardless of whether the token is sent at the message layer (in the SOAP message), or at the transport layer (in an HTTP header).

Client Token Location for Message-Level Standards:
The Kerberos Service ticket can be sent in the `Authorization` HTTP header or inside the message itself (for example, inside a <BinarySecurityToken> element). Alternatively, it may be contained within a message attribute. Select one of the following options:

- **Message Body:**
  Select this option if you expect the Kerberos Service ticket to be contained within the XML message. You must enter an XPath expression to point to the expected location of the Kerberos token.

  Some default expressions that point to common locations are available for selection from the dropdown. Otherwise you can add a new XPath expression by clicking the **Add** button. Similarly, existing XPath expressions can be configured by clicking the **Edit** and **Delete** buttons, respectively. Please refer to the *Configuring XPath Expressions* for more information on configuring XPath expressions.
- **Message Attribute:**

When using the **WS-Trust for SPNEGO** standard above, the **Consume WS-Trust** filter will place the client-side Kerberos token inside the `ws.trust.spnego.token` message attribute.

Message Level

This section allows you to configure settings that adhere to the message-level standards, i.e. Kerberos Token Profile and WS-Trust for SPNEGO.

Extract Session Keys:
You must check this checkbox if you want to use the Kerberos/SPNEGO session keys to perform a signing or encryption/decryption operation in a subsequent filter. This option is only available when the token is extracted from the message body.

WS-Trust Settings: Key Length:
When using **WS-Trust for SPNEGO**, the **Kerberos Service Authentication** filter will generate a new symmetric key and wrap it using the Kerberos session key. This setting determines the length of the new symmetric key.

WS-Trust Settings: Cache Security Context Session Key:
The service-side may need to cache the session key in order to process (i.e. decrypt and verify) multiple requests from the client.

Transport Level

The options available in this section are specific to Kerberos tokens received over HTTP and are only relevant when the **SPNEGO Over HTTP** option is selected above.

Cookie Name:
The initial handshake between a Kerberos Client and Service can sometimes involve the exchange of a series of request and responses until the secure context has been established. In such cases, an HTTP cookie can be used to keep track of the context across multiple request and response messages. Enter the name of this cookie in the field provided.

Allow Client Challenge:
In some cases, the client may not authenticate (send the `Authorization` HTTP header) to the Kerberos Service on its first request. The Kerberos Service should then respond with an HTTP 401 response code, instructing the client to authenticate to the server by sending up the `Authorization` header. The client then sends up a second request, this time with the `Authorization` header, which contains the relevant Kerberos token. Check this option if you want to allow this type of negotiation between the client and service.

Client Sends Body Only After Context is Established:
The Kerberos client may wait to mutually authenticate the Kerberos service before sending the body of the message. If this setting is enabled, the Kerberos service will accept the body after the context has been established if the client provides the known cookie. The cookies are cached in the configured cache.

Advanced SPNEGO

Complete the following fields on this tab:

Cache Partially Established Contexts:
In theory, the Kerberos client and service may need to send and receive a number of tokens between each other in order to authenticate to each other. In this case, the **Kerberos Service Authentication** filter will need to cache the partially established context for each client. The contexts will only be cached during the establishment of the context.

In practice however, a single client-side Kerberos token is normally enough to establish a context on the service-side, in which case this setting is not required. This setting applies to the **WS-Trust for SPENGO** and **SPENGO over HTTP** standards only.

Kerberos Configuration

Overview

The **Kerberos Configuration** screen enables you to configure Process-wide Kerberos settings. The most important setting allows you to upload a Kerberos configuration file to the API Gateway, which contains information about the location of the Kerberos Key Distribution Center (KDC), encryption algorithms and keys, and domain realms to use.

You can also configure trace options for the various APIs used by the Kerberos system. For example, these include the Generic Security Services (GSS) and Simple and Protected GSSAPI Negotiation (SPNEGO) APIs.

Linux and Solaris platforms ship with a native implementation of the GSS library, which can be leveraged by the API Gateway. The location of the GSS library can be specified using settings on this screen.

Kerberos Configuration File - krb5.conf

The Kerberos configuration file (`krb5.conf`) is required by the Kerberos system to configure the location of the Kerberos KDC, supported encryption algorithms, and default realms.

The file is required by both Kerberos Clients and Services that are configured for the API Gateway. Kerberos Clients need to know the location of the KDC so that they can obtain a Ticket Granting Ticket (TGT). They also need to know what encryption algorithms to use and to what realm they belong.

A Kerberos Client or Service knows what realm it belongs to because either the realm is appended to the principal name after the `@` symbol. Alternatively, if the realm is not specified in the principal name, it is assumed to be in the `default_realm` as specified in the `krb5.conf` file.

Kerberos Services do not need to talk to the KDC to request a TGT. However, they still require the information about supported encryption algorithms and default realms contained in the `krb5.conf` file. There is only one `default_realm` specified in this file, but you can specify a number of additional named realms. The `default_realm` setting is found in the `[libdefaults]` section of the `krb5.conf` file. It points to a realm in the `[realms]` section. This setting is not required.

A default `krb5.conf` is displayed in the text area, which can be modified where appropriate and then uploaded to the API Gateway's configuration by clicking the **OK** button. Alternatively, if you already have a `krb5.conf` file that you want to use, browse to this file using the **Load File** button. The contents of the file are displayed in the text area, and can subsequently be uploaded by clicking the **OK** button.

Note

You can also type directly into the text area to modify the `krb5.conf` contents. Please refer to your Kerberos documentation for more information on the settings that can be configured in the `krb5.conf` file.

Advanced Settings

The checkboxes on this screen enable you to configure various tracing options for the underlying Kerberos API. Trace output is always written to the `/trace` directory of your API Gateway installation.

Kerberos Debug Trace:
Enables extra tracing from the Kerberos API layer.

SPNEGO Debug Trace:
Turns on extra tracing from the SPNEGO API layer.

Extra Debug at Login:

Provides extra tracing information during login to the Kerberos KDC.

Native GSS Library

The Generic Security Services API (GSS-API) is an API for accessing security services, including Kerberos. Implementations of the GSS-API ship with the Linux and Solaris platforms and can be leveraged by the API Gateway when it is installed on these platforms. The fields on this tab allow you to configure various aspects of the GSS-API implementation for your target platform.

Note

These are process-wide settings. If use of the native GSS API is selected, it will be used for all Kerberos operations. All Kerberos Clients and Services must therefore be configured to load their credentials natively.

If the native API is used the following will not be supported:

- The SPNEGO mechanism.
- The WS-Trust for SPNEGO standard as it requires the SPNEGO mechanism.
- The SPNEGO over HTTP standard as it requires the SPNEGO mechanism. (It is possible to use the KERBEROS mechanism with this protocol, but this would be non-standard.)
- Signing and encrypting using the Kerberos session keys.

Use Native GSS Library:
Check this checkbox to use the operating system's native GSS implementation. This option only applies to API Gateway installations on the Linux and Solaris platforms.

Native GSS Library Location:
If you have opted to use the native GSS library, enter the location of the GSS library in the field provided, for example, `/usr/lib/libgssapi.so`. On Linux, the library is called `libgssapi.so`. On Solaris, this library is called `libgss.so`.

Note

This setting is only required when this library is in a non-default location.

Native GSS Trace:
Use this option to enable debug tracing for the native GSS library.

Kerberos Clients

Overview

The API Gateway can act as a Kerberos client. In doing so, it must authenticate to the Kerberos KDC (Key Distribution Center) as a specific Principal and use the TGT (Ticket Granting Ticket) granted to it to obtain tickets from the TGS (Ticket Granting Service) so that it can authenticate to Kerberos services.

Kerberos Clients can be configured globally under the **External Connections** node in the tree view of the Policy Studio. To configure a Kerberos Client, right-click the **Kerberos Clients** node in the tree, and select the **Add a Kerberos Client** option from the context menu. Enter a name for the Kerberos Client in the **Name** field of the **Kerberos Client** dialog, and complete the following sections where necessary.

Having configured the Kerberos Client, it will be available for selection when configuring other Kerberos-related filters. Make sure to select the **Enabled** checkbox at the bottom of the screen, which is checked by default.

Ticket Granting Ticket Source

Use this section to configure where to obtain the Kerberos client credentials required in order to request service tickets, i.e. Ticket Granting Tickets (TGT) and the session key used in communications with the TGS. The TGT can be retrieved from a cache created as part of a JAAS (Java Authentication and Authorization Service) login, from delegated credentials, or from the native GSS implementation on Linux and Solaris platforms.

Note

Depending on what option is selected here, the **Kerberos Principal**, **Password**, and **Keytab File** fields below may or may not be disabled because some of the TGT source options do not require these fields to be configured.

Load via JAAS Login:
By default, the API Gateway will perform a JAAS login to the Kerberos KDC, after which the credentials will be cached by the API Gateway and used to acquire service tickets as they are needed. The JAAS login acquires the credentials in one of the following ways:

- *Request from KDC:*
  Request a new TGT from the Key Distribution Center. This is performed at server startup, refresh (for example, when a configuration update is deployed), and also when the TGT expires.
- *Extract from Default System Ticket Cache:*
  If a TGT has already been obtained out of bounds of the API Gateway and has been stored in the default system ticket cache, this option can be used to retrieve the TGT from this cache. On a Windows 2000 machine, the TGT will be extracted from the cache using the Local Security Authority (LSA) API. On a Linux/Solaris box, it is assumed that the ticket cache resides in `/tmp/krb5cc_uid`, where the uid is a numeric user identifier. If the ticket cache can not be found in these locations (or if we are running on a different Windows platform), the API Gateway will look for the cache in `{user.home}{file.separator}krb5cc_{user.name}`.

  ### Note

  A system ticket cache may only hold the credentials of a single Kerberos client. If you wish to load the credentials of more than one client from system ticket caches, they must be be explicitly named using the **Extract from System Cache** option. Ticket caches can be populated with client credentials using an external utility such as `kinit`.
- *Extract from System Ticket Cache:*
  Get the TGT from an explicitly named system ticket cache instead of from the default ticket cache. Browse to the

location of the alternative cache using the **Browse** button.

Load from Delegated Credentials:
The Kerberos Client can use a TGT that has been delegated for use by the server and has been already retrieved from a Kerberos Service Authentication filter. In this case, the TGT is extracted from message attributes (i.e. `authentication.delegated.credentials` and `authentication.delegated.credentials.client.name`) that have been set by a previous Kerberos Service Authentication filter. It is not necessary to configure the **Kerberos Principal** or **Secret Key** fields if this option is selected.

Load via Native GSS Library:
Select this option to have the Native GSS API acquire the client's credentials. The Native GSS API will expect the credentials to be already in a system ticket cache that it can access.

If **Load via Kinit** is not selected, the client credentials must exist in the default system ticket cache. In this case only one Kerberos client can be used within the API Gateway, as the API Gateway cannot support accessing credentials natively from the default system ticket cache and other system ticket caches. See above for more details on the location of the default system ticket cache.

If **Load via Kinit** is selected the API Gateway can support multiple Kerberos clients natively. In this case, the API Gateway will run `kinit` and create a ticket cache for each client in the `/conf/plugins/kerberos/cache` directory. The Native GSS API will know to acquire the client credentials from these caches.

Important

To use the GSS library and optionally the `kinit` tool in this manner, you must select to use the native GSS library on the Process-level **Kerberos Configuration** settings. To configure these settings, right-click the Process in the tree view of the Policy Studio, and select the **Kerberos** -> **Add** option from the context menu. Open the **Native GSS Library** tab of the **Kerberos Configuration** dialog and check the **Use Native GSS Library** checkbox.

Kerberos Principal

A Kerberos Principal is used to assign a unique identity to the API Gateway for use in the Kerberos environment. Select a previously configured Principal from the dropdown. You can configure Kerberos Principals globally under the **External Connections** node in the Policy Studio tree. For more information, see the *Kerberos Principals* topic.

Note

The semantics of this field are slightly different depending on what you selected as the TGT source above. If you have opted to retrieve the TGT from the KDC, then you are effectively asking the KDC to issue a TGT for the Principal selected here.

Alternatively, if you have opted to retrieve the TGT from a system ticket cache, then the Principal selected here will be used to lookup the cache in order to retrieve the TGT for this Principal. Similarly, if you want to use the `kinit` utility, the Principal name selected here will be passed as an argument to `kinit`.

Finally, if you wish to retrieve a TGT from delegated credentials, it is not necessary to specify any Principal.

Secret Key

The secret key is used by the principal to talk to the KDC's Authentication Service in order to acquire a TGT. The secret key can either be generated from a password or it can be taken from the principal's keytab file. Once again, the options available here will depend on what has been selected as the source of the TGT.

Password:

A password can only be entered if you have chosen to request the TGT from the KDC. The password will be used when generating the secret key. A secret key is not required at all if the TGT has been already retrieved either from a system ticket cache or from delegated credentials.

Important

The password entered here is stored by default in clear-text form in the API Gateway's underlying configuration data. If necessary, this can be encrypted using a Passphrase. For more information on encrypting all sensitive API Gateway configuration data, such as passwords, see *Setting the Encryption Passphrase*.

Keytab:
When the **Request from KDC** option is selected above, the secret key for the principal can also be extracted from a *Keytab* file, which maps Principal names to encryption keys. Similarly, the `kinit` tool requires a *Keytab* file.

You can load the Principal-to-key mappings into the table by selecting the **Load Keytab** button and then browsing to the location of an existing Keytab file. A new Keytab Entry can be added by clicking the **Add Principal** button See the *Kerberos Keytab* help page for more information on configuring the **Keytab Entry** dialog.

A Ketab Entry can be deleted by selecting the entry in the table and clicking on the **Delete Entry** button. You can also export the entire contents of the Keytab table by clicking the **Export Keytab** button.

Important

The contents of the Keytab table (whether derived from a Keytab file or manually entered using the **Keytab Entry** dialog) are stored in the clear in the API Gateway's underlying configuration data. The Keytab contents can be stored encrypted, if required, by setting a passphrase. For more details, see *Setting the Encryption Passphrase*.

When the server starts up it writes the stored Keytab contents out to the `/conf/plugin/kerberos/keytabs/` folder of your API Gateway installation. Oracle recommends that you configure directory- or file-based access control for this directory and its contents.

Advanced Tab

The following fields can be configured on this tab:

Mechanism:
Select the mechanism used to establish a context between the API Gateway and the Kerberos service. The Kerberos service must use the same mechanism.

Mutual Authentication:
Request that mutual authentication be carried out during context setup, i.e. the service authenticates back to the client. For the SPNEGO mechanism this must be turned on.

Integrity:
Enables data integrity for GSS operations.

Confidentiality:
Enables data confidentiality for GSS operations.

Credential Delegation:
Request that the initiator's credentials be delegated to the acceptor during context setup. When this option is checked, the acceptor can then assume the initiator's identity and authenticate to other Kerberos services on behalf of the initiator.

Anonymity:
Request that the client's identity is not disclosed to the service.

Replay Detection:
Enables replay detection for the per-message security services after context establishment.

Sequence Checking:
Turns on sequence checking for the per-message security services after context establishment.

Synchronize to Avoid Replays Errors at Service:
In cases where the Kerberos Client is running "under stress" and is attempting to send many requests to a Kerberos Service within a very short (millisecond) timeframe, it is possible that sequential Kerberos *Authenticator* tokens generated by the client will contain identical values for the *ctime* (i.e. the current time on the client's host) and *cusec* (i.e. the microsecond portion of the client's timestamp) fields.

Since Kerberos Service implementations often compare the ctime and cusec values on successive Authenticator tokens to determine replay attacks, it is possible that the Service will reject Authenticator requests in which the ctime and cusec fields have the same value.

To avoid situations where the Client may generate successive Authenticator requests (for a particular Service) in which the ctime and cusec fields are identical, you can select this option to synchronize the creation of the Authenticator requests. The Authenticator request generation will be synchronized using the **Pause Time** field below.

Pause Time:
Specify the time interval (in milliseconds) to wait before generating client-side Authenticator tokens when synchronizing to avoid over-zealous replay detection at the Kerberos Service. This field is only enabled if the **Synchronize to Avoid Replays Errors at Service** checkbox is checked above.

Note

The default value of 15 milliseconds matches the clock resolution time of operating systems such as Windows. Consult your operating system documentation for more information on the clock resolution for your target system.

Kerberos Services

Overview

The API Gateway can act as a Kerberos Service. In this case, Kerberos clients must obtain a Kerberos service ticket to authenticate to the Kerberos Service exposed by the API Gateway. Clients must present this ticket to the API Gateway for their requests to be processed (to be successfully authenticated). The Kerberos Service is responsible for consuming these tickets.

You can configure Kerberos Services globally under the **External Connections** node in the tree view of the Policy Studio. To configure a Kerberos Service, right-click the **Kerberos Services** node in the tree, and select the **Add a Kerberos Service** option from the context menu. The following sections describe how to configure the various fields on the **Kerberos Service** dialog.

Globally configured Kerberos Services are selected by name as part of the Kerberos Service Authentication filter, which is responsible for validating the tickets consumed by the Kerberos Service. Make sure to enter a descriptive name for the service in the **Name** field of the **Kerberos Service** dialog. For more information on configuring this filter, see the *Kerberos Service Authentication* topic.

Having configured the Kerberos Service, it is available for selection when configuring other Kerberos-related filters. Make sure to select the **Enabled** checkbox at the bottom of the screen, which is selected by default.

Kerberos Endpoint Tab

Complete the following fields on this tab:

Kerberos Principal:
Select the name of the principal to be associated with the API Gateway. Clients wishing to authenticate to the API Gateway ***must*** present a service ticket containing a matching principal name to the API Gateway.

Kerberos Principals are configured globally under the **External Connections** node in the tree view of the Policy Studio. Right-click the **Kerberos Principals** node, and select the **Add a Kerberos Principal** option from the context menu.

Alternatively, you can select the **Add** button under the **Kerberos Principal** drop-down list to add a new principal. For more information on configuring a principal, see the *Kerberos Principals* topic.

Secret Key:
Use this section to specify the location of the Kerberos Service's secret key, which is used to decrypt service tickets received from Kerberos clients.

Password:
The Kerberos Service's secret key is originally created for a specific Principal on the KDC. A password is required to generate this key, which can be entered directly into the **Password** field here.

Keytab:
Usually, however, a *Keytab* file is generated, which contains a mapping between a Principal name and that Principal's secret key. The Keytab file can then be loaded into the API Gateway configuration using the fields provided on this section.

You can load the Principal-to-key mappings into the table by selecting the **Load Keytab** button, and then browsing to the location of an existing Keytab file. You can add a new Keytab Entry by clicking the **Add Principal** button. For more information on configuring the **Keytab Entry** dialog, see the *Kerberos Keytab* topic.

You can delete a Keytab Entry by selecting the entry in the table, and clicking the **Delete Entry** button. You can also export the entire contents of the Keytab table by clicking the **Export Keytab** button.

Important

The contents of the Keytab table (whether derived from a Keytab file or manually entered using the **Keytab Entry** dialog) are stored in the clear in the API Gateway's underlying configuration. The Keytab contents can be stored encrypted, if required, by setting a passphrase for the API Gateway configuration data. For more information on how to do this, see the *Setting the Encryption Passphrase* topic.

When the server starts up it writes the stored Keytab contents out to the `/conf/plugin/kerberos/keytabs/` folder of your API Gateway installation. Oracle recommends that you configure directory-based or file-based access control for this directory and its contents.

Load via Native GSS Library:
If you have configured the API Gateway to **Use Native GSS Library** on the Process-level **Kerberos Configuration** settings, you must choose to load the Kerberos Service's secret key from the location preferred by the GSS library. The native GSS library expects the Kerberos service's secret key to be in the system's default Keytab file. The location of this Keytab file is specified in the `default_keytab_name` setting in the `krb5.conf` file that the native GSS library reads using the `KRB5_CONFIG` environment variable. This Keytab may contain keys for multiple Kerberos services.

Advanced Tab

Configure the following fields on this tab:

Mechanism:
Select the mechanism used to establish a context between this Kerberos service and the Kerberos client. The Kerberos client must use the same mechanism selected here.

Extract Delegated Credentials:
A Kerberos client can set an attribute on the context with the Kerberos service to indicate that they wish to allow the service to act on behalf of the client in subsequent communications. For example, this enables the Kerberos service (the API Gateway) to assume the identity of the client when communicating with a back-end Kerberos service. In this way, the client's credentials are propagated to the back-end service as opposed to the API Gateway's credentials. This is called *credential delegation*.

In cases where a Kerberos client wishes to delegate its credentials to a Kerberos service, you can configure the service to extract the delegated credentials from the context it establishes with the client. Select the **Extract Delegated Credentials** checkbox to extract the client's delegated credentials and store them in the `gss.delegated.credentials` and `gss.delegated.credentials.client.name` message attributes.

The extracted delegated credentials can be forwarded on to the back-end Kerberos service (on behalf of the user) using the Kerberos settings on the **Kerberos Client Authentication** filter or the **Connection** filter. When configuring the **Kerberos Client** used on the **Kerberos Authentication** tab of the **Connection** filter, make sure to select the option to retrieve the Ticket Granting Ticket(TGT) from the extracted delegated credentials (the **Extract from delegated credentials** checkbox on the **Kerberos Endpoint** tab).

For more details on configuring these options, see the following topics:

- *Connection*
- *Kerberos Clients*

Kerberos Principals

Overview

A Kerberos Principal represents a unique identity in a Kerberos system to which Kerberos can assign tickets to access Kerberos-aware services. Principal names are made up of several components separated by the `/` separator. You can also specify a realm as the last component of the name by using the `@` character. If no realm is given, the Principal is assumed to belong to the default realm, as configured in the `krb5.conf` file.

Typically, a Principal name comprises three parts: the *primary*, the *instance*, and the *realm*. The format of a typical Kerberos v5 Principal name is:
`primary/instance@realm`

- *Primary:*
  If the Principal represents a user in the system, the primary is the username of the user. Alternatively, for a host, the primary is specified as the `host` string.
- *Instance:*
  The instance can be used to further qualify the primary (for example, `user/admin@foo.abc.com`).
- *Realm:*
  This is your Kerberos realm, which is usually a domain name in upper case letters. For example, the `foo.abc.com` machine is in the `ABC.COM` Kerberos realm.

Configuration

You can configure Kerberos Principals globally under the **External Connections** node in the Policy Studio tree. To configure a Kerberos Principal, right-click the **Kerberos Principals** node, and select the **Add a Kerberos Principal** option from the context menu. Complete the following fields on the **Kerberos Principal** dialog:

Name:
Enter a friendly name for the Kerberos Principal. This name will be available for selection from drop-down lists in other Kerberos-related configuration screens in the Policy Studio.

Principal Name:
Enter the name of the Kerberos Principal in this field. The Principal name consists of a number of components separated using the `/` separator. The realm should be specified here if the Principal belongs to either a non-default realm or if a default realm is not specified.

Principal Type:
Select the type of Principal specified in the field above. The following table lists the available Principal Types.

Note

The Principal Name Types and their corresponding OIDs are defined in the General Security Services (GSS) API.

Principal Name Type	***Explanation***
`NT_USER_NAME`	The Principal name identifies a named user on the local system
`KERBEROS_V5_PRINCIPAL_NAME`	The Principal name represents a Kerberos version 5 Principal.
`NT_EXPORT_NAME`	The Principal name represents an exported canonical byte

Principal Name Type	***Explanation***
	representation of the name (for example, which can be used when searching for the Principal in an Access Control List (ACL)).
`NT_HOSTBASED_SERVICE`	The Principal name identifies a service associated with a specific host.

You can add new Principal Types by clicking the **Add** button. The name entered in the **Name** field on the **Kerberos Principal Name OID** must correspond to one of the constant fields defined in the `org.ietf.jgss.GSSName` Java class. Please refer to the Javadocs for the GSSName [http://java.sun.com/javase/6/docs/api/index.html] class for other allowable name types. Similarly, the corresponding OID for this name type must be entered in the **OID** field of the dialog. Please consult the GSSName Javadoc here [http://java.sun.com/javase/6/docs/api/index.html] for more information.

Important

OIDs and Principal Type Names should only be changed to reflect changes in the underlying GSS API. Because of this, you should only choose to **Edit** existing **Principal Types** under strict supervision from theOracle support team.

Kerberos Keytab

Overview

The *Kerberos Keytab* file contains mappings between Kerberos Principal names and DES-encrypted keys that are derived from the password used to log into the Kerberos Key Distribution Center (KDC). The purpose of the Keytab file is to allow the user to access distinct Kerberos Services without being prompted for a password at each Service. Furthermore, it allows scripts and daemons to login to Kerberos Services without the need to store clear-text passwords or for human intervention.

Important

Anyone with read access to the Keytab file has full control of all keys contained in the file. For this reason, it is imperative that the Keytab file is protected using very strict file-based access control.

The **Keytab Entry** dialog, which is available from the **Secret Key** section on both the Kerberos Client and Kerberos Service screens after clicking the **Add Principal** button, is essentially a graphical interface to entries in a Kerberos Keytab file.

This dialog enables you to generate keytab entries. You can remove entries from the Keytab file by clicking the **Delete Entry** button on the Kerberos Client and Kerberos Service screens. You can configure Kerberos Clients and Kerberos Services under the **External Connections** node in the Policy Studio tree.

Each key entry in the file is identified by a Kerberos Principal and an encryption type. For this reason, the Keytab file may hold multiple keys for the same principal where each key has a different encryption type. It may also contain keys for several different Principals.

In cases where the Keytab file contains encryption keys for different Principals, at runtime the Kerberos Client or Service only considers keys mapped to the Principal name selected in the **Kerberos Principal** drop-down list on their respective screens.

If the Keytab file contains several keys for the Principal, the Kerberos Client or Service uses the key with the strongest encryption type as agreed during the negotiation of previous messages with the Kerberos Key Distribution Center (KDC).

Configuration

Configure the following fields on the **Keytab Entry** dialog:

Kerberos Principal:
Select an existing Kerberos Principal from the drop-down list or add a new one by clicking on the **Add** buttons. You can configure Kerberos Principals globally under the **External Connections** node in the Policy Studio tree. For more information on configuring Kerberos Principals, see the *Kerberos Principals* topic.

Password:
The password entered here is used to seed the encryption algorithm(s) selected below.

Encryption Types:
The encryption types selected here determine the algorithms used to generate the encryption keys that are stored in the Keytab file. In cases where the Keytab file contains multiple keys for the Principal, the encryption type is used to select an appropriate encryption key.

To ensure maximum interoperability between Kerberos Clients/Services configured in the API Gateway and different types of KDC, all encryption types are selected by default. With this configuration, the generated Keytab file contains a separate encryption key for each encryption type listed here where each key is mapped to the Principal name selected above.

Important

You must Ensure that the required encryption types exist in the Keytab as defined by settings in the `krb5.conf`. For a Kerberos Client to request a Ticket Granting Ticket, it must have at least one key that matches one of the encryption types listed in the `default_tkt_enctypes` setting in the `krb5.conf`file. A Kerberos Service requires a key of a certain encryption type to be able to decrypt the service ticket presented by a client.

For Windows 2003 Active Directory, by default, the service ticket is encrypted using the `rc4-hmac` encryption type. However, if the service user has the **Use DES encryption types for this account** option enabled, the `des-cbc-md5` encryption type is used.

SAML Authentication XML-Signature Verification

Overview

A SAML (Security Assertions Markup Language) *authentication assertion* is issued as proof of an authentication event. Typically an end-user will authenticate to an intermediary, who generates a SAML authentication assertion to prove that it has authenticated the user. The intermediary will usually *sign* the assertion as proof that only it could have signed the assertion, and also to guarantee the integrity of the assertion. It then inserts the assertion, together with its signature, into the message for consumption by a downstream Web Service.

The following sample SOAP message contains a signed SAML authentication assertion:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
 <soap-env:Header xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <wsse:Security>
   <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
                    AssertionID="oracle-1056477425082"
                    Id="oracle-1056477425082"
                    IssueInstant="2003-06-24T17:57:05Z"
                    Issuer="CN=Sample User,....,C=IE"
                    MajorVersion="1"
                    MinorVersion="0">
      <saml:Conditions
                      NotBefore="2003-06-20T16:20:10Z"
                      NotOnOrAfter="2003-06-20T18:20:10Z"/>
      <saml:AuthenticationStatement
                      AuthenticationInstant="2003-06-24T17:57:05Z"
                      AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
        <saml:SubjectLocality IPAddress="192.168.0.32"/>
        <saml:Subject>
          <saml:NameIdentifier
                  Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
                  sample
          </saml:NameIdentifier>
        </saml:Subject>
      </saml:AuthenticationStatement>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
      <dsig:SignedInfo>
        .....
      </dsig:SignedInfo>
      <dsig:SignatureValue>
        rpa/......0g==
      </dsig:SignatureValue>
      <dsig:KeyInfo>
        .....
      </dsig:KeyInfo>
    </dsig:Signature>
   </saml:Assertion>
  </wsse:Security>
 </soap-env:Header>
<soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
  </soap-env:Body>
</soap-env:Envelope>
```

Configuration

Configure the following fields to validate the XML Signature over a SAML assertion:

SAML Signature:
Use this section to specify the location of the signature to validate. The signature can be selected using 3 options:

- ***Check signature inside the assertion:***
  Select this option if the signature will be present inside the SAML assertion itself.
- ***Check signature contained in WS-Security Block:***
  If the signature is contained within a WS-Security block (but outside the assertion), it is necessary to specify whether the signature covers only the assertion, or the assertion and the SOAP Body. Select the appropriate option depending on what the signature covers.
- ***Use advanced XPath:***
  If the signature is to be found in a non-standard location, an XPath expression can be used to identify it. Use the **Signature location XPath** to find a signature in a non-standard place.
  It is also necessary to specify the nodes that are signed by the signature. Use the **What must be signed XPath** to configure this.

Signer's Public Key/Certificate
Select the **Certificate in Message** radio button in order to use the certificate from the XML-Signature specified in the **SAML Signature** section. The certificate will be extracted from the `KeyInfo` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ....... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key must be retrieved from an LDAP directory of the API Gateway's Certificate Store.

For example, the following signed XML message does not include the signatory's certificate. Instead only the *Common Name* of the signatory's certificate is included. In this case, the API Gateway must obtain the certificate from either an LDAP directory or the Certificate Store in order to validate the signature on the assertion.

```
<?xml version="1.0" encoding="UTF-8"?>
 <soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
    <dsig:SignedInfo>
     <dsig:CanonicalizationMethod
                   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     <dsig:SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference URI="">
      <dsig:Transforms>
```

```
       <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <dsig:XPath>ancestor-or-self::soap-env:Body</dsig:XPath>
       </dsig:Transform>
       <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
      </dsig:Transforms>
      <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <dsig:DigestValue>rvJMkZ1RDo3pNfqCUBa4Qhs8i+M=</dsig:DigestValue>
     </dsig:Reference>
    </dsig:SignedInfo>
    <dsig:SignatureValue>
      AXL2gKhqqKwcKujVPftVoztySvtCdARGf97Cjt6Bbpf0w8QFiNuLJncQVnKB
      cQ+91KvudYZ/Sk8u7tXhoEiLvNwg76B2STPh+ypEWO+J7OSPedlUdnfVRRvW
      vjYLwJVjGNZ+mMTxvfO1wwcIb2Hg94n1BOaeBrNJ+2uO4i87W5TyufAGI+V8
      S6oSpPc5KQeHLXoyHS2+fXyqReSiwdhOeli4D4xT+HbjRgYJIwIikXn2k1Fr
      D/hnd1/xVf/LjrOwoY9id8W3IcZAzMIRh5SBZjWHYOQzk79xy4YDpzNVYIOB
      laAFqzg9G+Z4VYj+RdgrIVHhOXt+mq+fGZV6VheWGQ==
    </dsig:SignatureValue>
    <dsig:KeyInfo>
     <dsig:KeyName>
       CN=User,OU=R&D,O=Org Ltd.,L=Dublin 4,ST=Dublin,C=IE
     </dsig:KeyName>
    </dsig:KeyInfo>
   </dsig:Signature>
  </soap-env:Header>
 <soap-env:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </soap-env:Body>
</soap-env:Envelope>
```

To retrieve a client certificate from an LDAP directory, select a pre-configured one from the **LDAP Source** dropdown, or add/edit a new/existing LDAP directory by clicking the **Add/Edit** button.

Alternatively, select a certificate from the Trusted Certificate Store by selecting the **Certificate in Store** radio button and clicking on the **Select** button. This certificate will then be associated with the incoming message so that all subsequent certificate-based filters will use this user's certificate.

XML Signature Authentication

Overview

The API Gateway can authenticate a client by validating the XML Signature contained within an incoming XML message. A successful signature validation proves that the client had access to the signing key. Since the signing key is only accessible by the client (i.e. is not publicly available), the validation process authenticates the client.

Configuration

The following sections can be configured on the **XML Signature Authentication** screen:

Signature Location
There may be multiple signatures present in a given XML message. For this reason, it is necessary to tell the API Gateway which signature it should use to authenticate the client.

The signature can be extracted:

- Using WS-Security Actors
- From the SOAP Header
- Using XPath

Select the most appropriate method from the **Signature Location** dropdown. Your selection will depend on the types of SOAP messages that you expect to receive. For example, if incoming SOAP messages will contain an XML Signature within a WS-Security block, you should choose this option from the dropdown.

What Must be Signed
This section defines the content that must be signed in order for a SOAP message to pass the filter. This ensures that the client has signed something meaningful (i.e. part of the SOAP message) as opposed to some arbitrary data that would pass a "blind" signature validation. This further strengthens the authentication process.

An XPath expression is used to identify the nodeset that should be signed. To specify that nodeset, select either an existing XPath expression from the **XPath Expression** dropdown list, or add a new one using the **Add** button. XPath expressions can also be edited and removed with the **Edit** and **Remove** buttons respectively.

Signer's Public Key/Certificate
Select the **Certificate in Message** radio button in order to use the certificate from the XML-Signature specified in the **Signature Location** section. The certificate will be extracted from the `KeyInfo` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ....... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key can be retrieved from a specified LDAP directory or from the **Certificate Store**.

To retrieve a client certificate from an LDAP directory, select a pre-configured one from the **LDAP Source** dropdown, or add/edit a new/existing LDAP directory by clicking the **Add/Edit** button.

Alternatively, select the name of a **User** from the **Certificate** field. This user's certificate will then be associated with the incoming message so that all subsequent certificate-based filters will use this user's certificate.

SSL Authentication

Overview

A client can mutually authenticate to the API Gateway through the exchange of X.509 certificates. An X.509 certificate contains identity information about its owner and is digitally signed by the Certificate Authority that issued it.

A client will present such a certificate to the API Gateway while the initial SSL/TLS session is being negotiated, in other words, during the *SSL handshake*. The **SSL Authentication** filter extracts this information from the client certificate and sets it as message attributes. These attributes can then be used by subsequent filters in the policy.

The **SSL Authentication** filter can be used as a decision-making node on the policy. For example, it can be used to determine a path through a policy based on how users authenticate to the API Gateway.

Configuration

Name:
Enter a name for the filter on the **SSL Authentication** configuration screen.

Security Token Service Client

Overview

The **Security Token Service Client** filter enables the API Gateway to act as a client to a Security Token Service (STS). An STS is a third-party Web Service that authenticates clients by validating credentials and issuing security tokens across different formats (for example, SAML, Kerberos, or X.509). The API Gateway can use the **Security Token Service Client** filter to request security tokens from an STS using WS-Trust. WS-Trust specifies the protocol for issuing, exchanging, and validating security tokens.

An STS has its own security requirements for authenticating and authorizing requests for tokens. This means that the API Gateway may need to insert tokens, digitally sign, and encrypt the request that it sends to the STS for the required token. Because the STS is exposed as a Web Service, it should have a WSDL file with WS-Policies that describe its security requirements.

For example, the API Gateway can use the **Security Token Service Client** filter to request tokens that it cannot issue itself, and which may be required by an endpoint service. The endpoint service may require tokens to be signed by a particular authority (STS), or there may be a requirement for a token that contains a key encrypted for the endpoint service, and which only the STS can generate. You can also use the **Security Token Service Client** filter to virtualize an STS using the API Gateway.

Example Request

Using WS-Trust, requests for tokens are placed in a `RequestSecurityToken` (RST) element in the SOAP `Body` element. The STS returns the requested token in a `RequestSecurityTokenResponse` (RSTR) element in the SOAP `Body`. The following example is an extract from a token request message sent from the API Gateway to the STS:

```
<soap:Body
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-utility-1.0.xsd"
     wsu:Id="Id-0000012e71431904-00000000011d5641-19">
     <wst:RequestSecurityToken
         xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
         Context="Id-0000012e71431904-00000000011d5641-15">
         <wst:RequestType>
             http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
         </wst:RequestType>
         <wst:TokenType>
             http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
         </wst:TokenType>
         <wst:KeyType>
             http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey
         </wst:KeyType>
         <wst:Entropy>
             <wst:BinarySecret
                 Type="http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey">
                 WLQmo5mRYiBRqq2D7677Dg==
             </wst:BinarySecret>
         </wst:Entropy>
         <wsp:AppliesTo
             xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
             <wsa:EndpointReference
                 xmlns:wsa="http://www.w3.org/2005/08/addressing">
                 <wsa:Address>default</wsa:Address>
             </wsa:EndpointReference>
         </wsp:AppliesTo>
     </wst:RequestSecurityToken>
 </soap:Body>
```

In this simple example, the client (API Gateway) requests a SAML token with a symmetric `KeyType`. The SAML token is requested for an endpoint service named `default`. An optional `OnBehalfOf` token is not supplied.

The **Security Token Service Client** filter enables you to configure token requests sent by the API Gateway to the STS. This rest of this topic explains all of the available settings.

General Settings

You can configure the following general setting on the filter screen:

Name:
Enter an appropriate name for this filter.

Request Settings

Configure the following general request settings on the **Request** tab:

Request Type:
Select one of the following request types:

Issue	A request to issue a token. This is the default request type.
Validate	A request to validate a token.

Token Type to Request:
Select the token type to request from the STS (for example, `SAML 1.0`, `SAML 1.1`, `SAML 2.0`, or `UsernameToken`). You can also request a custom token type by entering the custom token URI (for example, `http://www.mycustomtoken.com/EmailToken`). The default is `SAML 1.1 (#SAMLV1.1)`.

Issue: POP Key

A *proof-of-possession* (POP) security token contains secret data used to demonstrate authorized use of an associated security token. Typically, the POP data is encrypted with a key known only to the recipient of the POP token. For **Issue** requests, you can configure the following POP key settings on the **Request** -> **Issue: POP Key** tab:

Proof of Possession Key Type:
Select the POP key type for the token you are requesting. This only applies to certain types of tokens (for example, SAML tokens). Select one of the following key types from the drop-down list:

`SymmetricKey`	When a SAML token is requested with a symmetric POP key, the SAML assertion returned by the STS has a subject confirmation type of `holder-of-key`. The subject confirmation data contains a symmetric key encrypted for the endpoint service. The API Gateway (the client) can request the SAML token from the STS with the endpoint service specified as the token scope, so the STS knows what certificate to use to encrypt the symmetric key it places in the SAML assertion's subject confirmation data. The API Gateway cannot decrypt the symmetric key in the SAML assertion because it is encrypted for the endpoint service. The STS passes the symmetric key to the requesting API Gateway in the RSTR so that the API Gateway also has the symmetric key. It can then use the SAML assertion (symmetric key) to sign the message to the endpoint service, proving that it holds the key in the SAML assertion.

	The endpoint service can verify the signature because it can decrypt the key in the SAML assertion. This is the default POP key type.
`PublicKey`	When a SAML token is requested with a public asymmetric POP key, the SAML assertion returned by the STS has a subject confirmation type of `holder-of-key`. The subject confirmation data contains a public key or certificate. The API Gateway (the client) can also use this SAML assertion to sign messages to the endpoint service using the related private key, thus proving they hold the key referenced in the SAML assertion. The public key in the SAML assertion is not encrypted because it is not sensitive data. This SAML assertion can be used to sign messages to multiple endpoint services because it does not contain a key encrypted for a specific service. The API Gateway can specify the public key used in the **Public Proof of Possession Key** settings. This public key can be associated with a certificate in the certificate store, or generated on-the-fly using the **Generate Key** filter. For more details, see the *Generate Key* topic.
`Bearer`	When a SAML token is requested with a bearer POP key, the SAML assertion returned by the STS has a subject confirmation type of `bearer`. In this case, the SAML token does not contain a POP key.

Important

An STS can also generate a SAML token with a subject confirmation type of `sender-vouches`. In this case, the endpoint service trusts the client directly, the SAML assertion does not need to be signed by the STS. The client signs the SAML assertion and the SOAP `Body` before sending the message to the endpoint service. This type of SAML assertion does not map to a value for **Proof of Possession Key Type**, but can be returned from the STS if no key type is specified.

Key Size:
Enter the key size in bits to indicate the desired strength of the security. Defaults to `256` bits.

Entropy Type:
If the **Proof of Possession Key Type** requested is a `SymmetricKey`, you must specify an **Entropy Type**. If the API Gateway provides *entropy*, this means that it provides some of the binary material used to generate the symmetric key. In general, both the API Gateway and the STS provide some entropy for the symmetric key (a computed key). However, either side can also fully generate the symmetric key. Select one of the following options:

None	The API Gateway does not provide any entropy, so the STS must fully generate the symmetric key.
Binary Secret	The API Gateway provides entropy in the form of a Base64-encoded binary secret (or key). You must specify a **Binary Secret Type**. For details, see the next setting.
EncryptedKey	The API Gateway provides entropy in the form of an `EncryptedKey` element. You must configure an **XML-Encryption** filter in the policy, which applies security before creating the WS-Trust message. This filter generates a

	symmetric key and encrypts it, but does not encrypt any data. The key must be encrypted with the STS certificate.

Binary Secret Type:
If the **Entropy Type** is **Binary Secret**, you must specify a **Binary Secret Type**. Select one of the following:

`Nonce`	The API Gateway generates a nonce value and places it in the RST.
`SymmetricKey`	The **Binary Secret Message Attribute** value must be specified. In this case, this is the name of the message attribute that contains the symmetric key passed to the STS to be used as entropy for generating the POP symmetric key. The type of this message attribute must be `byte[]` when the **Binary Secret Type** is `SymmetricKey`.
`AsymmetricKey`	The **Binary Secret Message Attribute** value must be specified. In this case, this is the name of the message attribute that contains the private asymmetric key passed to the STS to be used as entropy for generating the POP symmetric key. The type of this message attribute must be `byte[]`, `PrivateKey`, `KeyPair`, or `X509Certificate` when the **Binary Secret Type** is `AsymmetricKey`. In each case, the private key is used.

Binary Secret Message Attribute:
Enter or select the message attribute that contains the binary secret. This setting is required when the **Binary Secret Type** is `SymmetricKey` or `AsymmetricKey`.

Computed Key Algorithm:
When both the API Gateway and STS provide entropy values for the symmetric POP key, you can specify a computed key algorithm (for example, `PSHA1`). This is used when the key resulting from the token request is not directly returned, and is computed.

Public Proof of Possession Key:
If the **Proof of Possession Key Type** requested is a `PublicKey`, you can specify what public key to include in the token using the following settings:

Use Key Format	Select how the `UseKey` element in the RST formats the public key from the drop-down list (for example, `PublicKey`, `Certificate`, `BinarySecurityToken`, and so on).
Use Key Message Attribute	Select or enter the message attribute that contains the public key. The public key can be of type `X509Certificate`, `PublicKey` or `KeyPair`.

Issue: On Behalf Of Token

For **Issue** requests, you can optionally configure the OnBehalfOf token for the RST. If an OnBehalfOf token is in the RST, this means you are requesting a token on behalf of the subject identified by the token or endpoint reference in the OnBehalfOf element. You can configure the following settings on the **Request** -> **Issue: On Behalf Of Token** tab:

On Behalf Of:
Select one of the following options:

None	No OnBehalfOf token is specified. This is the default.
Token	The token is embedded directly under the <OnBehalfOf> element in the RST.
EmbeddedSTR	The token is placed in the <OnBehalfOf><SecurityTokenReference><Embedded> element in the RST.
Endpoint Reference	A reference to the token is placed in the <OnBehalfOf><SecurityTokenReference>< element. The token is placed in the WS-Security header.

On Behalf Of Token Message Attribute:
Enter or select the message attribute that contains the OnBehalfOf token. This may be a UsernameToken, SAML token, X.509 certificate, and so on. The type of this message attribute can be Node, List of Nodes, String, or X509Certificate. This message attribute must be populated using a filter configured in the policy that applies security before creating the WS-Trust message. For example, this includes a filter to extract a UsernameToken from the incoming message, or a **Find Certificate** filter.

Endpoint Address:
When the **On Behalf Of** type is **Endpoint Reference**, no token is placed in the OnBehalfOf element. Instead, you can enter an endpoint address in this field that identifies the subject on whose behalf you are requesting the token.

Identity Type:
When the **On Behalf Of** type is **Endpoint Reference**, you can select an identity type from the drop-down list (for example, DNSName, ServicePrincipaName, or UserPrincipalName).

Identity:
When the **Identity Type** is set to DNSName, ServicePrincipaName, or UserPrincipalName, you must specify a value in this field.

Identity Message Attribute:
When the selected **Identity Type** is one of PublicKey, Certificate, BinarySecurityToken, SecurityTokenReference_x509v3, or SecurityTokenReference_ThumbprintSHA1, you must specify a message attribute in this field. This specifies the name of the message attribute that contains the certificate for the subject on whose behalf you are requesting the token. The type of this message attribute must be X509Certificate.

Issue: Token Scope and Lifetime

For **Issue** requests, you can optionally specify details for the scope of the requested token (for example, the endpoint service this token is used for). These details are placed in the AppliesTo element of the RST. You can configure the following settings on the **Request** -> **Issue: Token Scope and Lifetime** tab:

Endpoint Address:
Enter an address for the endpoint.

Identity Type:
Select an identity type from the drop-down list (for example, Certificate, BinarySecurityTokenDNSName, Servi-

cePrincipalName, or UserPrincipalName).

Identity:
When the **Identity Type** is set to DNSName, ServicePrincipaName, or UserPrincipalName, you must specify a value in this field.

Identity Message Attribute:
When the **Identity Type** selected is one of PublicKey, Certificate, BinarySecurityToken, SecurityTokenReference_x509v3, or SecurityTokenReference_ThumbprintSHA1, you must specify a message attribute in this field. This specifies the name of the message attribute that contains the certificate for the endpoint service that the token is sent to. The type of this message attribute must be X509Certificate.

Expires In:
Specify when the token is due to expire in the day and time boxes.

Lifetime Format:
Enter the date and time format in which the token lifetime is specified. Defaults to yyyy-MM-dd'T'HH:mm:ss.SSS'Z'.

Note

The STS may choose to ignore the token lifetime specified in the RST.

Validate: Target

If the request type is set to **Validate**, you can use the **Request** -> **Validate: Target** tab to specify the token that you require the STS to validate. In this case, the STS does not issue a token. It validates the token passed to it in the RST and returns a status. The response from the STS is placed in the sts.validate.code and sts.validate.reason message attributes.

You can configure the following settings on the **Request** -> **Validate: Target** tab:

Token:
Specifies that the token is placed directly under the <ValidateTarget> element in the RST.

EmbeddedSTR:
Specifies that the token is placed in the <ValidateTarget><SecurityTokenReference><Embedded> element.

STR:
Specifies that a reference to the token is placed in the <ValidateTarget><SecurityTokenReference> element. The token is placed in the WS-Security header.

Validate Target Message Attribute:
Select the message attribute that contains the token that you wish to validate. The type of this message attribute can be Node, a List of Nodes, or String. This message attribute must be populated using a filter configured in the policy that applies security before creating the WS-Trust message. For example, you can run a filter to extract a SAML token from the incoming message.

Policies Settings

The **Policies** tab enables you to specify the policies that the **Security Token Service Client** filter delegates to. You can configure the following settings on this tab by clicking the button next to each field:

Policy to run to apply security before creating the WS-Trust message:
Specifies the policy that runs before the **Security Token Service Client** filter creates the RST (the WS-Trust request message for the STS). The filters in this policy are used to set up message attribute values that the STS client filter requires (for example, the OnBehalfOf token).

Policy to run to apply security to the WS-Trust request:

Specifies the policy that runs after the **Security Token Service Client** filter has created the RST. The filters in this policy can sign and/or encrypt the message as required by the STS. It can also inject other security tokens into the WS-Security header if required.

Policy to run to apply security to the WS-Trust response:
Specifies the policy that runs to apply security to the WS-Trust response. This policy runs when the response is received from the STS. The filters in this policy can decrypt and verify signatures on the response message.

Routing

When routing to an STS, you can specify a direct connection to the Web Service endpoint by entering a URL on the **Routing** tab. Alternatively, when the routing behavior is more complex, you can delegate to a custom routing policy to handle the added complexity. The options on the **Routing** tab allow for these alternative routing configurations.

Use the following URL:
Select this option to route to the specified URL. You can enter the URL in the text box, or specify the URL as a selector so that the URL is built dynamically at runtime from the specified message attributes (for example `${host}:${port}`, or `${http.destination.protocol}://${http.destination.host}:${http.destination.port}`). For more details on selectors, see *Selecting Configuration Values at Runtime*

In both cases, you can configure connection details such as SSL for the direct connection using the fields in the **Connection Details** group. For more details, see the *Connection* filter topic.

Delegate to Routing Policy:
Select this option if you wish to use a dedicated routing policy to send messages on to the STS. Click the browse button next to the **Routing policy** field to select the policy that you want to use to route messages.

No Routing:
Select this option to only allow request reflection for test purposes.

Response Settings

The **Response** tab enables you to specify options for processing the response message from the STS. You can configure the following settings on this tab:

Verify returned security token type:
When selected, the filter checks that the `TokenType` returned is what was requested. This is selected by default.

Put security token into message attribute:
When specified, the token returned from the STS is placed in the specified message attribute. The type of this attribute is `String`. Defaults to `sts.security.token`. An element version of the token is placed in a message attribute named *`attrname.element`*.

Insert security token into original message in SOAP Actor/Role:
When specified, the token returned from the STS is inserted into the original message. This is the original message received by the API Gateway (was the current message before the **Security Token Service Client** filter ran). Defaults to `Current actor/role only`.

Extract Token Lifetime:
When selected, the token lifetime is extracted from the response, and the `sts.token.lifetime.created` and `sts.token.lifetime.expires` message attributes are populated. This setting is selected by default.

Advanced Settings

The **Advanced** tab enables you to specify the following options:

Versions and Namespaces:
The version and namespace options are as follows:

WS-Trust Version	Specifies the WS-Trust namespace to use in the generated RST. Defaults to `WS-Trust 1.3`.
SOAP version	Specifies the SOAP version to use in the generated RST. Defaults to `SOAP 1.1`.
WS-Addressing Namespace	Specifies the WS-Addressing namespace to use in the generated RST. Defaults to `http://www.w3.org/2005/08/addressing`.
WS-Policy Namespace	Specifies the WS-Policy namespace to use in the generated RST. Defaults to `WS-Policy 1.2`.
WS-Security Actor	Specifies the actor in which to place tokens that are referred to from the RST using STRs (for example, `OnBehalfOf`). Defaults to `Current actor/role only`.

Algorithms:
The algorithm options are as follows:

Cannonicalization Algorithm	When selected, additional elements are added to the RST, which specify a client-requested cannonicalization algorithm (for example, `ExC14n`).
Encryption Algorithm	When selected, additional elements are added to the RST, which specify a client-requested encryption algorithm (for example, `Aes256`).
Encrypt with	When selected, specifies the encryption algorithm with which to encrypt the RSTR (for example, `Aes256`).
Sign with	When selected, specifies the signature algorithm with which to digitally sign the RSTR (for example, `RsaSha256`).

Advanced Settings:
The advanced options are as follows:

Content-Type	Specifies the `Content-Type` of the message to be sent to the STS. For example, for Microsoft Windows Communication Foundation (WCF), select `application/soap+xml`. Defaults to `text/xml`.
Store and restore original message	When selected, the original message is saved before messages sent from the API Gateway to the STS and messages sent from the STS to the API Gateway are processed. It is then reinstated after this filter finishes processing the STS response. This is the default behavior. For debug purposes, you may wish to return the STS response from your policy. In this case, deselect this setting, and the current message after this filter completes should then be the STS response. You may also wish to debug the RST (the request to the STS), and return that from your policy. In this case, disable this setting, click the **Routing** tab, and select the **No routing** option.

WS-Security Username Authentication

Overview

A WS-Security *Username Token* enables an end-user identity to be passed over multiple hops before reaching the destination Web Service. The user identity is inserted into the message and is available for processing at each hop on its path.

The client user name and password are encapsulated in a WS-Security `<wsse:UsernameToken>`. When the API Gateway receives this token, it can perform one of the following tasks, depending on the requirements:

- Ensure that the timestamp on the token is still valid
- Authenticate the user name against a repository
- Authenticate the user name and password against a repository

The following sample SOAP message contains two `<wsse:UsernameToken>` blocks:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="sample"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>sample</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:44:51Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
  <wsse:Security soap:actor="oracle"
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="oracle"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>oracle</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:46:04Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
 </soap:Header>
  <soap:Body>
   <getHello xmlns="http://www.oracle.com"/>
  </soap:Body>
</soap:Envelope>
```

This topic explains how to configure the API Gateway to authenticate users using a WS-Security `<wsse:UsernameToken>`.

General Configuration

To configure general settings, complete the following fields:

Name:
Enter an appropriate name for this filter.

Actor:
The example SOAP message at the top of this page contains two `<wsse:UsernameToken>` blocks. You must specify which block contains the `<wsse:UsernameToken>` used to authenticate the end-user. Specify the SOAP Actor/Role of

the WS-Security block that contains the token.

Credential Format:
The API Gateway can authenticate users against a user profile repository based on User Names, X.509 Distinguished Names, or email addresses. Unfortunately, the WS-Security specification does not provide a means of specifying the type of `<wsse:UsernameToken>`, and so it is necessary for the administrator to do so using the **Credential Format** field. The type specified here is used internally by the API Gateway in subsequent authorization filters.

Token Validation

Each `wsse:UsernameToken` contains a timestamp inserted into the `<wsu:Created>` element. Using this timestamp together with the details entered in this section, the API Gateway can determine whether the WS-Security `UsernameToken` has expired. The `<wsu:Created>` element is as follows:

```
<wsse:UsernameToken wsu:Id="oracle"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
  <wsu:Created>2006.01.13T-10:42:43Z</wsu:Created>
  ...
</wsse:UsernameToken>
```

To configure token validation settings, complete the following fields:

Drift Time:
Specified in seconds to account for differences in the clock times between the machine on which the token was generated and the machine running the API Gateway. Using the *start time*, *end time*, and *drift time*, the token is considered valid if the current time falls between the following times:

```
[start - drift] and [start + drift + end]
```

Validity Period:
Specifies the lifetime of the token, where the value of the `<wsu:Created>` element represents the *start time* of the assertion, and the time period entered represents the *end time*.

Timestamp Required:
Select this option if you want to ensure that the Username Token contains a timestamp. If no timestamp is found in the Username Token, a SOAP Fault is returned.

Nonce Required:
Select this option to ensure that the Username Token contains a `<wsse:Nonce>` element. This is a randomly generated number that is added to the message. You can use the combination of a timestamp and a nonce to help prevent replay attacks.

Select cache to store WSS username token nonces in:
Click the button on the right, and select the cache that stores the nonce value (for example, `Kerberos Session Keys`). Defaults to the local `WSS Username Token Nonce Cache`.

To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Token Verification via Repository

Having validated the timestamp on the token, the API Gateway can then optionally authenticate the user name and password contained in the token. The following options are available:

- **No Verification**
  No verification of the user name and password is performed. Only the timestamp on the token is validated. This is

the default behavior.

- **Verify Username Only**
  Only the user name is looked up in the selected repository. If the user name is found in this repository, the user is authenticated. Select the **No password allowed** checkbox to block messages that contain a Username Token with a `<wsse:Password>` element.
- **Verify Username and Password**
  The user name is looked up in the selected repository and is only authenticated if the corresponding password matches the one configured in the repository. If you select this option, you must select the type of the password. Both cleartext and digest formats are supported. Select the appropriate option.

Repository Name:
The API Gateway attempts to authenticate users against the selected **Authentication Repository**. User profiles can be stored in the local store, a database, or an LDAP directory. For details on adding a new repository, and editing or deleting a repository, see the *Authentication Repository* tutorial.

Remove enclosing WS-Security element on successful validation:
Select this option if you wish to remove the WS-Security block that contains the Username Token after the token has been successfully authenticated. For example, in the above sample SOAP message that contains two `<wsse:UsernameToken>` elements in two different WS-Security blocks, you could configure the API Gateway to remove one of these on successful authentication.

Attributes

Overview

The purpose of the filters in the **Attributes** filter group is to extract user attributes from various sources. It is possible to retrieve attributes from the message, an LDAP directory, a database, the **User Store**, HTTP headers, and finally, from a SAML attribute assertion.

Having retrieved a set of user attributes, the API Gateway then stores them in the `attribute.lookup.list` message attribute, which is essentially a map of name-value pairs. It is the role of the **Attributes** authorization filter to check the value of these attributes to authorize the user.

Configuration

The following fields are available on the **Attributes** configuration screen:

Name:
Enter a suitable name for this filter.

Attributes:
The **Attributes** table lists the checks that the API Gateway performs on user attributes stored in the `attribute.lookup.list` message attribute. The API Gateway performs the following checks:

- The entries in the table are OR-ed together so that if any one of them succeeds, the filter returns a pass result.
- The attribute checks listed in the table are run in series until one of them passes.
- You can add a number of attribute-value pairs to a single attribute check by separating them with commas (for example, `company=oracle, department=engineering, role=engineer`).
- If multiple attribute-value pairs are present in a given attribute check, these pairs are AND-ed together so that the overall attribute check only passes if all the attribute-value pairs pass. For example, if the attribute check comprises, `department=engineering, role=engineer`, this check only passes if both attributes are found with the correct values in the `attribute.lookup.list` message attribute.

To add an attribute check to the **Attributes** table, click **Add**, and enter attributes in the **Add Attributes** dialog.

For attribute checks involving attributes extracted from a SAML attribute assertion, it is necessary to specify the namespace of the attribute as it was given in the assertion. For example, the API Gateway can extract the `role` attribute from the following SAML `<Attribute Statement>`, and store it in the `attribute.lookup.list` map:

```
<saml:AttributeStatement>
 <saml:Attribute Name="role" NameFormat="http://www.company.com">
  <saml:AttributeValue>admin</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute Name="email" NameFormat="http://www.company.com">
  <saml:AttributeValue>joe@company.com</saml:AttributeValue>
 </saml:Attribute>
 <saml:Attribute Name="dept" NameFormat="">
  <saml:AttributeValue>engineering</saml:AttributeValue>
 </saml:Attribute>
</saml:AttributeStatement>
```

The `NameFormat` attribute of the `<Attribute>` gives the namespace of the attribute name. You must enter this namespace (together with a corresponding prefix) in the **Add Attributes** dialog. For example, to extract the `role` attribute from the SAML attribute statement above, enter `pre:role=admin` in the **Attribute Requirement** field. Then you must also map the `pre` prefix to the `http://www.company.com` namespace, as specified by the `NameFormat` attribute in the attribute statement.

Certificate Attributes

Overview

The API Gateway can authorize access to a Web Service based on the X.509 attributes of an authenticated client's certificate. For example, a simple **Certificate Attributes** filter might only authorize clients whose certificates have a Distinguished Name (DName) containing the following attribute: `O=oracle`. In other words, only `oracle` users are authorized to access the Web Service.

An X.509 certificate consists of a number of fields. The `Subject` field is the one of most relevance to this topic. It gives the DName of the client to which the certificate belongs. A DName is a unique name given to an X.500 directory object. It consists of a number of attribute-value pairs called Relative Distinguished Names (RDNs). Some of the most common RDNs and their explanations are as follows:

- `CN`: CommonName
- `OU`: OrganizationalUnit
- `O`: Organization
- `L`: Locality
- `S`: StateOrProvinceName
- `C`: CountryName

For example, the following is the DName of the *sample.p12* client certificate supplied with the API Gateway:

```
CN=Sample Cert, OU=R&D, O=Company Ltd., L=Dublin 4, S=Dublin, C=IE
```

Using the **Certificate Attributes** filter, it is possible to authorize clients based on (for example, the `CN`, `OU`, or `C` in the DName).

Configuration

The **X.509 Attributes** table lists a number of attribute checks to be run against the client certificate. Each entry tests a number of certificate attributes in such a way that the check only passes if all of the configured attribute values match those in the client certificate. In effect, the attributes listed in a single attribute check are AND-ed together.

For example, imagine the following is configured as an entry in the **X.509 Attributes** table:

```
OU=Eng, O=Company Ltd
```

If the API Gateway receives a certificate with the following DName, this attribute check passes because *all* the configured attributes match those in the certificate DName:

```
CN=User1, OU=Eng, O=Company Ltd, L=D4, S=Dublin, C=IE
CN=User2, OU=Eng, O=Company Ltd, L=D2, S=Dublin, C=IE
```

However, if the API Gateway receives a certificate with the following DName, the attribute check fails because the attributes in the DName do not match *all* the configured attributes (the `OU` attribute has the wrong value):

```
CN=User1, OU=qa, O=Company Ltd, L=D4, S=Dublin, C=IE
```

The **X.509 Attributes** table can contain several attribute check entries. In such cases, the attribute checks (the entries in the table) are OR-ed together, so that if any of the checks succeed, the overall **Certificate Attributes** filter succeeds.

So to summarize:

- Attribute values within an attribute check only succeed if *all* the configured attribute values match those in the DName of the client certificate.
- The filter succeeds if *any* of the attribute checks listed in the **X.509 Attributes** table succeed.

To configure a **Certificate Filter** complete the following fields:

Name:
Enter a suitable name for the filter here.

X.509 Attributes:
To add a new X.509 attribute check, click the **Add button** button. In the **Add X.509 Attributes** dialog, enter a comma-separated list of name-value pairs representing the X.509 attributes and their values (for example, `OU=dev,O=Company`).

The new attribute check is displayed in the **X.509 Attributes** table. You can edit and delete existing entries by clicking the **Edit** and **Remove** buttons.

RSA Access Manager Authorization

Overview

RSA Access Manager (formerly known as RSA ClearTrust) provides Identity Management and access control services for Web applications. It centrally manages access to Web applications, ensuring that only authorized users are allowed access to resources.

The API Gateway's **Access Manager** filter enables integration with RSA Access Manager. This filter can query Access Manager for authorization information for a particular user on a given resource. In other words, the API Gateway asks Access Manager to make the authorization decision. If the user has been given authorization rights to the Web Service, the request is allowed through to the service. Otherwise, the request is rejected.

Prerequisites

RSA Access Manager integration requires RSA ClearTrust SDK version 6.0.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

General Details

Configure the following general setting:

Name:
Enter an appropriate name for the filter.

Connection Details

This section enables you to specify a group of Access Manager servers to connect to in order to authenticate clients. You can select a group of Access Manager servers to provide failover in cases where one or more servers are not available.

Connection Group Type
The API Gateway can connect to a group of Access Manager **Authorization Servers** or **Dispatcher Servers**. When multiple Access Manager Authorization Servers are deployed for load-balancing purposes, the API Gateway should first connect to a Dispatcher Server, which returns a list of active Authorization Servers. An attempt is then made to connect to one of these Authorization Servers using round-robin DNS. If the first Dispatcher Server in the Connection Group is not available, the API Gateway attempts to connect to the Dispatcher Server with the next highest priority in the group, and so on.

If a Dispatcher Server has not been deployed, the API Gateway can connect directly to an Authorization Server. If the

Authorization Server with the highest priority in the Connection Group is not available, the API Gateway attempts to connect to the Authorization Server with the next highest priority, and so on.

Select the type of the Connection Group using the **Authorization Server** or **Dispatcher Server** radio button. All servers in the group must be of the same type.

Connection Group:
Click the button on the right, and select the **Connection Group** to use for authenticating clients. To add a Connection Group, right-click the **RSA ClearTrust Connection Sets** tree node, and select **Add a Connection Set**. Alternatively, you can configure a Connection Set under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring Connection Groups*.

Authorization Details

This section describes the resource for which the user is requesting access.

- **Server:**
  Enter the name of the server that is hosting the requested resource. The name entered must correspond to a preconfigured Server Name in Access Manager.
- **Resource:**
  Enter the name of the requested resource. This resource must also have been pre-configured in Access Manager.

Alternatively, you can enter a selector representing a message attribute in the **Resource** field. The API Gateway expands this selector at runtime to the value of the corresponding message attribute. API Gateway message attribute selectors take the following format:

```
${message.attribute}
```

The following example of a typical SOAP message received by the API Gateway shows how this works:

```
POST /services/timeservice HTTP/1.0
Host: localhost:8095
Content-Length: 374
SOAPAction: TimeService
Accept-Language: en-US
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
  </soap:Body>
</soap:Envelope>
```

The following table shows an example of selector expansion:

Selector	***Expanded To***
`${http.request.uri}`	`/services/timeservice`

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Entrust GetAccess Authorization

Overview

Entrust GetAccess provides Identity Management and access control services for Web resources. It centrally manages access to Web applications, enabling users to benefit from a single sign-on capability when accessing the applications that they are authorized to use.

The API Gateway's **GetAccess** filter enables integration with Entrust GetAccess. This filter can query GetAccess for authorization information for a particular user for a given resource. In other words, the API Gateway asks GetAccess to make the authorization decision. If the user has been given authorization rights to the Web Service, the request is allowed through to the Service. Otherwise, the request is rejected.

GetAccess WS-Trust STS

This section configures how the API Gateway authenticates to the GetAccess WS-Trust Security Token Service (STS). You can configure the API Gateway to connect to a group of GetAccess STS servers in a round-robin fashion. This provides the necessary failover capability when one or more STS servers are not available.

Configure the following fields:

- **URL Group:**
  Click the button on the right, and select an STS URL group in the tree. This group consists of a number of GetAccess STS Servers to which the API Gateway round-robins connection attempts. To add a URL group, right-click the **Entrust GetAccess URL Sets** node, and select **Add a URL Set**. Alternatively, you can configure a URL Connection Set under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring URL Groups*.
- **Drift Time:**
  Having successfully authenticated to a GetAccess STS server, the STS server issues a SAML authentication assertion and returns it to the API Gateway. When checking the validity period of the assertion, the specified **Drift Time** is used to account for a possible difference between the time on the STS server and the time on the machine hosting the API Gateway.
- **WS-Trust STS Attribute Field Name:**
  Specify the field name for the `Id` field in the WS-Trust request. The default is `Id`.

GetAccess SAML PDP

When the API Gateway has successfully authenticated to a GetAccess STS server, it can then obtain authorization information about the end-user from the GetAccess SAML PDP. The authorization details are returned in a SAML authorization assertion, which is then validated by the API Gateway to determine whether the request should be denied.

Configure the following fields:

- **URL Group:**
  Click the button on the right, and select an SAML PDP URL group in the tree. This group consists of a number of GetAccess SAML PDP Servers to which the API Gateway round-robins connection attempts. To add a URL group, right-click the **Entrust GetAccess URL Sets** node, and select **Add a URL Set**. Alternatively, you can configure a URL Connection Set under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring URL Groups*.
- **Drift Time:**
  The specified **Drift Time** is used to account for the possible difference between the time on the GetAccess SAML PDP and the time on the machine hosting the API Gateway. This comes into effect when validating the SAML authorization assertion.
- **Resource:**

This is the resource for which the client is requesting access. You can enter a selector representing a message attribute, which is looked up and expanded to a value at runtime. Message attribute selectors have the following format:
`${message.attribute}`
For example, to specify the original path on which the request was received by the API Gateway as the resource, enter the following selector:
`${http.request.uri}`

For more details on selectors, see *Selecting Configuration Values at Runtime*.

- **Actor/Role:**
To add the SAML authorization assertion to the downstream message, select a SOAP actor/role to indicate the WS-Security block where the assertion is added. By leaving this field blank, the assertion is not added to the message.

Insert SAML Authorization Assertion

Overview

After successfully authorizing a client, the API Gateway can insert a Security Assertion Markup Language (SAML) authorization assertion into the SOAP message. Assuming all other security filters in the policy are successful, the assertion will eventually be consumed by a downstream Web Service.

It may be useful to refer to the following example of a signed SAML authorization assertion when configuring this filter.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://.../soap/envelope/">
 <soap:Header xmlns:wsse="http://.../secext">
  <wsse:Security>
   <saml:Assertion
     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
     AssertionID="oracle-1056130475340"
     Id="oracle-1056130475340"
     IssueInstant="2003-06-20T17:34:35Z"
     Issuer="CN=Sample User,...........,C=IE"
     MajorVersion="1"
     MinorVersion="0">
    <saml:Conditions
      NotBefore="2003-06-20T16:20:10Z"
      NotOnOrAfter="2003-06-20T18:20:10Z"/>
    <saml:AuthorizationDecisionStatement
      Decision="Permit"
      Resource="http://www.oracle.com/service">
     <saml:Subject>
      <saml:NameIdentifier
        Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
         sample
      </saml:NameIdentifier>
     </saml:Subject>
    </saml:AuthorizationDecisionStatement>
    <dsig:Signature xmlns:dsig="http://.../xmldsig#" id="Sample User">
      <!-- XML SIGNATURE INSIDE ASSERTION -->
    </dsig:Signature>
   </saml:Assertion>
  </wsse:Security>
 </soap:Header>
 <soap:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </soap:Body>
</soap:Envelope>
```

General Configuration

Configure the following field:

Name:
Enter an appropriate name for the filter.

Assertion Details

Configure the following fields on the **Assertion Details** tab:

Issuer Name:

Select the certificate containing the Distinguished Name (DName) that you want to use as the Issuer of the SAML assertion. This DName is included in the SAML assertion as the value of the `Issuer` attribute of the `<saml:Assertion>` element. For an example, see the sample SAML assertion above.

Expire In:
Specify the lifetime of the assertion in this field. The lifetime of the assertion lasts from the time of insertion until the specified amount of time has elapsed.

Drift Time:
The **Drift Time** is used to account for differences in the clock times of the machine hosting the API Gateway (that generate the assertion) and the machines that consume the assertion. The specified time is subtracted from the time at which the API Gateway generates the assertion.

SAML Version:
You can create SAML 1.0, 1.1, and 2.0 attribute assertions. Select the appropriate version from the drop-down list.

Important

SAML 1.0 recommends the use of the `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` XML Signature Canonicalization algorithm. When inserting signed SAML 1.0 assertions into XML documents, it is quite likely that subsequent signature verification of these assertions will fail. This is due to the side effect of the algorithm including inherited namespaces into canonical XML calculations of the inserted SAML assertion that were not present when the assertion was generated.

For this reason, Oracle recommend that SAML 1.1 or 2.0 is used when signing assertions as they both uses the exclusive canonical algorithm `http://www.w3.org/2001/10/xml-exc-c14n#`, which safeguards inserted assertions from such changes of context in the XML document. Please see section 5.4.2 of the `oasis-sstc-saml-core-1.0.pdf` and section 5.4.2 of `sstc-saml-core-1.1.pdf` documents, both of which are available at `http://www.oasis-open.org`.

Resource:
Enter the resource for which you want to obtain the authorization assertion. You should specify the resource as a URI (for example, `http://www.oracle.com/TestService`). The name of the resource is then included in the assertion.

Action:
You can specify the operations that the user can perform on the resource using the **Action** field. This entry is a comma-separated list of permissions. For example, the following is a valid entry: `read`, `write`, `execute`.

Assertion Location

The options on the **Assertion Location** tab specify where the SAML assertion is inserted in the message. By default, the SAML assertion is added to the WS-Security block with the current SOAP actor/role. The following options are available:

Append to Root or SOAP Header:
Appends the SAML assertion to the message root for a non-SOAP XML message, or to the SOAP Header for a SOAP message. For example, this option may be suitable for cases where this filter may process SOAP XML messages or non-SOAP XML messages.

Add to WS-Security Block with SOAP Actor/Role:
Adds the SAML assertion to the WS-Security block with the specified SOAP actor (SOAP 1.0) or role (SOAP 1.1). By default, the assertion is added with the current SOAP actor/role only, which means the WS-Security block with no actor. You can select a specific SOAP actor/role when available from the drop-down list.

XPath Location:
If you wish to insert the SAML assertion at an arbitrary location in the message, you can use an XPath expression to specify the exact location in the message. You can select XPath expressions from the drop-down list. The default is the `First WSSE Security Element`, which has an XPath expression of `//wsse:Security`. You can add, edit, or re-

move expressions by clicking the relevant button. For more details, see the *Configuring XPath Expressions* topic.

You can also specify how exactly the SAML assertion is inserted using the following options:

- **Append to node returned by XPath expression** (the default)
- **Insert before node returned by XPath expression**
- **Replace node returned by XPath expression**

Insert into Message Attribute:
Specify a message attribute to store the SAML assertion from the drop-down list (for example, `saml.assertion`). Alternatively, you can also enter a custom message attribute in this field (for example, `my.test.assertion`). The SAML assertion can then be accessed downstream in the policy.

Subject Confirmation Method

The settings on the **Subject Confirmation Method** tab determines how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the `<SubjectConfirmation>` block can be used to authenticate either the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical `<SubjectConfirmation>` block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

The following configuration fields are available on the **Subject Confirmation Method** tab:

Method:
The selected value determines the value of the `<ConfirmationMethod>` element. The following table shows the available methods, their meanings, and their respective values in the `<ConfirmationMethod>` element:

Method	***Meaning***	***Value***
Holder Of Key	The API Gateway includes the key used to prove that the API Gateway is the holder of the key, or it includes a reference to the key.	`urn:oasis:names:tc:SAML:1.0:cm:holder-of-key`
Bearer	The subject of the assertion is the bearer of the assertion.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`
SAML Artifact	The subject of the assertion is the user that presented a SAML Artifact to the API Gateway.	`urn:oasis:names:tc:SAML:1.0:cm:artifact`
Sender Vouches	Use this confirmation method to assert that the API Gateway is acting on behalf of the authenticated end-user. No	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	*Meaning*	*Value*
	other information relating to the context of the assertion is sent. It is recommended that both the assertion **and** the SOAP Body must be signed if this option is selected. These message parts can be signed by using the *XML Signature Generation* filter.	

Note

You can also leave the **Method** field blank, in which case no `<ConfirmationMethod>` block is inserted into the assertion.

Holder-of-Key Configuration:
When you select `Holder-of-Key` as the SAML subject confirmation in the **Method** field, you must configure how information about the key is included in the message. There are a number of configuration options available depending on whether the key is a symmetric or asymmetric key.

Asymmetric Key:
If you want to use an asymmetric key as proof that the API Gateway is the holder-of-key entity, you must select the **Asymmetric Key** radio button, and then configure the following fields on the **Asymmetric** tab:

- **Certificate from Store:**
  If you want to select a key that is stored in the Certificate Store, select this option and click the **Signing Key** button. On the **Select Certificate** screen, select the box next to the certificate that is associated with the key that you want to use.
- **Certificate from Message Attribute:**
  Alternatively, the key may have already been used by a previous filter in the policy (for example, to sign a part of the message). In this case, the key is stored in a message attribute. You can specify this message attribute in this field.

Symmetric Key:
If you want to use a symmetric key as proof that the API Gateway is the holder of key, select the **Symmetric Key** radio button, and then configure the fields on the **Symmetric** tab:

- **Generate Symmetric Key, and Save in Message Attribute:**
  If you select this option, the API Gateway generates a symmetric key, which is included in the message before it is sent to the client. By default, the key is saved in the `symmetric.key` message attribute.
- **Symmetric Key in Message Attribute:**
  If a previous filter (for example, a **Sign Message** filter) has already used a symmetric key, you can to reuse this key as proof that the API Gateway is the holder-of-key entity. You must enter the name of the message attribute in the field provided, which defaults to `symmetric.key`.
- **Encrypt using Certificate from Certificate Store:**
  When a symmetric key is used, you must assume that the recipient has no prior knowledge of this key. It must, therefore, be included in the message so that the recipient can validate the key. To avoid meet-in-the-middle style attacks, where a hacker could eavesdrop on the communication channel between the API Gateway and the recipient and gain access to the symmetric key, the key must be encrypted so that only the recipient can decrypt the key. One way of doing this is to select the recipient's certificate from the Certificate Store. By encrypting the symmetric key with the public in the recipient's certificate, the key can only be decrypted by the recipient's private key, to which only the recipient has access. Select the **Signing Key** button and then select the recipient's certificate on the **Select Certificate** dialog.

- **Encrypt using Certificate from Message Attribute:**
  Alternatively, if the recipient's certificate has already been used (perhaps to encrypt part of the message) this certificate is stored in a message attribute. You can enter the message attribute in this field.
- **Symmetric Key Length:**
  Enter the length (in bits) of the symmetric key to use.
- **Key Wrap Algorithm:**
  Select the algorithm to use to encrypt (*wrap*) the symmetric key.

Key Info:
The **Key Info** tab must be configured regardless of whether you have elected to use symmetric or asymmetric keys. It determines how the key is included in the message. The following options are available:

- **Do Not Include Key Info:**
  Select this option if you do not wish to include a `<KeyInfo>` section in the SAML assertion.
- **Embed Public Key Information:**
  If this option is selected, details about the key are included in a `<KeyInfo>` block in the message. You can include the full certificate, expand the public key, include the distinguished name, and include a key name in the `<KeyInfo>` block by selecting the appropriate boxes. When selecting the **Include Key Name** field, you must enter a name in the **Value** field, and then select the **Text Value** or **Distinguished Name Attribute** radio button, depending on the source of the key name.
- **Put Certificate in Attachment:**
  Select this option to add the certificate as an attachment to the message. The certificate is then referenced from the `<KeyInfo>` block.
- **Security Token Reference:**
  The Security Token Reference (STR) provides a way to refer to a key contained in a SOAP message from another part of the message. It is often used in cases where different security blocks n a message use the same key material and it is considered an overhead to include the key more than once in the message.
  When this option is selected, a `<wsse:SecurityTokenReference>` element is inserted into the `<KeyInfo>` block. It references the key material using a `URI` to point to the key material and a `ValueType` attribute to indicate the type of reference used. For example, if the STR refers to an encrypted key, you should select `EncryptedKey` from the dropdown, whereas if it refers to a `BinarySecurityToken`, you should select `X509v3` from the dropdown. Other options are available to enable more specific security requirements.

Advanced

Configure the following fields on the **Advanced** tab:

Select Required Layout Type:
WS-Policy and SOAP Message Security define a set of rules that determine the layout of security elements that appear in the WS-Security header within a SOAP message. The SAML assertion is inserted into the WS-Security header according to the layout option selected here. The available options correspond to the WS-Policy Layout assertions of `Strict`, `Lax`, `LaxTimestampFirst`, and `LaxTimestampLast`.

Insert SAML Attribute Statement:
You can specify to insert a SAML attribute statement into the generated SAML authorization assertion. If this option is selected, a SAML attribute assertion is generated using attributes stored in the `attribute.lookup.list` message attribute and subsequently inserted into the assertion. The `attribute.lookup.list` attribute must have been populated previously by an attribute lookup filter for the attribute statement to be generated successfully.

Indent:
Select this method to ensure that the generated signature is properly indented.

Security Token Reference:
The generated SAML authorization assertion can be encapsulated within a `<SecurityTokenReference>` block. The following example demonstrates this:

```
<soap:Header>
 <wsse:Security
     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
     soap:actor="oracle">
  <wsse:SecurityTokenReference>
   <wsse:Embedded>
    <saml:Assertion
      xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
      AssertionID="oracle-1056130475340"
      Id="oracle-1056130475340"
      IssueInstant="2003-06-20T17:34:35Z"
      Issuer="CN=Sample User,...........,C=IE"
      MajorVersion="1"
      MinorVersion="0">
     <saml:Conditions
       NotBefore="2003-06-20T16:20:10Z"
       NotOnOrAfter="2003-06-20T18:20:10Z"/>
     <saml:AuthorizationDecisionStatement
       Decision="Permit"
       Resource="http://www.oracle.com/service">
      <saml:Subject>
       <saml:NameIdentifier
         Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
          sample
       </saml:NameIdentifier>
      </saml:Subject>
     </saml:AuthorizationDecisionStatement>
     <dsig:Signature xmlns:dsig="http://.../xmldsig#" id="Sample User">
       <!-- XML SIGNATURE INSIDE ASSERTION -->
     </dsig:Signature>
    </saml:Assertion>
   </wsse:Embedded>
  </wsse:SecurityTokenReference>
 </wsse:Security>
</soap:Header>
```

To add the SAML assertion to a `<SecurityTokenReference>` block as in the example above, select the **Embed SAML assertion within Security Token Reference** option. Otherwise, select **No Security Token Reference**.

RBAC Filter

Overview

Role-Based Access Control (RBAC) is used to protect access to the API Gateway management services. For example, management services are invoked when a user accesses the server using the Policy Studio or the API Gateway Manager tools (`https://localhost:8090/`). For more information, see *Configuring Role-Based Access Control (RBAC)*.

The **RBAC** filter is used in the **Protect Management and Policy Director Interfaces** policy to perform the following tasks:

- Read the user roles from the configured message attribute (for example, `authentication.subject.role`).
- Determine which management service URI is currently being invoked.
- Return true if one of the roles has access to the management service currently being invoked, as defined in the `acl.json` file.
- Otherwise, return false, and the **Return HTTP Error 403: Access Denied (Forbidden)** policy is called. The message content of this filter is shown when a valid user has logged into the browser, but their roles do not give them access to the URI they have invoked. For example, this occurs if a new user is created and they have not yet been assigned any roles.

Configuration

Name:
Enter an appropriate name for this filter.

Role Attribute:
Select or enter the message attribute that contains the user roles. Defaults to `authentication.subject.role`.

SAML Authorization Assertion

Overview

A Security Assertion Markup Language (SAML) authorization assertion contains proof that a certain user has been authorized to access a specified resource. Typically, such assertions are issued by a SAML Policy Decision Point (PDP) when a client requests access to a specified resource. The client must present identity information to the PDP, which ensures that the client does have permission to access the resource. The PDP then issues a SAML authorization assertion stating whether the client is allowed access the resource.

When the API Gateway receives a request containing such an assertion, it performs the following validation on the assertion details:

- Ensures the resource in the assertion matches that configured in the filter
- Checks the client's access permissions for the resource
- Ensures the assertion has not expired

If the information validates, the API Gateway authorizes the message for the resource specified in the assertion.

When configuring this filter, it may be useful to refer to the following SAML authorization assertion as an example:

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            MajorVersion="1" MinorVersion="0"
            AssertionID="192.168.0.131.1010924615489"
            Issuer="AA" IssueInstant="2002-03-26 16:23:35">
    <saml:Conditions NotBefore="2002-04-18T09:19:00Z"
                     NotOnOrAfter="2003-06-28T09:21:00Z"/>
    <saml:AuthorizationDecisionStatement
      Resource="http://www.abc.org/services/getPrice"
      Decision="Permit">
        <saml:Action>Read</saml:Action>
    </saml:AuthorizationDecisionStatement>
</saml:Assertion>
```

General Settings

Configure the following field on the **SAML Authorization** screen:

Name:
Enter an appropriate name for the filter.

Details

The following fields are available on the **Details** tab:

SOAP Actor/Role:
There may be several authorization assertions contained in a message. You can identify the assertion to validate by entering the name of the SOAP actor/role of the WS-Security header that contains the assertion.

XPath Expression:
Alternatively, you can enter an XPath expression to locate the authorization assertion. You can configure XPath expressions using the **Add**, **Edit** and **Delete** buttons.

SAML Namespace:
Select the SAML namespace that must be used on the SAML assertion for this filter to succeed. If you do not wish to

check the namespace, select the `Do not check version` option from the drop-down list.

SAML Version:
Enter the SAML Version that the assertion must adhere to by entering the major version in the first field, followed by the minor version in the second field. For example, for SAML version 2.0, enter `2` in the first field and `0` in the second field.

Drift Time:
The *drift time*, specified in seconds, is used when checking the validity dates on the authorization assertion. The drift time allows for differences between the clock times of the machine on which the assertion was generated and the machine hosting the API Gateway.

Remove Enclosing WS-Security Element on Successful Validation:
Select this checkbox if you wish to remove the WS-Security block that contains the SAML assertion after the assertion has been successfully validated.

Trusted Issuers

You can use the table on this tab to select the issuers that you consider trusted. In other words, this filter only accepts assertions that have been issued by the selected SAML Authorities.

Click the **Add** button to display the **Trusted Issuers** screen. Select the Distinguished Name of a SAML Authority whose certificate has been added to the Certificate Store, and click **OK**. Repeat this step to add more SAML Authorities to the list of trusted issuers.

Optional Settings

The optional settings enable further examination of the contents of the authorization assertion. The assertion can be checked to ensure that the authorized subject matches a specified value, and that the resource specified in the assertion matches the one entered here.

The API Gateway can verify that the subject in the SAML assertion (the `<NameIdentifier>`) matches one of the following options:

- **The subject of the authentication filter**
- **The following value**: (for example, `user@oracle.com`)
- **Neither of the above**

The API Gateway examines the `<Resource>` tag inside the SAML authorization assertion. By default, it compares the `<Resource>` to the `destination.uri` attribute that is set in the policy. If they are identical, this filter passes. Otherwise, it fails.

You can enter a value for the resource in the **Resource** field. The API Gateway then compares the `<Resource>` in the assertion to this value instead of the `destination.uri` attribute. The filter passes if the `<Resource>` value matches the value entered in the **Resource** field.

SAML PDP Authorization

Overview

The API Gateway can request an authorization decision from a Security Assertion Markup Language(SAML) Policy Decision Point (PDP) for an authenticated client using the SAML Protocol (SAMLP). In such cases, the API Gateway presents evidence to the PDP in the form of some user credentials, such as the Distinguished Name of a client's X.509 certificate.

The PDP decides whether the user is authorized to access the requested resource. It then creates an authorization assertion, signs it, and returns it to the API Gateway in a SAML Protocol response. The API Gateway can then perform a number of checks on the response, such as validating the PDP signature and certificate, and examining the assertion. It can also insert the SAML authorization assertion into the message for consumption by a downstream Web Service.

Request Configuration

This section describes how the API Gateway should package the SAMLP request before sending it to the SAML PDP.

SAML PDP URL Sets
You can configure a group of SAML PDPs to which the API Gateway connects in a round-robin fashion if one or more of the PDPs are unavailable. This is known as a SAML PDP URL Set. You can configure a SAML PDP URL Set using this screen or under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring URL Groups*.

You can configure the following general fields:

- **SAML PDP URL Set:**
  Click the button on the right, and select a previously configured SAML PDP URL Set in the tree. To add a URL Set, right-click the **SAML PDP URL Sets** tree node, and select **Add a URL Set**. Alternatively, you can configure a SAML PDP URL Set under the **External Connections** node in the Policy Studio tree.
- **SAML Version:**
  Select the SAML version to use in the SAMLP request.
- **Signing Key:**
  If the SAMLP request is to be signed, click the **Signing Key** button, and select the appropriate signing key from the Certificate Store.

SAML Subject tab
The details specified on the **SAML Subject** tab describe the *subject* of the SAML assertion. Complete the following fields:

- **Subject Attribute:**
  Select the message attribute that contains the name of an authenticated username. By default, the `authentication.subject.id` message attribute is selected, which contains the username of the authenticated user.
- **Subject Format:**
  Select the format of the message attribute selected in the **Subject Attribute** field above. You do not need to select a format if the **Subject Attribute** field is set to `authentication.subject.id`.

Subject Confirmation tab
The settings on the **Subject Confirmation** tab determine how the `<SubjectConfirmation>` block of the SAML assertion is generated. When the assertion is consumed by a downstream Web Service, the information contained in the `<SubjectConfirmation>` block can be used to authenticate the end-user that authenticated to the API Gateway, or the issuer of the assertion, depending on what is configured.

The following is a typical `<SubjectConfirmation>` block:

```
<saml:SubjectConfirmation>
  <saml:ConfirmationMethod>
    urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </saml:ConfirmationMethod>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
      <dsig:X509Data>
        <dsig:X509SubjectName>CN=oracle</dsig:X509SubjectName>
        <dsig:X509Certificate>
          MIICmzCCAY ...... mB9CJEw4Q=
        </dsig:X509Certificate>
      </dsig:X509Data>
    </dsig:KeyInfo>
  </saml:SubjectConfirmation>
</saml:SubjectConfirmation>
```

You must configure the following fields on the **Subject Confirmation** tab:

Method:
The selected value determines the value of the `<ConfirmationMethod>` element. The following table shows the available methods, their meanings, and their respective values in the `<ConfirmationMethod>` element:

Method	***Meaning***	***Value***
Holder Of Key	Inserts a `<SubjectConfirmation>` into the SAMLP request. The `<SubjectConfirmation>` contains a `<dsig:KeyInfo>` section with the certificate of the user selected to sign the SAMLP request. The user selected to sign the SAMLP request must be the authenticated subject (`authentication.subject.id`). Select the **Include Certificate** option if the signer's certificate is to be included in the `SubjectConfimration` block. Alternatively, select the **Include Key Name** option if only the key name is to be included.	`urn:oasis:names:tc:SAML:1.0:cm:holder-of-key`
Bearer	Inserts a `<SubjectConfirmation>` into the SAMLP request.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`
SAML Artifact	Inserts a `<SubjectConfirmation>` into the SAMLP request.	`urn:oasis:names:tc:SAML:1.0:cm:artifact`
Sender Vouches	Inserts a `<SubjectConfirmation>` into the SAMLP request. The SAMLP request must be signed by a user.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

If the **Method** field is left blank, no `<ConfirmationMethod>` block is inserted into the assertion.

Include Certificate:
Select this option if you wish to include the SAML subject's certificate in the `<KeyInfo>` section of the `<SubjectConfirmation>` block.

Include Key Name:
Alternatively, if you do not want to include the certificate, you can select this option to only include the key name in the

`<KeyInfo>` section.

Resource:
Enter the resource for which you want to obtain the authorization assertion. You should specify the resource as a URI (for example, `http://www.oracle.com/TestService`). The name of the resource is then included in the assertion.

Evidence:
The SAML Protocol stipulates that proof of identity in the form of a SAML authentication assertion must be presented to the SAML PDP as part of the SAMLP request. The API Gateway can either use an existing SAML authentication assertion that is already present in the message, or it can generated one based on the user that authenticated to it.

Select the **Use SAML Assertion in message** option to include an existing assertion in the SAMLP request. Specify the actor/role of the WS-Security block where the assertion can be found in the **SOAP Actor/Role** field.

Alternatively, select the **Create SAML Assertion from authenticated client** radio button to generate a new authentication assertion for inclusion in the SAMLP request. You can sign the newly generated assertion by selecting a key from the drop-down list, which shows all the keys from the Certificate Store.

The specified **Drift Time** is subtracted from the time at which the API Gateway generates the authentication assertion. This is to account for any possible difference in the times of the machines hosting the SAML PDP and the API Gateway.

Response

You can configure the API Gateway to perform a number of checks on the SAML Protocol response from the PDP by examining the contents of various key elements in the authorization assertion.

SOAP Actor/Role:
If the SAMLP response from the PDP contains a SAML authentication assertion, the API Gateway can extract it from the response and insert it into the downstream message. The SAML assertion is inserted into the WS-Security block identified by the specified SOAP actor/role.

Drift Time:
The SAMLP request to the PDP is timestamped by the API Gateway. To account for differences in the times on the machines running the API Gateway and the SAML PDP the specified time is subtracted from the time at which the API Gateway generates the SAMLP request.

Subject in the Assertion Must Match:
The authorization assertion can be checked to ensure that the authorized subject matches a specified value, and that the resource specified in the assertion matches the one entered here.

The API Gateway can verify that the subject in the SAML assertion (the `<NameIdentifier>`) matches one of the following options:

- **The subject of the authentication filter**
- **The following value** (for example, `CN=sample, O=Company, C=ie`)
- **Neither of the above**

Tivoli Integration

Overview

Oracle API Gateway is a dedicated network device for offloading processor-intensive tasks from applications running in general purpose application servers. The API Gateway performs application networking by routing traffic based on both content and sender. Its patented high performance XML acceleration engine, coupled with acceleration hardware ensures wirespeed network performance.

Tivoli Access Manager is a commonly used product for securing web resources. The Tivoli message filter allows the API Gateway to leverage existing Access Manager policies, thus avoiding the need to maintain duplicate policies in both products. At runtime, the Tivoli filters can delegate authentication and authorization decision to Access Manager, and can also retrieve user attributes. Therefore, the API Gateway integrates with Tivoli Access Manager by providing the following functionality:

- Connects to Tivoli Access Manager
- Authenticates a user against Access Manager
- Authorizes a user against Access Manager
- Retrieves user attributes from Access Manager

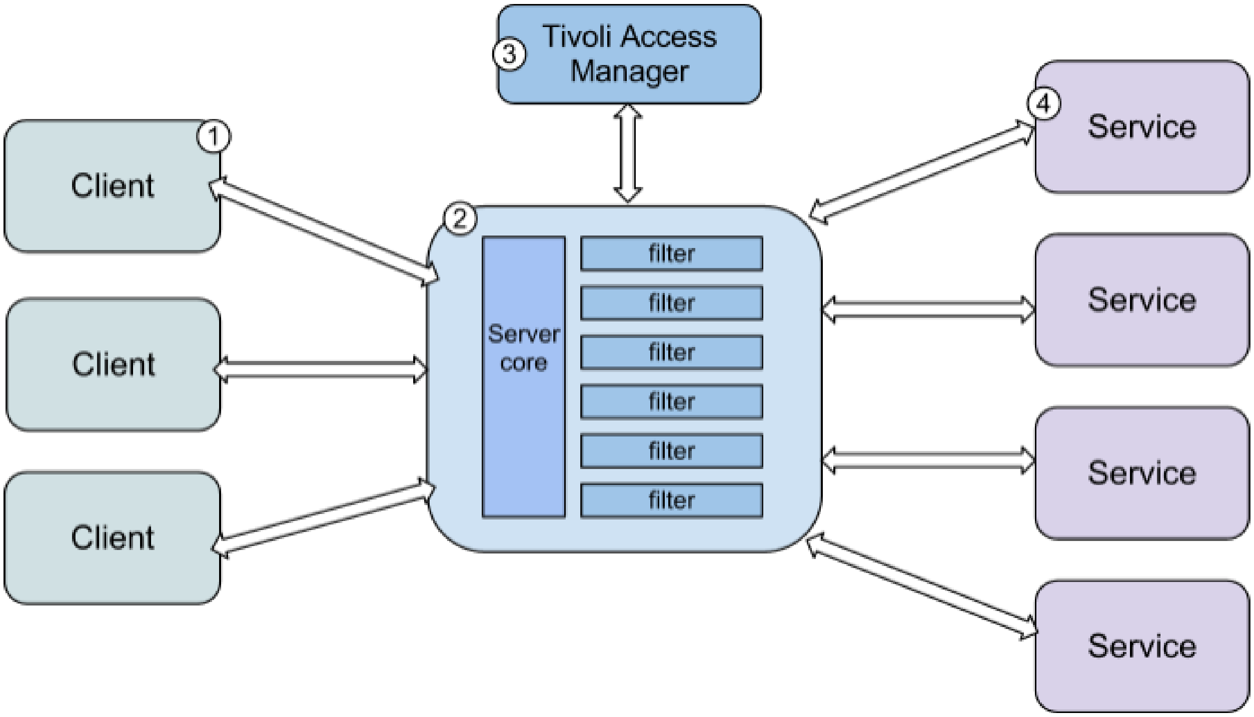
The API Gateway has been built to integrate with Tivoli Access Manager 6.0.

Integration Architecture

The Oracle API Gateway contains a set of message filters that directly or indirectly restrict access to Web Services. For example, filters that directly control access include XML-signature verification, CA certificate chain verification, and SAML assertion verification. With this class of filters, policy decisions are made and enforced within the API Gateway's core engine itself.

On the other hand, filters that indirectly control access offload the policy decision to an external system, such as Tivoli Access Manager. With indirect filters, the policy decision is made by the external system but then enforced by the API Gateway.

The objective of this integration solution is to implement a message filter that forwards policy decisions to IBM's Tivoli Policy Director/Access Manager. The architecture can be seen in the following diagram.



The following processing stages are executed:

1. The client sends a message to the API Gateway (for example, using SOAP over HTTPS).
2. The API Gateway dispatches the message to the appropriate policy. The filters configured for that policy are then executed.
3. Assuming that the Tivoli filter is one of the filters that is configured for this policy, the API Gateway asks Tivoli Access Manager to authenticate, authorize, or retrieve attributes for a given user. Tivoli Access Manager makes its security decision and returns it to the API Gateway, where the decision is enforced.
4. If Tivoli Access Manager successfully authenticates or authorizes the user, or can retrieve attributes about that user, the message is routed on to the configured target system. Otherwise, the message is blocked and a fault is returned to the client.

Prerequisites

IBM Tivoli integration requires the following:

Tivoli API:
IBM Tivoli Access Manager requires the IBM Tivoli Access Manager for e-business Authorization C API.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Tivoli Runtime:
The Tivoli Access Manager Runtime must be installed on the machine running the API Gateway.

Note

The Tivoli Access Manager Runtime for Java is *not* required. The Tivoli runtime is not packaged with the API Gateway product, so the IBM installers need to run to install the runtime.

The Tivoli Access Manager Runtime may be installed using the native utilities instead of the Installation Wizard. This is advised so that the IBM Java Runtime 1.4.2 does *not* get installed. The Java Runtime 1.4.2 is required by the Installation Wizard, but not by any of the runtime software.

Tivoli Configuration Files:
The API Gateway uses information stored in the Tivoli configuration files in order to connect to a Tivoli server. These configuration files can be generated using the **svrsslcfg** command line utility, which is shipped with the Tivoli Access Manager Runtime discussed in the previous section. The generated configuration files are then either uploaded to the API Gateway using Oracle Policy Studio or manually copied into a location on the API Gateway's file system.

The following example shows how to run the **svrsslcfg** utility (using Windows file paths):

```
 svrsslcfg -config -f "C:\conf\config.conf" -d "C:\conf" -n API Gateway
-s remote -P passw0rd -S passw0rd -r 7777 -h test.vordel.com
```

The following list describes the available arguments:

- **-config**
  Create the configuration files required for the API Gateway to communicate with Tivoli.
- **-f**
  The name of the main Tivoli configuration file. This file is generated by the command.
- **-d**
  The name of the directory that is to contain the SSL key file (`.kdb`) for the server. This command generates the key file.
- **-n**
  The name of the application that is connecting to Tivoli (the API Gateway).
- **-s**
  The mode in which the application (the API Gateway) runs. The most likely scenario is that the API Gateway runs remotely.
- **-P**
  The administrator's password.
- **-S**
  The password for the API Gateway.
- **-r**
  The listening port for the API Gateway.
- **-h**
  The name of the host on which the API Gateway is running.

After the **svrsslcfg** utility has been run with the **-config** option, the following command must be run:

```
svrsslcfg -add_replica -f "c:\conf\config.conf" -h tivoli.qa.vordel.com
```

The following list describes the available arguments:

- **-add_replica**
  Add a Tivoli authorization server replica. The API Gateway contacts this server to make authorization decisions.
- **-f**
  The name of the main Tivoli configuration file. This command adds settings to this file.
- **-h**
  The name of the Tivoli authorization server.

Generated Files
The following files are generated after running these commands:

- **c:\conf\config.conf** - The main Tivoli configuration file.
- **c:\conf\API Gateway.kdb** - The SSL key file.
- **c:\conf\API Gateway.sth** - The stash file for the SSL key file.
- **c:\conf\API Gateway.conf.obf** - The database configuration file.

Note

Depending on the version of the API Gateway you are running, the above file names may or may not have spaces in them.

Please refer to the Tivoli documentation for more information on running these command line utilities.

Creating a Tivoli Object Space:
An object space, user, and ACL (Access Control List) may be created within Tivoli using the **pdadmin** command as follows:

```
> login -a sec_master  passw0rd
> objectspace create /vordel/test "For testing purposes" 9
> user create -gsouser jsmith "cn=John Smith, o=Vordel" "John Smith" Smith passw0rd
> user modify jsmith account-valid yes
> acl create VordelACL
> acl modify VordelACL set user jsmith brT
> acl attach /vordel/test VordelACL
```

The following commands allow you to view the details of the newly added user:

```
> user show jsmith
> objectspace list
> acl show VordelACL
```

Please refer to the Tivoli documentation for more information on running the **pdadmin** utility.

Global Tivoli Configuration

Policy Studio is used to configure all Tivoli connections and settings within the API Gateway. It can be run from the `/bin` directory of your product installation.

There are two global Tivoli-specific settings that can be configured:

- Tivoli Connections
- Tivoli Repositories

Tivoli Connections:
Tivoli Connections determine how a particular API Gateway instance connects to an instance of a Tivoli server. Each API Gateway instance can connect to a single Tivoli server. This connection can be configured by right-clicking the API Gateway instance under the **Listeners** node in the tree on the left of the Policy Studio, and clicking the **Tivoli** menu option.

Alternatively, you can add a global Tivoli Connection by right-clicking the **External Connections** -> **Tivoli Connection** node in the tree, and selecting the **Add a Tivoli Connection** option. The newly added connection can then be assigned to a particular API Gateway instance.

In both cases, the **Tivoli Configuration** dialog is used to add the connection details required for the API Gateway to connect to the Tivoli server. As stated in the Prerequisites section above, the connection details are stored in the Tivoli configuration files that are generated by the **svrsslcfg** utility. This dialog allows you to upload these files to the API Gateway.

The Policy Studio can be used to upload the Tivoli configuration files to the machine on which the API Gateway is running or, alternatively, the configuration files may be copied manually using the tool of your choice onto the API Gateway's file system. This section now describes how to perform each of these methods in turn.

Complete the following steps to upload a configuration file to the API Gateway:

1. Enter a name on the **Tivoli Configuration** dialog. A previously configured Tivoli Connection can be selected to base the new configuration on.
2. Select the **Upload Tivoli configuration files** option.
3. Select the version of the Tivoli server that this connection connects to. Both Tivoli 5.1 and 6.0 are supported.
4. Check the **Connection enabled** checkbox if you want to immediately enable the connection. It can be disabled at a later stage by toggling this checkbox. Click the **Next** button.
5. On the **Upload Tivoli configuration files** screen, click the **Load File** button and browse to the location of the main *Tivoli configuration* file. The contents of this file are then displayed in the text area. Any of the details can be edited in the text area at this stage if required.
   For example, it may be necessary to change the file locations of the configuration files. This is because when you use the **Upload ...** option, the API Gateway writes out the files on startup and on server update to the following directory, where `PROCESS_NAME` is API Gateway instance and `INSTALL_DIR` refers to the root of your product installation:
   `[INSTALL_DIR]\conf\plugin\tivoli\[PROCESS_NAME]`

   ### Note

   Spaces are substituted with `-` in the API Gateway instance name.In addition, the API Gateway names each file as `config.[EXTENSION]`. For example, the directory, `[INSTALL_DIR]\conf\plugin\tivoli\API Gateway` contains `config.conf`, `config.kdb`, `config.sth`, and `config.conf.obf`. The API Gateway overwrites these files each time at startup or refresh (for example, when configuration updates are deployed). This means that any changes to the main configuration file must be made using the Policy Studio and not directly to the file on disk.

6. Click the **Next** button.
7. Click the **Load File** button and browse to the location of the *Tivoli SSL key file*. Once again, the contents of this file are displayed in the text area.

   ### Note

   In this case, the (base-64 encoded) SSL keys can *not* be edited in the text area. Click the **Next** button.

8. Click the **Load File** button and browse to the location of the *Tivoli SSL stash* file. Click the **Next** button.
9. Click the **Load File** button and browse to the location of the *Tivoli Configuration database configuration* file. Click the **Finish** button to upload all the selected files.

Alternatively, the configuration files may be copied manually onto the API Gateway's file system. Having done this using some out-of-bounds method, complete the following steps to configure the API Gateway to pick up the uploaded files:

1. Enter a name on the **Tivoli Configuration** dialog.
2. Select the **Set file location for main Tivoli Configuration file** option and click the **Next** button. Click the **Next** button.
3. Enter the full path to the main Tivoli configuration file on the server's file system in the **Server-side Tivoli configuration** field. Click the **Finish** button.
4. If you have not already manually copied the configuration files on to the API Gateway's file system you should do so now. Please ensure that the settings contained in the main configuration file that point to other configuration filenames are set correctly.

Note

When the **Set file location** option is selected, the API Gateway does not overwrite the files at startup or refresh time. You may edit the main configuration file directly using an editor of your choice.

Tivoli Repositories:
A Tivoli Repository is used to authenticate clients against a running instance of a Tivoli server. All authentication filters can pass identity credentials to the Tivoli Repository in order to authenticate clients. The Tivoli server decides whether or not to authenticate the client and the API Gateway subsequently enforces the decision.

Tivoli Repositories can be configured globally by right clicking on the **Tivoli Repositories** node in the tree on the Policy Studio and selecting the **Add** option.

Enter a name for the Repository in the **Repository Name** field on the **Authentication Repository** dialog. Select the **Finish** button to complete the configuration. You may specify Tivoli Connection details from the **Repository Configuration** screen or via the **Settings** button. On the **Tivoli Configuration** dialog, select the API Gateway instance whose connection details you want to configure, then follow the steps outlined above in the Tivoli Connections section.

When configuring an authentication filter, you can select this globally configured Tivoli Repository to authenticate clients against. The authentication filter uses the connection details of whatever API Gateway instance was selected.

Tivoli Authorization

The **Tivoli Authorization** filter can be found in the **Authorization** category of filters. To configure this filter, drag and drop it on to the policyt editor and configure the following fields:

Name:
Enter a name for the Tivoli filter here.

Object Space:
The object space represents the resource for which the client must be authorized. Enter the name of the resource in the **Object Space** field.

You can also enter a selector that represents the value of a message attribute. At runtime, the API Gateway expands the selector to current value of the corresponding message attribute.

Message attribute selectors have the following format:
`${message.attribute}`
For example, to specify the original path on which the request was received as the resource, enter the following selector:

```
${http.request.uri}
```

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Permissions:
Clients can access a resource with a number of permissions such as read, write, execute, and so on. A client is only authorized to access the requested resource if it has the relevant permissions checked in the **Action Bit** table. You can edit existing permissions by clicking the **Edit** button.

Attributes:
You can specify a list of user attributes to retrieve from the Tivoli server. You can add attributes to be retrieved can be added by clicking the **Add** button and entering the attribute name in the dialog. If you want all attributes to be retrieved, leave the table blank, and select the **Set attributes for SAML Attribute token** option. These attributes can then be made available to the **Insert SAML Attribute Assertion** filter at a later stage. If you do not require any attribute retrieval, do not select the **Set attributes for SAML Attribute token** option.

Note

The permissions for the `primary` action group are available by default. You can also configure custom action groups and make them available for selection in the filter. The groups created here reflect custom groups created on the Tivoli server. To create a new group with custom action bits, click the **Edit** button to display the **Tivoli Action Group** dialog.

Enter a name for the group in the **Name** field. Click the **Add** button to add a new action bit to the group. The **Tivoli Action** dialog is displayed. You must enter an **Action Bit** (for example, `r`) and a **Description** (for example, `Read permission`) for the new action bit. Click the **OK** button on the **Tivoli Action** dialog to return to the **Tivoli Action Group** dialog.

Add as many action bits as required to your new group before clicking **OK** on the **Tivoli Action Group** dialog. The new action bits are then available for selection in the table on the main filter screen.

Tivoli Configuration Files:
As stated earlier, a Tivoli configuration file that contains all the required connection details is associated with a particular Oracle API Gateway instance. Click the **Settings** button to display the **Tivoli Configuration** dialog.

On the **Tivoli Configuration** dialog, select the API Gateway instance whose connection details you want to configure, then follow the steps as outlined above in the Tivoli Connections section.

Note

You do not have to configure the Tivoli Connection for each filter or Authentication Repository. The **Settings** button is placed on the filter screen as a convenient way to access the Tivoli Connection settings. The Tivoli Connection needs to be configured once per API Gateway instance.

Tivoli Authentication

It is possible to authenticate clients against a Tivoli Access Manager repository. In this way, the API Gateway can leverage existing Tivoli security policies without the need to duplicate policies across both products.

The Tivoli repository is available from all authentication filters. However, for demonstration purposes, assume that you want to use the HTTP Basic Authentication filter to authenticate a client against a Tivoli Repository using a username and password combination.

Drag the HTTP Basic filter from the Authentication category of filters and drop it onto the policy editor of the Policy Studio. Complete the following fields:

Name:

Enter a name for the authentication filter here.

Realm:
The **Realm** entered here is presented to the client at the same time as they are entering their username and password. The client is then said to be logging into this realm. It is useful in cases where a given user might belong to many different realms, and so, by presenting the realm to the client, he can specify which realm he wants to log into.

Credential Format:
The username presented to the API Gateway during the HTTP Basic handshake can be of many formats, usually either username or distinguished name. Since the API Gateway has no way of inherently telling one format from the other (the client's username could be a DName), it is necessary to specify the format of the credential presented by the client.

Allow Client Challenge:
HTTP Basic Authentication can be configured to work in two ways:

1. **Direct Authentication:**
   The client sends up the `Authorization` HTTP Basic Authentication header in its first request to the server.
2. **Challenge-Response Handshake:**
   The client does *not* send the `Authorization` header when sending its request to the server (it does not know that the server requires HTTP Basic Authentication). The server responds with an `HTTP 401 response code`, instructing the client to authenticate to the server by sending up the `Authorization` header. The client then sends up a second request, this time including the `Authorization` header and the relevant username and password.

The first case is used mainly for machine-to-machine transactions in which there is no human intervention. The second case is typical of situations where a browser is talking to a Web Server. When the browser receives the HTTP 401 response to its initial request, it pops up a dialog to allow the user to enter the username and password combination.

If you wish to force clients to always send the HTTP Basic `Authorization` header to the API Gateway, disable the **Allow client challenge** checkbox. If, on the other hand, you wish to allow clients to engage in the HTTP Basic Authentication challenge-response handshake with the API Gateway, make sure this feature is enabled by checking this option.

Remove HTTP Authentication Header:
Select this checkbox to remove the HTTP `Authorization` header from the downstream message. If this option is left unchecked, the incoming `Authorization` header is forwarded onwards to the target system.

Repository Name:
The **Repository Name** field specifies the name of the **Authentication Repository** where all **User** profiles are stored. You can select a previously configured Tivoli repository by simply selecting the name of this repository from the **Repository Name** dropdown. Alternatively, a new repository can be added by clicking the **Add** button.

On the **Authentication Repository** dialog, select `Tivoli Repository` from the **Repository Type** field, and then enter a name for this type of store in the **Repository Name** field.

Select the **OK** button on the **Authentication Repository** dialog and then **Finish** button on the **HTTP Basic** filter to complete the configuration.

Connections to Tivoli authentication repositories can be configured globally by expanding the **Authentication Repository Profiles** node in the Policy Studio, right-clicking on the **Tivoli Repositories** node and selecting the **Add a New Repository** menu option. The globally configured repository is then available for selection in authentication filters, such as the HTTP Basic authentication filter, as described above.

Tivoli Attribute Retrieval

The **Tivoli Attribute Retrieval** filter can be used in cases where you would like to retrieve user attributes independently from authorizing the user against Tivoli Access Manager. This filter can be found in the **Attributes** category of filters. The following fields should be configured:

Name:

Enter a name for the filter in this field.

User ID:
Enter the ID of a user to retrieve attributes for. You can enter this as a static username, Distinguished Name (DName), or selector representing a message attribute. The selector is expanded at runtime to the value of the message attribute.

For example, you can enter `${authentication.subject.id}` in this field. This means that the ID of the authenticated user, which is normally a DName, is used to retrieve attributes for. For this to work correctly, an authentication filter must have been configured to run before this filter in the policy. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Attributes:
You can specify a list of user attributes to retrieve from the Tivoli server. Individual attributes to be retrieved can be added by clicking the **Add** button and entering the attributes in the dialog. If you want all attributes to be retrieved, simply leave the table blank.

Tivoli Configuration Files:
A Tivoli configuration file that contains all the required connection details is associated with a particular Oracle API Gateway instance. Click the **Settings** button to display the **Tivoli Configuration** dialog.

On the **Tivoli Configuration** dialog, select the API Gateway instance whose connection details you want to configure, then follow the steps as outlined above in the Tivoli Connections section.

Tivoli Authorization

Overview

Tivoli Access Manager provides authentication and access control services for Web resources. It also stores policies describing the access rights of users.

The API Gateway can integrate with this product through its Tivoli connector. The API Gateway Tivoli connector can query Tivoli for authorization information for a particular user on a given resource. In other words, the API Gateway asks Tivoli to make the authorization decision. If the user has been given authorization rights to the Web Service, the request is allowed through to the Service. Otherwise, the request is rejected.

For details on prerequisites for integration with IBM Tivoli, see the *Tivoli Integration* topic.

Adding a Tivoli Client

To add the machine running the API Gateway as a client of Tivoli, perform the following steps:

1. Open a terminal window on the machine running the Tivoli Authorization Server and Management Server.
2. Start the `pdadmin` tool using the following command, where `oracle` is the password for the Management Server:

   ```
   C:\WINNT> pdadmin -a sec_master -p oracle
   ```

   This starts the `pdadmin` terminal tool.
3. Use the `user create` command to add a user. The parameters are as follows:

   ```
   pdadmin> user create <username> <dn> <cn> <sn> <password>
   ```

   The following is an example where the API Gateway is running on a machine called `TEST_CLIENT` with an IP address of `192.168.0.100`:

   ```
   pdadmin> user create TEST_CLIENT cn=PdPermission/192.168.0.100,o=Company,c=ie \
   PdPermission/192.168.0.100 PdPermission myPass1234
   ```

   Make sure the DName you assign the user is *exactly* identical to the DName in your user's certificate. This includes case and attribute order. Also make sure that you put the IP address or hostname in the CN.
4. Next you must activate the account for the new user. Use the following command:

   ```
   pdadmin> user modify TEST_CLIENT account-valid yes
   ```

5. Finally, the user must be included in the remote Access Control List (ACL) client list:

   ```
   pdadmin> group modify remote-acl-users add batman
   ```

   The machine running the API Gateway has now been added as a client to Tivoli.

Adding Users and Web Services to Tivoli

To authorize a user to access a Web Service, you must first add the user to Tivoli as follows:

1. Add the user as before using the `user create` command as follows:

   ```
   pdadmin> user create <username> <dn> <cn> <sn> <password>
   ```

   Ensure that the DN you assign the user is identical to the DName in the user's certificate.
2. Next, you must insert the server that runs your Web Service into Tivoli's object space. Use the following command to

do this:

```
pdadmin> object create /API Gateway/<object-name> <description> 9
```

Note

The 9 parameter indicates that you are adding a Web Resource. In addition, it is the responsibility of the Policy Decision Point (API Gateway) to map an attempt to access a Web Service to a given object. The Tivoli Authorization server does not contain any mapping between its object space nodes and URLs.

3. Finally, you must create a binding between the user and the object by creating an ACL for the object, and adding the user to that list:

```
pdadmin> acl create <acl-name>
pdadmin> acl modify <acl-name> set user <username> rx
pdadmin> acl attach <object-name> <acl-name>
```

Configuring Tivoli Authorization

Open the **Tivoli Authorization** screen, and configure the following fields:

Name:
Enter a name for the Tivoli filter here.

Object Space:
The object space represents the resource for which the client must be authorized. Enter the name of the resources in the **Object Space** field. You can also enter selectors that represent the values of message attributes. At runtime, the API Gateway expands the selector to the current value of the corresponding message attribute.

Selectors have the following format:
`${message.attribute}`
For example, to specify the original path on which the request was received by the API Gateway as the resource, enter the following selector:
`${http.request.uri}`

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Access Method:
Clients can access a resource with a number of permissions such as read, write, execute and so on. A client is only authorized to access the requested resource if he has the relevant permissions checked in the **Access Types** listbox.

Tivoli Connection Settings:
You must enter details on how the API Gateway should connect to the Tivoli Access Manager in this section. The API Gateway must have been added to Tivoli as a user for it to connect to the Access Manager. Consult your Tivoli administrator for more information on how to do this.

Important

You must *never* allow more than one the API Gateway instance use the same account with the Tivoli server.

1. In the **Username** field, enter the username that the API Gateway uses to connect to the Tivoli server. This is the distinguished name of the API Gateway's X.509 certificate. You can use `%IP%` and `%HOSTNAME%` to generically repres-

ent the IP and hostname of the API Gateway instance. For example, the following entries are both valid:
`cn=PdPermission/%IP%, o=Company, c=ie`
`cn=PdPermission/%HOSTNAME%, o=Company, c=ie`
This means that multiple the API Gateway instances, each of which has been set up as a Tivoli user, can share this global setting. For example, one the API Gateway installation with `cn=10.10.10.10` and another with `cn=20.20.20.20`, can both be represented by `cn=PdPermission/%IP%` in the **Tivoli Username**. Similarly, an API Gateway instance with `cn=VS_1` and another with `cn=VS_2` can both be represented by `cn=PdPermission/%HOSTNAME%`.

2. In the **Security Master Password** field, enter the master password.
3. In the **Management Server** field, enter the IP address or hostname of the Tivoli Management Server.
4. In the **Authorization Server** field, enter the IP address or hostname of the Tivoli Authorization Server.

Tivoli Authentication Refresh

At start up, the API Gateway contacts the Tivoli server over SSL, and authenticates using the Security Master password. If you change the Security Master password in the **Policy Studio**, you must stop and start the API Gateway for this change to be picked up.

Retrieve Attributes from Tivoli

Overview

You can use the **Tivoli Attribute Retrieval** filter when you need to retrieve user attributes independently from authorizing the user against Tivoli Access Manager. This filter is found in the **Attributes** category of filters.

For details on prerequisites for integration with IBM Tivoli, see the *Tivoli Integration* topic.

Configuration

Complete the following fields to configure the **Retrieve Attributes from Tivoli** filter:

Name:
Enter an appropriate name for the filter.

User ID:
Enter the ID of a user to retrieve attributes for. You can enter a static user name, Distinguished Name (DName), or selector representing a message attribute. The selector is expanded to the value of the message attribute at runtime.

For example, you can enter `${authentication.subject.id}`. This means that the ID of the authenticated user, which is normally a DName, is used to retrieve attributes for. For this to work correctly, an authentication filter must have been configured to run before this filter in the policy. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Attributes:
You can specify a list of user attributes to retrieve from the Tivoli server. You can add individual attributes to be retrieved by clicking the **Add** button and entering the attributes in the dialog. If you want all attributes to be retrieved, leave the table blank.

Tivoli Configuration Files:
A Tivoli configuration file that contains all the required connection details is associated with a particular Oracle API Gateway instance. Click the **Settings** button to display the **Tivoli Configuration** dialog.

On the **Tivoli Configuration** dialog, select the API Gateway instance whose connection details you want to configure. For more details on configuring this wizard, see the *Tivoli Integration* topic.

CA SOA Security Manager Authorization

Overview

CA SOA Security Manager can authenticate end-users and authorize them to access protected Web resources. The API Gateway can interact directly with CA SOA Security Manager by asking it to make authorization decisions on behalf of end-users that have successfully authenticated to the API Gateway. CA SOA Security Manager decides whether to authorize the user, and relays the decision back to the API Gateway where the decision is enforced. The API Gateway, therefore, acts as a Policy Enforcement Point (PEP) in this situation, enforcing the authorization decisions made by the CA SOA Security Manager, which acts a Policy Decision Point (PDP).

⚠ **Important**

A CA SOA Security Manager authentication filter must be invoked before a CA SOA Security Manager authorization filter in a given policy. In other words, the end-user must authenticate to CA SOA Security Manager before they can be authorized for a protected resource.

Prerequisites

CA SOA Security Manager integration requires CA TransactionMinder SDK version 6.0 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Configuration

Configure the following fields on the **CA SOA Security Manager Authorization** filter:

Name:
Enter an appropriate name for the filter.

Attributes:
If the end-user is successfully authorized, the attributes listed here are looked up in CA SOA Security Manager, and returned to the API Gateway. These attributes are stored in the `attributes.lookup.list` message attribute. They can be retrieved at a later stage to generate a SAML attribute assertion.

Select the **Set attributes for SAML Attribute token** checkbox, and click the **Add** button to specify an attribute to fetch from CA SOA Security Manager.

SAML Authorization XML-Signature Verification

Overview

A SAML authorization assertion contains proof that a certain user has been authorized to access a specified resource. Typically such assertions are issued by a SAML PDP (Policy Decision Point) when a client requests access to a specified resource. The client must present identity information to the PDP, who makes sure that the client does indeed have permission to access the resource. The PDP then issues a SAML authorization assertion stating whether or not the client is allowed access the resource.

The PDP will usually *sign* the assertion as proof that only it could have signed the assertion, and also to guarantee the integrity of the assertion. It then inserts the assertion, together with its signature, into the message for consumption by a downstream Web Service.

When the API Gateway receives such a signed SAML authorization assertion, it can validate the signature on the assertion.

The following sample SOAP message contains a signed SAML authorization assertion:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
 <soap-env:Header xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <wsse:Security>
   <saml:Assertion
     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
     AssertionID="oracle-1056130475340"
     Id="oracle-1056130475340"
     IssueInstant="2006-06-20T17:34:35Z"
     Issuer="CN=Sample User,...........,C=IE"
     MajorVersion="1"
     MinorVersion="0">
    <saml:Conditions
      NotBefore="2006-06-20T16:20:10Z"
      NotOnOrAfter="2006-06-20T18:20:10Z"/>
    <saml:AuthorizationDecisionStatement
      Decision="Permit"
      Resource="http://www.oracle.com/service">
     <saml:Subject>
      <saml:NameIdentifier
        Format="urn:oasis:names:tc:SAML:1.0:assertion#X509SubjectName">
         sample
      </saml:NameIdentifier>
     </saml:Subject>
    </saml:AuthorizationDecisionStatement>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
      <dsig:SignedInfo>
        .....
      </dsig:SignedInfo>
      <dsig:SignatureValue>
        rpa/......0g==
      </dsig:SignatureValue>
      <dsig:KeyInfo>
        .....
      </dsig:KeyInfo>
    </dsig:Signature>
   </saml:Assertion>
  </wsse:Security>
 </soap-env:Header>
<soap-env:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
        </ns1:getTime>
  </soap-env:Body>
</soap-env:Envelope>
```

Configuration

Configure the following fields to validate the XML Signature over a SAML assertion:

SAML Signature:
Use this section to specify the location of the signature to validate. The signature can be selected using 3 options:

- ***Check signature inside the assertion:***
  Select this option if the signature will be present inside the SAML assertion itself.
- ***Check signature contained in WS-Security Block:***
  If the signature is contained within a WS-Security block (but outside the assertion), it is necessary to specify whether the signature covers only the assertion, or the assertion and the SOAP Body. Select the appropriate option depending on what the signature covers.
- ***Use advanced XPath:***
  If the signature is to be found in a non-standard location, an XPath expression can be used to identify it. Use the **Signature location XPath** to find a signature in a non-standard place.
  It is also necessary to specify the nodes that are signed by the signature. Use the **What must be signed XPath** to configure this.

Signer's Public Key/Certificate
Select the **Certificate in Message** radio button in order to use the certificate from the XML-Signature specified in the **SAML Signature** section. The certificate will be extracted from the `KeyInfo` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ....... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key must be retrieved from an LDAP directory of the API Gateway's Certificate Store.

For example, the following signed XML message does not include the signatory's certificate. Instead only the *Common Name* of the signatory's certificate is included. In this case, the API Gateway must obtain the certificate from either an LDAP directory or the Certificate Store in order to validate the signature on the assertion.

```
<?xml version="1.0" encoding="UTF-8"?>
 <soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
    <dsig:SignedInfo>
     <dsig:CanonicalizationMethod
                   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     <dsig:SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference URI="">
      <dsig:Transforms>
       <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <dsig:XPath>ancestor-or-self::soap-env:Body</dsig:XPath>
       </dsig:Transform>
       <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
      </dsig:Transforms>
      <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <dsig:DigestValue>rvJMkZ1RDo3pNfqCUBa4Qhs8i+M=</dsig:DigestValue>
     </dsig:Reference>
    </dsig:SignedInfo>
    <dsig:SignatureValue>
      AXL2gKhqqKwcKujVPftVoztySvtCdARGf97Cjt6Bbpf0w8QFiNuLJncQVnKB
      cQ+91KvudYZ/Sk8u7tXhoEiLvNwg76B2STPh+ypEWO+J7OSPedlUdnfVRRvW
      vjYLwJVjGNZ+mMTxvfO1wwcIb2Hg94n1BOaeBrNJ+2uO4i87W5TyufAGI+V8
      S6oSpPc5KQeHLXoyHS2+fXyqReSiwdhOeli4D4xT+HbjRgYJIwIikXn2k1Fr
      D/hnd1/xVf/LjrOwoY9id8W3IcZAzMIRh5SBZjWHYOQzk79xy4YDpzNVYIOB
      laAFqzg9G+Z4VYj+RdgrIVHhOXt+mq+fGZV6VheWGQ==
    </dsig:SignatureValue>
    <dsig:KeyInfo>
     <dsig:KeyName>
       CN=User,OU=R&D,O=Company Ltd.,L=Dublin 4,ST=Dublin,C=IE
     </dsig:KeyName>
    </dsig:KeyInfo>
   </dsig:Signature>
  </soap-env:Header>
 <soap-env:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </soap-env:Body>
</soap-env:Envelope>
```

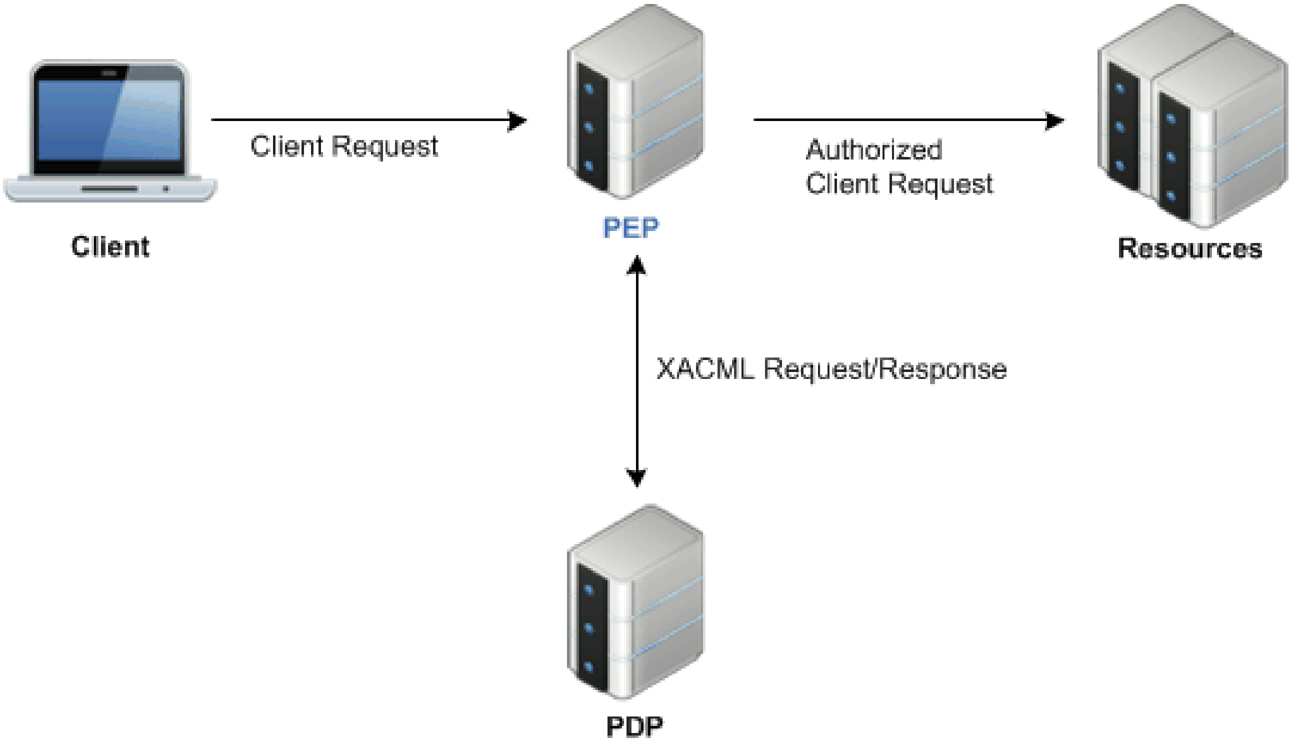
To retrieve a client certificate from an LDAP directory, select a pre-configured one from the **LDAP Source** dropdown, or add/edit a new/existing LDAP directory by clicking the **Add/Edit** button.

Alternatively, select a certificate from the Trusted Certificate Store by selecting the **Certificate in Store** radio button and clicking on the **Select** button. This certificate will then be associated with the incoming message so that all subsequent certificate-based filters will use this user's certificate.

XACML Policy Enforcement Point

Overview

The eXtensible Access Control Markup Language (XACML) Policy Enforcement Point (PEP) filter enables you to configure the API Gateway to act as a PEP. The API Gateway intercepts a user request to a resource, and enforces the decision from the Policy Decision Point (PDP). The API Gateway queries the PDP to see if the user has access to the resource, and depending on the PDP response, allows the filter to pass or fail. Possible PDP responses include `Permit`, `Deny`, `NotApplicable`, and `Indeterminate`.



Workflow
In more detail, when the **XACML PEP** filter is configured in the API Gateway, the workflow is as follows:

1. The client sends a request for the resource to the XACML PEP filter.
2. The PEP filter stores the original client request, and generates the XACML request.
3. The PEP filter delegates message-level security to the polices configured on the **XACML** tab.
4. The PEP filter routes the XACML request to the PDP using details configured on the **Routing** tab.
5. The PDP decides if access should be granted, and sends the XACML response back to the API Gateway.
6. The PEP filter validates the response from the PDP.
7. By default, if the response is `Permit`, the PEP filter passes, and the original client request for the resource is authorized, and the policy flow continues on the success path.

Further Information
For more details on XACML, see the XACML specification:
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

Example XACML Request

The following example XACML request is used to illustrate the XACML request configuration settings explained in this topic:

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
       <AttributeValue>admin</AttributeValue>
     </Attribute>
     <Attribute AttributeId=”department" DataType="http://www.w3.org/2001/
        XMLSchema#string">
       <AttributeValue>sysadmin</AttributeValue>
     </Attribute>
  </Subject>
  <Resource>
     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
       <AttributeValue>http://localhost:8280/services/echo/echoString</AttributeValue>
     </Attribute>
  </Resource>
  <Action>
     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                  DataType="http://www.w3.org/2001/XMLSchema#string">
     <AttributeValue>read</AttributeValue>
     </Attribute>
  </Action>
  <Environment/>
</Request>
```

General Settings

In the **XACML PEP** filter screen, configure the following general field:

Name:
Enter an appropriate name for this filter.

XACML Settings

The **XACML** tab specifies configuration settings for the generated XACML request. Configure the following fields on this tab:

XACML Version:
Select the XACML version from the list. Defaults to `XACML2_0`.

Create XACML Request Assertion with the following attributes:
Click the **Add** button on the following tabs to add attributes to the XACML request:

Subject	Represents the entity making the access request (who wants access to the resource). The `Subject` element can contain multiple `Attribute` elements, which are used to identify the `Subject`. Each `Attribute` element has two attributes: `AttributeId` and `DataType`. You can define your own `AttributeId` or use those provided by the XACML specification. For more details on adding attributes, see the next subsection.
Resource	Defines the data, service, or system component that the `Subject` wants to access. The `Resource` element con-

	tains one or more attributes of the resource to which subjects request access. There can be only one `Resource` element per XACML request. A specific `Resource` is identified by the `Attribute` child element. In the Example XACML Request, the `Subject` wants to access the following `Resource`: `http://localhost:8280/services/echo/echoString`.
Action	Contains one or more attributes of the action that subjects wish to perform on the resource. There can be only one `Action` element per XACML request. A specific `Action` is identified by the `Attribute` child element. In the Example XACML Request, the `Subject` wants read access the following `Resource`: `http://localhost:8280/services/echo/echoString`.
Environment	A more complex request context may contain some attributes not associated with the `Subject`, `Resource`, or `Action`. These are placed in an optional `Environment` element after the `Action` element.

Adding Attributes

When you click the **Add** button on each tab, the **XACML** dialog is displayed to enable you to add attributes. Complete the following fields on this dialog:

Attribute ID	Enter a custom `AttributeId` or select one provided by the XACML specification from the list. For example, the XACML special identifiers defined for the `Subject` include the following:
	`urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name`
	`urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address`
	`urn:oasis:names:tc:xacml:1.0:subject:authentication-method`
	`urn:oasis:names:tc:xacml:1.0:subject:authentication-time`
	`urn:oasis:names:tc:xacml:1.0:subject:key-info`
	`urn:oasis:names:tc:xacml:1.0:subject:request-time`
	`urn:oasis:names:tc:xacml:1.0:subject:session-start-time`
	`urn:oasis:names:tc:xacml:1.0:subject:subject-id`
	`...`
	In the Example XACML Request, the first attribute under the `Subject` element uses the `urn:oasis:names:tc:xacml:1.0:subject:subject-id` identifier. The next is a custom `department` attribute. This can be any custom attribute (for example, `mail`,

	givenName, or accessList), which can be identified by the XACML policy defined where this request is evaluated.
Value(s)	Click the **Add** button to add an attribute value. Enter the value in the **Add** dialog, and click **OK**. You can add multiple values for a single attribute.
Type	Select the type of data that the AttributeValue element should contain from the list. For example, the set of data types defined in XACML includes the following: http://www.w3.org/2001/XMLSchema#string http://www.w3.org/2001/XMLSchema#boolean http://www.w3.org/2001/XMLSchema#integer http://www.w3.org/2001/XMLSchema#double http://www.w3.org/2001/XMLSchema#time http://www.w3.org/2001/XMLSchema#date http://www.w3.org/2001/XMLSchema#dateTime http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration http://www.w3.org/2001/XMLSchema#anyURI http://www.w3.org/2001/XMLSchema#hexBinary ... In the Example XACML Request, the Attributes are of type http://www.w3.org/2001/XMLSchema#string.
Issuer	Specify an optional issuer for the attribute. For example, this may be a Distinguished Name, or some other identifier agreed with the issuer.

AuthzDecisionQuery Settings

This section enables you to configure settings for the Authorization Decision Query, which is sent in the XACML request to the PDP. Complete the following fields in this group:

Decision based on external XACML attributes	If this is selected, the authorization decision must be made based only on the information contained in the XACML Authz Decision Query, and external XACML attributes must not be used. If this is unselected, the authorization decision can be made based on XACML attributes not contained in the XACML Authz Decision Query. This is unselected by default, which is equivalent to the following setting in the XACML Authz Decision Query: <InputContextOnly value="false">
Return Context	If this is selected, the PDP must include an xacmlcontext:Request instance in the XACMLAuthzDecision statement in the XACMLAuthzDecision response. The xacmlcontext:Request instance must include all attributes supplied by the PEP in the xacml-samlp:XACMLAuthzDecisionQuery used to make the authorization decision. If this is unselected, the PDP must not include an xacmlcontext:Request instance in the XACMLAuthzDecision statement in the XACMLAuthzDecision response. This is unselected by default, which is equivalent to the following setting in the XACML request:

	`<ReturnContext value="false">`
Combine Policies	If this is selected, the PDP must insert all policies passed in the `xacmlsamlp:XACMLAuthzDecisionQuery` into the set of policies or policy sets that define the PDP. If this is unselected, there must be no more than one `xacml:Policy` or `xacml:PolicySet` passed in the `xacml-samlp:XACMLAuthzDecisionQuery`. This is selected by default, which is equivalent to the following setting in the XACML request: `<CombinePolicies value="true">`

XACML Message Security
This section enables you to delegate message-level security to the configured custom security polices. Complete the following fields in this group:

XACML Request Security	Click the browse button on the right, select a policy in the **XACML request security policy** dialog, and click **OK**.
XACML Response Security	Click the browse button on the right, select a policy in the **XACML response security policy** dialog, and click **OK**.

XACML Response:
Select the **Required response decision** from the PDP that is required for this **XACML PEP** filter to pass. Defaults to `Permit`. Possible values are as follows:

- `Permit`
- `Deny`
- `Indeterminate`
- `NotApplicable`

Routing Settings

The **Routing** tab enables you to specify configuration settings for routing the XACML request to the PDP. You can specify a direct connection to the PDP using a URL. Alternatively, if the routing behavior is more complex, you can delegate to a custom routing policy, which takes care of the added complexity.

Use the following URL:
If you wish to route XACML requests to a URL, select this option, and enter the **URL**. You can also specify the URL as a selector so that the URL is built dynamically at runtime from the specified message attributes. For example, `${host}:${port}`, or `${http.destination.protocol}://${http.destination.host}:${http.destination.port}`.

In both cases, you can configure the connection details (for example, SSL and other authentication schemes) using the fields in the **Connection Details** group below. For more details, see the *Connect to URL* topic. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Delegate routing to the following policy:
If you wish to use a dedicated routing policy to send XACML requests to the PDP, select this option. Click the browse button next to the **Routing Policy** field. Select the policy that you want to use to route XACML requests, and click **OK**.

Advanced Settings

Configure the following settings on the **Advanced** tab:

SOAP Settings:
The available SOAP settings are as follows:

SOAP version required	Specifies the SOAP version required when creating the XACML request message. The available options are as follows: • `SOAP1_1` • `SOAP1_2` • `NONE` Defaults to `SOAP1_1`.
SOAP Operation	Specifies the SOAP operation name used in the XACML request message. Defaults to `XACMLAuthzDecisionQuery`.
Prefix	Specifies the prefix name used in the XACML request message. Defaults to `xacml-samlp`.
Namespace	Specifies the namespace used in the XACML request message. Defaults to `urn:oasis:xacml:2.0:saml:protocol:schema:os`.
SOAP Action	You can specify an optional `SOAPAction` field used in the XACML request header to indicate the intent of the request message.

Advanced Settings:
The available advanced settings are as follows:

Store and restore original message	Specifies whether to store the original client request before generating the XACML request, and then to restore the original client request after access is granted. This option is selected by default.
Split subject attributes into individual elements	Specifies whether to split `Subject` attributes into individual elements in the XACML request. This option is not selected by default.
Split resource attributes into individual elements	Specifies whether to split `Resource` attributes into individual elements in the XACML request. This option is not selected by default.

SiteMinder Certificate Authentication

Overview

CA SiteMinder can authenticate end-users and authorize them to access protected Web resources. When the API Gateway retrieves an X.509 certificate from a message or during an SSL handshake, it can authenticate to SiteMinder on behalf of the user using the certificate. SiteMinder decides whether the user should be authenticated, and the API Gateway then enforces this decision.

Prerequisites

CA SiteMinder integration requires CA SiteMinder SDK version 12.0-sp1-cr005 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the `*InstallDir*/ext/lib` directory.
   - Add `.dll` files to the `*InstallDir*\Win32\lib` directory.
   - Add `.so` files to the `*InstallDir*/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the `*InstallDir*/plugins/thirdparty.runtime.dependencies_11.1.2.1.0` directory.
2. Restart the Policy Studio.

Configuration

Configure the following fields:

Name:
Enter an appropriate name for the filter.

Agent Name:
Click the button on the right to select a previously configured agent to connect to SiteMinder. This name *must* correspond with the name of an agent previously configured in the SiteMinder **Policy Server**. At runtime, the API Gateway connects as this agent to a running instance of SiteMinder.

To add an agent, right-click the **SiteMinder/SOA Security Manager Connections** tree node, and select **Add a SiteMinder Connection**. Alternatively, you can add SiteMinder connections under the **External Connections** node in the Policy Studio tree view. For details on how to configure a SiteMinder connection, see the *SiteMinder/SOA Security Manager Connection* topic.

Resource:
Enter the name of the protected resource for which the end-user must be authenticated. You can enter a selector representing a message attribute, which is expanded to a value at runtime. Message attribute selectors have the following format:

```
${message.attribute}
```

For example, to specify the original path on which the request was received by the API Gateway as the resource, enter the following selector:

```
${http.request.uri}
```

Action:
The end-user must be authenticated for a specific action on the protected resource. By default, this action is taken from the HTTP verb used in the incoming request. You can use the following selector to get the HTTP verb:

```
${http.request.verb}
```

Alternatively, any user-specified value can be entered. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Single Sign-On Token:
When a client has been authenticated for a given resource, SiteMinder can generate a *single sign-on token* and return it to the client. The client can then pass this token with future requests to the API Gateway. When the API Gateway receives such a request, it can validate the token using the **SiteMinder Session Validation** filter to authenticate the client. In other words, the client is authenticated for the entire lifetime of the token. As long as the token is still valid, the API Gateway does not need to authenticate the client against SiteMinder for every request, which increases throughput considerably.

In this section, you can instruct SiteMinder to generate a single sign-on token. The API Gateway can then store this token in a user-specified message attribute. By default, the token is stored in the `siteminder.session` message attribute.

Typically, the token is copied to the `attribute.lookup.list` message attribute using the **Copy / Modify Attributes** filter, before being inserted into a SAML attribute statement using the **Insert SAML Attribute Assertion** filter. The attribute statement is then returned to the client for use in subsequent requests.

Select the **Create single sign-on token** checkbox to instruct SiteMinder to generate the single sign-on token. Enter the name of the message attribute where the token is stored in the field provided.

SiteMinder Session Validation

Overview

CA SiteMinder can authenticate end-users and authorize them to access protected Web resources. When the API Gateway has authenticated successfully to SiteMinder on behalf of a user using the *SiteMinder Certificate Authentication* filter, SiteMinder can issue a *single sign-on* token and return it to the API Gateway. Typically, the API Gateway inserts this token into a SAML attribute assertion or an HTTP Header, and returns it to the client.

The client then sends the single-sign on token in subsequent requests to the API Gateway. The API Gateway extracts the single-sign on token from the message payload or HTTP headers, and stores it in a message attribute, usually the `siteminder.session` attribute.

The API Gateway can then use the **SiteMinder Session Validation** filter to ensure that the token is still valid, and hence, that the user is still authenticated. This means that the API Gateway does not have to authenticate every request to SiteMinder. By validating the token, the user can be authenticated, and therefore, unnecessary round-trips to SiteMinder can be avoided.

Prerequisites

CA SiteMinder integration requires CA SiteMinder SDK version 12.0-sp1-cr005 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the `*InstallDir*/ext/lib` directory.
   - Add `.dll` files to the `*InstallDir*\Win32\lib` directory.
   - Add `.so` files to the `*InstallDir*/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the `*InstallDir*/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Configuration

Configure the following fields on the **SiteMinder Session Validation** screen:

Name:
Enter an appropriate name for the filter.

Agent Name:
Click the button on the right to select a previously configured agent to connect to SiteMinder. This name *must* correspond with the name of an agent previously configured in the SiteMinder **Policy Server**. At runtime, the API Gateway connects as this agent to a running instance of SiteMinder.

To add an agent, right-click the **SiteMinder/SOA Security Manager Connections** tree node, and select **Add a SiteMinder Connection**. Alternatively, you can add SiteMinder connections under the **External Connections** node in the Policy Studio tree view. For details on how to configure a SiteMinder connection, see the *SiteMinder/SOA Security Manager Connection* topic.

Resource:
Enter the name of the protected resource for which the end-user must be authenticated. You can enter a selector representing a message attribute, which is expanded to a value a runtime. Message attribute selectors have the following format:

```
${message.attribute}
```

For example, to specify the original path on which the request is received by the API Gateway as the resource, enter the following selector:

```
${http.request.uri}
```

Action:
The end-user must be authenticated for a specific action on the protected resource. By default, this action is taken from the HTTP verb used in the incoming request. You can use the following selector to get the HTTP verb:

```
${http.request.verb}
```

Alternatively, any user-specified value can be entered here. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Message attribute containing session:
Enter the name of the message attribute that contains the single sign-on token generated by SiteMinder. By default, the token is stored in the `siteminder.session` message attribute, but can be stored in any attribute.

SiteMinder Logout

Overview

When the API Gateway authenticates to CA SiteMinder on behalf of a user, SiteMinder can issue a *single sign-on* token as evidence of the authentication event. The token is eventually returned to the client, which can then use it in subsequent requests to the API Gateway.

Instead of authenticating the client against SiteMinder for every request, the API Gateway need only validate the token. If the token validates, the client can be considered authenticated. If the token does not validate, the client is not considered authenticated.

You can use the **SiteMinder Logout** filter to invalidate a single sign-on token that was previously issued by SiteMinder. When the token has been invalidated, the client is no longer be considered authenticated.

Note

You must have already validated the session before calling the **SiteMinder Logout** filter in your policy. For more details, see the *SiteMinder Session Validation* topic.

Prerequisites

CA SiteMinder integration requires CA SiteMinder SDK version 12.0-sp1-cr005 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_11.1.2.1.0` directory.
2. Restart the Policy Studio.

Configuration

Enter a name for the filter in the **Name** field of the **SiteMinder Logout** screen.

SiteMinder Authorization

Overview

CA SiteMinder can authenticate end-users and authorize them to access protected Web resources. The API Gateway can interact directly with SiteMinder by asking it to make authorization decisions on behalf of end-users that have successfully authenticated to API Gateway. The API Gateway then enforces the decisions made by SiteMinder.

Important

A SiteMinder authentication filter must be configured before a SiteMinder authorization filter is created. In other words, end-users must authenticate to SiteMinder before they can be authorized.

Prerequisites

CA SiteMinder integration requires CA SiteMinder SDK version 12.0-sp1-cr005 or later.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Configuration

Configure the following fields on the **SiteMinder Authorization** filter:

Name:
Enter an appropriate name for the filter.

Attributes:
If the end-user is successfully authorized, the attributes listed here are returned to the API Gateway and stored in the `attribute.lookup.list` message attribute. They can then be used by subsequent filters in a policy to make decisions on their values. Alternatively, they can be inserted into a SAML attribute assertion so that the target service can apply some business logic based on their values (for example, if role is CEO, escalate the request, and so on).

Select the **Retrieve attributes from CA SiteMinder** checkbox, and click the **Add** button to specify an attribute to fetch from SiteMinder. If you select the **Retrieve attributes from CA SiteMinder** checkbox, and do not specify attribute names to be retrieved, all attributes returned by SiteMinder are added to the `attribute.lookup.list` message attribute.

SiteMinder/SOA Security Manager Connection

Overview

This topic explains how to create connections to CA SiteMinder and CA SOA Security Manager. Under the **External Connections** tree node in the Policy Studio, right-click the **SiteMinder/SOA Security Manager Connection** node, and select **Add CA SiteMinder Connection** or **Add CA SOA Security Manager Connection**.

You can specify how the API Gateway connects to CA SiteMinder using the **SiteMinder Connection Details** dialog. You can specify how the API Gateway connects to CA SOA Security Manager using the **CA SOA Security Manager Connection Details** dialog. In both cases, the API Gateway must have already been set up as an agent in the CA **Policy Server**.

The connection details to be configured for the API Gateway are the same for both SiteMinder and SOA Security Manager, with an additional setting for SOA Security Manager.

SiteMinder and SOA Security Manager Connection Details

This section describes details that are common to both SiteMinder and CA SOA Security Manager connections.

Agent Name:
Enter the name of the agent to connect to SiteMinder or SOA Security Manager in the **Agent Name** field. This name *must* correspond to the name of an agent previously configured in the CA **Policy Server**.

Agent Configuration Object:
The name entered must match the name of the Agent Configuration Object (ACO) configured in the CA **Policy Server**. The API Gateway currently does not support any features represented by the ACO parameters except for the `PersistentIPCheck` setting. For example, the API Gateway ignores the `DefaultAgent` parameter, and uses the agent value it collects separately during agent registration.

When the `PersistentIPCheck` ACO parameter is set to `yes`, this instructs the API Gateway to compare the IP address from the last request (stored in a persistent cookie) with the IP address in the current request to see if they match. If the IP addresses do not match, the API Gateway rejects the request. If this parameter is set to `no`, this check is disabled.

SmHost.conf file created by smreghost:
The API Gateway host machine must be registered with SiteMinder or SOA Security Manager. To register the host machine, you must use the `smreghost` tool on the API Gateway machine. The `smreghost` tool creates a file called `SmHost.conf`. You must then use the **Browse** button to upload this file into the API Gateway configuration.

If you have already generated a suitable `SmHost.conf` file, and copied it to the machine on which you are running the Policy Studio, you can browse to the location of the file using the **Browse** button at the bottom right of the text area. You can select whether to use an `SmHost.conf` or `SmHost.cnf` file in the dialog. You can also enter the file name as an environment variable selector (for example, `${env.SMHOST}`). For more details, see *Deploying the API Gateway in Multiple Environments*. After selecting the configuration file, the connection details are displayed in the text area.

If you do not have a suitable `SmHost.conf` file, you can generate one by running the `smreghost` command on the machine running the API Gateway. Complete the following steps:

1. You need to run the `smreghost` command on the machine on which you have installed the API Gateway. The `smreghost` tool is found in the following location, depending on your target platform:
   Windows: `/Win32/lib`
   Linux: `/Linux.i386/bin`
   Solaris: `/SunOS.sun4u-32/bin`

   Open a command prompt at this directory, and run the `smreghost` command. You must pass the appropriate command-line arguments, depending on the `hostname` and `hostconfigobject` configured to represent the API Gateway in the CA **Policy Server**. Similarly, you must specify the hostname/IP and port of the CA Policy Server.

2. The `smreghost` tool writes its output to a `SmHosts.conf` file in the same directory. You must manually copy this file from the machine running the API Gateway to the machine running the Policy Studio.
3. Browse to the location of this file using the **Browse** button on the connection details dialog.

SOA Security Manager Connection Details Only

This section describes details that are specific to CA SOA Security Manager connections only. In addition to the fields already described in the previous section, you must also configure the following field on the **CA SOA Security Manager Connection Details** dialog.

XMLSDKAcceptSMSessionCookie:
This setting controls whether the CA SOA Security Manager authentication filter accepts a single sign-on token for authentication purposes. The single sign-on token must reside in the HTTP header field named `SMSESSION` to authenticate using this mechanism. This token is created and updated when the CA SOA Security Manager authorization filter runs successfully.

When this checkbox is selected, the authentication filter allows authentication using a single sign-on token.

Note

If no single sign-on token is present in the message, the authentication filter authenticates fully by gathering credentials from the request in whatever manner has been configured in the CA SOA Security Manager. When this checkbox is unselected, the authentication filter authenticates fully (it never allows authentication using a single sign-on token).

Static CRL Certificate Validation

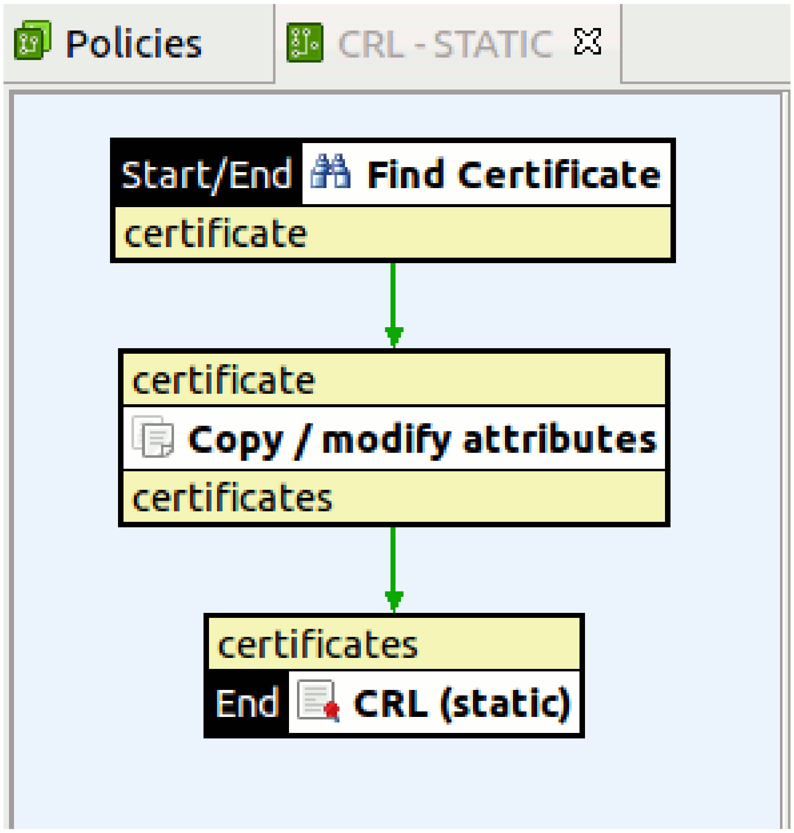
Overview

A Certificate Authority (CA) may wish to publish a Certificate Revocation List (CRL) to a file. In such cases, the API Gateway can load the revoked certificates from the file-based CRL and validate user certificates against it.

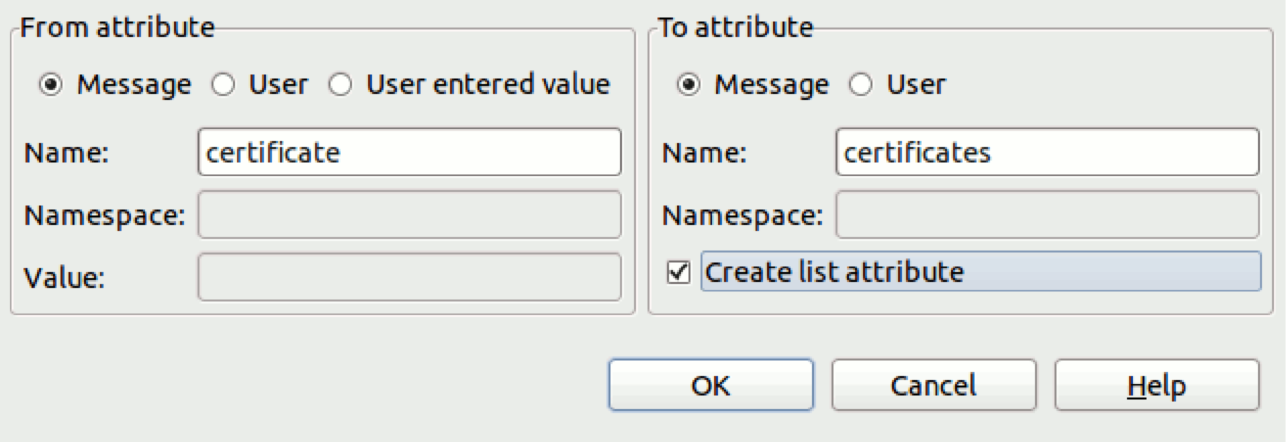
Because the CRL is typically signed by the CA that owns it, the certificate of the CA that issued the CRL *must* be imported into the **Certificate Store** before this filter can work correctly. In addition, the **CRL (Static)** filter requires the `certificates` message attribute to be set by a preceding filter.

Example Policy
Typically, a **Find Certificate** filter is first used to find the certificate, which is stored in a `certificate` message attribute. You can then use a **Copy / Modify Attributes** filter to copy the `certificate` attribute to the `certificates` attribute by selecting its **Create list attribute** setting.

The following example policy shows the filters used:



The following example shows the settings used in the **Copy / Modify Attributes** filter:



From attribute

● Message ○ User ○ User entered value

Name: certificate

Namespace:

Value:

To attribute

● Message ○ User

Name: certificates

Namespace:

☑ Create list attribute

Important

Typically, a CA publishes a new CRL, containing the most up-to-date list of revoked certificates at regular intervals. However, the **CRL (Static)** filter does not automatically update the CRL when it is loaded from a local file. If you need to automatically retrieve updated CRLs from a particular URL, you should use the **CRL (Dynamic)** filter.

Configuration

Enter a name for the filter in the **Name** field, and click the **Load CRL** button to browse to the location of the CRL file. When the CRL has been loaded from the selected location, read-only information regarding revoked certificates and update dates is displayed in the other fields on the screen.

Dynamic CRL Certificate Validation

Overview

This filter is responsible for validating certificates against a Certificate Revocation List (CRL) that has been published by a Certificate Authority (CA). The CRL is retrieved from the specified URL and is cached by the server for certificate validation. The filter automatically fetches a potentially updated CRL from this URL when the criteria specified in the **Automatic CRL Update Preferences** section are met.

Configuration

Configure the following fields on the **CRL (Dynamic)** screen:

Name:
Enter an appropriate name for the filter.

CRL Import URL:
Enter the full URL of the CRL to use to validate the certificate. Alternatively, you can browse to the location of the CRL by clicking the button.

Automatic CRL Update Preferences:
Typically, a CA publishes an updated CRL at regular intervals. You can configure the filter to dynamically pull down the latest CRL published by the CA at specified intervals. Select the appropriate update option from the following:

- **Do not update:**
  The filter never attempts to automatically retrieve the latest CRL.
- **Update on "next update" date:**
  The CRL published by the CA contains a *Next Update* date, which indicates the next date on which the CA publishes the CRL. You can choose to dynamically retrieve the updated CRL on the Next Update date by selecting this option. This effectively synchronizes the server with the CA updates.
- **Update every number of days:**
  The filter retrieves the CRL every number of days specified.
- **Trigger update on cron expression:**
  You can enter a cron expression to determine when to perform the automatic update.

CRL LDAP Validation

Overview

A Certificate Revocation List (CRL) is a signed list indicating a set of certificates that are no longer considered valid (revoked certificates) by the certificate issuer. The API Gateway can query a CRL to find out if a given certificate has been revoked. If the certificate is present in the CRL, it should not be trusted.

To validate a certificate using a CRL lookup, the certificate's issuing CA certificate should be trusted by the API Gateway. This is because for a CRL lookup, the CA public key is needed to verify the signature on the CRL. The issuing CA public key is not always included in the certificates that it issues, so it is necessary to retrieve it from the API Gateway's certificate store instead.

Configuration

The **Name** and **URL** of all currently configured LDAP directories are displayed in the table on the **CRL Certificate Validation** screen. The API Gateway checks the CRL of all selected LDAP directories to validate the client certificate. The filter fails as soon as the API Gateway determines that one of the CRLs has revoked the certificate.

To configure LDAP connection information, complete the following fields:

Name:
Enter an appropriate name for the filter.

LDAP Connection:
Click the button on the right, and select the LDAP directory to check its CRL. If you wish to use an existing LDAP directory, (for example, `Sample Active Directory Connection`), you can select it in the tree. To add an LDAP directory, right-click the **LDAP Connections** tree node, and select **Add an LDAP Connection**.

Alternatively, you can add LDAP connections under the **External Connections** node in the Policy Studio tree view. For more details on how to configure LDAP connections, see the topic on *Configuring LDAP Directories*.

CRL Responder

Overview

This filter enables the API Gateway to behave as Certificate Revocation List (CRL) responder, which returns CRLs to clients. This filter imports the CRL from a specified URL. You can also configure it to periodically retrieve the CRL from this URL to ensure that it always has the latest version.

Configuration

Configure the following fields on the **CRL Responder** screen:

Name:
Enter an appropriate name for the filter.

CRL Import URL:
Enter the full URL of the CRL that you want to return to clients. Alternatively, browse to the location of the CRL file by clicking the browse button on the right.

Automatic CRL Update Preferences:
Because keeping up-to-date with the latest list of revoked certificates is crucial in any *trust network*, it is important that you configure the filter to retrieve the latest version of the CRL on a regular basis. The following automatic update options are available:

- **Do not update:**
  The CRL is not automatically updated.
- **Update on "next update" date:**
  The CRL published by the CA contains a *Next Update* date, which indicates the next date on which the CA publishes the CRL. You can choose to dynamically retrieve the updated CRL on the Next Update date by selecting this option. This effectively synchronizes the server with the CA updates.
- **Update every number of days:**
  The CRL is updated after the specified number of days has elapsed (for example, every 3 days).
- **Trigger update on cron expression:**
  You can enter a cron expression to determine when to perform the automatic update.

Create Thumbprint from Certificate

Overview

The **Create Thumbprint** filter can be used to create a human-readable thumbprint (or fingerprint) from the X.509 certificate that is stored in the `certificate` message attribute. The generated thumbprint is stored in the `certificate.thumbprint` attribute.

Configuration

Configure the following fields on this filter:

Name:
Enter a name for this filter.

Digest Algorithm:
Select the digest algorithm to create the thumbprint of the certificate from the drop-down list.

Certificate Validity

Overview

The validity period of an X.509 certificate is encoded in the certificate. The **Certificate Validity** performs a simple check on a certificate to ensure that it has not expired.

By default, the **Certificate Validity** filter searches for the X.509 certificate in the `certificate` message attribute, which must be set by a predecessor filter in the policy (for example, by an **SSL Authentication** filter).

Configuration

Configure the following fields on the **Certificate Validity** screen:

Name:
Enter an appropriate name for the filter.

Certificate Selector Expression:
Enter the selector expression that specifies where to obtain the certificate (for example, from a message attribute). The filter checks the validity of the specified certificate. If no certificate is found, the filter returns an error. Defaults to `${certificate}`.

Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Find Certificate

Overview

The **Find Certificate** filter locates a certificate and sets it in the message for use by other certificate-based filters. Certificates can be extracted from the **User Store**, message attributes, HTTP headers, or attachments.

Configuration

By default, the API Gateway stores the extracted certificate in the `certificate` message attribute. However, it can store the certificate in any message attribute, including any arbitrary attribute specified by the user (for example, a `user_certificate` attribute). The certificate can be extracted from this attribute by a successor filter in the policy.

Name:
Enter an appropriate name for the filter in the **Name** field.

Attribute Name:
Enter or select the name of the message attribute to store the extracted certificate in.

When the target message attribute has been selected, the next step is to specify the location of the certificate from one of the following options:

User:
Select a **User** whose certificate is extracted from the **Certificate Store** and set to the message.

Certificate Store:
Click the **Select** button, and select a certificate from the Certificate Store.

User or Wildcard:
This field represents an alternative way to specify what user's certificate is used. Either an explicitly named **User's** certificate is used, or you can specify a selector to locate a **User** name or DName, which can then be used to locate the certificate.

You can specify a selector by enclosing the message attribute that contains the user name or DName in curly brackets, and prefixing this with $. For example:

```
${authentication.subject.id}
```

This selector means that the API Gateway uses the certificate belonging to the subject of the authentication event in subsequent certificate-related filters. The certificate is set to the `certificate` message attribute. Using selectors is a more flexible way of locating certificates than specifying the **User** directly. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Message Attribute Name:
Enter the name of the message attribute that contains the certificate.

HTTP Header Name:
Enter the name of the HTTP header that contains the certificate.

Attachment Name:
Enter the name of the attachment (`Content-Id`) that contains the certificate. Alternatively, you can enter a selector in this field to represent the value of a message attribute.

Alias Name or Wildcard:
Enter the alias name of the certificate. Alternatively, you can enter a selector to represent the value of a message attribute. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Extract Certificate Attributes

Overview

You can use the **Extract Certificate Attributes** filter to extract the X.509 attributes from a certificate stored in a specified Oracle message attribute.

Typically, this filter is used in conjunction with the **Find Certificate** filter, which is found in the **Certificates** category of message filters. In this case, the **Find Certificate** filter can locate a certificate from one of many possible sources (for example, the message itself, an HTTP header, or the API Gateway Certificate Store), and store it in a message attribute, which is usually the `certificate` attribute.

The **Extract Certificate Attributes** filter can then retrieve this certificate and extract the X.509 attributes from it. For example, you can then use a **Validate Message Attribute** filter to check the values of the attributes.

Generated Message Attributes

The **Extract Certificate Attributes** filter extracts the X.509 certificate attributes and populates a number of Oracle message attributes with their respective values. The following table lists the message attributes that are generated by this filter, and shows what each of these attributes contains after the filter has executed:

Generated Message Attribute	***Contains***
`attribute.lookup.list`	This user attribute list contains an attribute for each Distinguished Name (DName) attribute for the subject (`cn`, `o`, `l`, and so on). The user attributes are named `cn`, `o`, and so on.
`attribute.subject.id`	The DName of the subject of the cert.
`attribute.subject.format`	Set to `X509DName`.
`cert.basic.constraints`	If the subject is a Certificate Authority (CA), and the `BasicConstraints` extension exists, this field gives the maximum number of CA certificates that may follow this certificate in a certification path. A value of zero indicates that only an end-entity certificate may follow in the path. This contains the value of `pathLenConstraint` if the `BasicConstraints` extension is present in the certificate and the subject of the certificate is a CA, otherwise its value is -1. If the subject of the certificate is a CA and `pathLenConstraint` does not appear, there is no limit to the allowed length of the certification path.
`cert.extended.key.usage`	A String representing the OBJECT IDENTIFIERs of the `ExtKeyUsageSyntax` field of the extended key usage extension (`OID = 2.5.29.37`). It indicates a purpose for which the certified public key may be used, in addition to, or instead of, the basic purposes indicated in the key usage extension field.
`cert.hash.md5`	An MD5 hash of the certificate.
`cert.hash.sha1`	A SHA1 hash of the certificate.
`cert.issuer.alternative.name`	An alternative name for the certificate issuer from the `IssuerAltName` extension (`OID = 2.5.29.18`).
`cert.issuer.id`	The DName of the issuer of the certificate.
`cert.issuer.id.c`	The c attribute of the issuer of the certificate, if it exists.

Generated Message Attribute	***Contains***
cert.issuer.id.cn	The cn attribute of the issuer of the certificate, if it exists.
cert.issuer.id.emailaddress	The email or emailaddress attribute of the issuer of the certificate, if it exists.
cert.issuer.id.l	The l attribute of the issuer of the certificate, if it exists.
cert.issuer.id.o	The o attribute of the issuer of the certificate, if it exists.
cert.issuer.id.ou	The ou attribute of the issuer of the certificate, if it exists.
cert.issuer.id.st	The st attribute of the issuer of the certificate, if it exists.
cert.key.usage.cRLSign	Set to true or false if the key can be used for crlSign.
cert.key.usage.dataEncipherment	Set to true or false if the key can be used for dataEncipherment.
cert.key.usage.decipherOnly	Set to true or false if the key can be used for decipherOnly.
cert.key.usage.digitalSignature	Set to true or false if the key can be used for digital signature.
cert.key.usage.encipherOnly	Set to true or false if the key can be used for encipherOnly.
cert.key.usage.keyAgreement	Set to true or false if the key can be used for keyAgreement.
cert.key.usage.keyCertSign	Set to true or false if the key can be used for keyCertSign.
cert.key.usage.keyEncipherment	Set to true or false if the key can be used for keyEncipherment.
cert.key.usage.nonRepudiation	Set to true or false if the key can be used for nonrepudiation.
cert.not.after	Not after validity period date.
cert.not.before	Not before validity period date.
cert.serial.number	Certificate serial number.
cert.signature.algorithm	The signature algorithm for certificate signature.
cert.subject.alternative.name	An alternative name for the subject from the SubjectAltName extension (OID = 2.5.29.17).
cert.subject.id	The DName of the subject of the certificate.
cert.subject.id.c	The c attribute of the subject of the certificate, if it exists.
cert.subject.id.cn	The cn attribute of the subject of the certificate, if it exists.
cert.subject.id.emailaddress	The email or emailaddress attribute of the subject of the certificate, if it exists.
cert.subject.id.l	The l attribute of the subject of the certificate, if it exists.
cert.subject.id.o	The o attribute of the subject of the certificate, if it exists.
cert.subject.id.ou	The ou attribute of the subject of the certificate, if it exists.
cert.subject.id.st	The st attribute of the subject of the certificate, if it exists.
cert.version	The certificate version.

Configuration

Name:
Enter a name for the filter.

Certificate Attribute:
The **Extract Certificate Attributes** filter extracts the attributes from the certificate contained in the message attribute selected or entered here. The selected attribute must contain a single certificate only.

Include Distribution Points:
If the certificate contains CRL Distribution Point X.509 extension attributes (which point to the location of the certificate issuer's CRL), you can also extract these and store them in message attributes by selecting this checkbox. The extracted distribution points are stored in message attributes that are prefixed by:
`distributionpoint.`

Certificate Chain Check

Overview

It is a trivial task for a user to generate a structurally sound X.509 certificate, and use it to negotiate mutually authenticated connections to publicly available services. However, this scenario is a security nightmare for IT administrators. You can not allow every user to generate their own certificate and use it on the Internet. For this reason, the API Gateway can establish the authenticity of the client certificate by ensuring that the certificate originated from a trusted source. To do this, a server can perform a *certificate chain check* on the client certificate.

The main purpose of certificate chain validation is to ensure that a certificate has been issued by a trusted source. Typically, in a Public Key Infrastructure (PKI), a Certificate Authority (CA) is responsible for issuing and distributing certificates. This infrastructure is based on the premise of *transitive trust*—if everybody trusts the CA, everybody transitively trusts the certificates issued by that CA. If entities only trust certificates that have been issued by the CA, they can reject certificates that have been self-generated by clients.

When a CA issues a certificate, it digitally signs the certificate and inserts a copy of its own certificate into it. This is called a certificate chain. Whenever an application (such as the API Gateway) receives a client certificate, it can extract the issuing CA certificate from it, and run a certificate chain check to determine whether it should trust the CA. If it trusts the CA, it also trusts the client certificate.

The API Gateway maintains a repository of trusted CA certificates, which is known as the **Certificate Store**. To *trust* a specific CA, that CA certificate must be imported into the **Certificate Store**. For more details, see the *Certificates and Keys* topic.

Configuration

You can configure the following settings on the **Certificate Chain Check** screen:

Name:
Enter an appropriate name for this filter.

Certificates Message Attribute:
You can specify a message attribute that contains the certificate or certificates to check. The message attribute type can be an `X509Certificate` object, or an `ArrayList` of `X509Certificate` objects.

Distinguished Name:
This table lists the Distinguished Names of the certificates currently in the **Certificate Store**. Select the checkbox beside a CA to enable this filter to consider it as *trusted* when performing the certificate chain check. You can select multiple CAs in the table.

OCSP Certificate Validation

Overview

Online Certificate Status Protocol (OCSP) is an automated certificate checking network protocol. The API Gateway can query an OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

To validate a certificate using an OCSP lookup, the issuing CA certificate should be trusted by the API Gateway. This is because for an OCSP request, the protocol stipulates that the CA public key must be submitted as part of the request. The issuing CA public key is not always included in the certificates that it issues, so it is necessary to retrieve it from the API Gateway's certificate store instead. For more information on how to trust CA certificates, see the *Certificates and Keys* tutorial.

Configuration

Configure the following fields on the **Certificate Validation - OCSP** dialog:

Name:
Enter an appropriate name for this OCSP filter.

OCSP Connection:
Click the button on the right, and select an OCSP connection in the tree. To add an OCSP connection, right-click the **OCSP Connections** node, and select **Add an OCSP Connection**. Alternatively, you can configure an OCSP connection under the **External Connections** node in the Policy Studio tree. For more details, see the *OCSP Certificate Validation Connection* topic.

OCSP Certificate Validation Connection

Overview

Online Certificate Status Protocol (OCSP) is an automated certificate checking network protocol. The API Gateway can query an OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

To validate a certificate using an OCSP lookup, the issuing CA certificate should be trusted by the API Gateway. This is because for an OCSP request, the protocol stipulates that the CA public key must be submitted as part of the request. The issuing CA public key is not always included in the certificates that it issues, so it is necessary to retrieve it from the API Gateway's certificate store instead. For more information on how to trust CA certificates, see the *Certificates and Keys* tutorial.

You can add OCSP Connections under the **External Connections** node in the Policy Studio tree. To add a global OCSP Connection, right-click the **OCSP Connections** node, and select **Add an OCSP Connection**.

Configuration

Configure the following fields on the **Certificate Validation - OCSP** dialog:

Name:
Enter a name for this OCSP connection.

URL Group:
Select a group of OCSP responders from the **URL Group** drop-down list.

The API Gateway attempts to connect to the OCSP responders in the selected group in a round-robin fashion. It attempts to connect to the responders with the highest priority first, before connecting to responders with a lower priority.

You can add, edit, or remove URL Groups by selecting the appropriate button. For more information on adding and editing URL groups, see the *Configuring URL Groups* topic.

User Name:
Requests to OCSP responders can be signed by a user to whom the **Sign OCSP or XKMS Requests** privilege has been assigned. Only those users who have been assigned this privilege are displayed in the drop-down list. For more information on assigning privileges to users, see the *API Gateway Users* tutorial.

Signing Key:
Click the **Signing Key** button to open the list of certificates in the Certificate Store. You can then select the key to use to sign requests to XKMS responders. This user must have been granted the Sign OCSP or XKMS Requests privilege.

Validate Response:
If the OCSP responders sign responses, select this checkbox to force the API Gateway to validate the signature on the response from the OCSP responder.

Validate Server's Certificate Store

Overview

This filter checks the API Gateway's certificate store for certificates that are due to expire before a specified number of days. This enables you to monitor the certificates that the API Gateway is running with.

For example, you can configure a policy that includes a **Validate Server's Certificate Store** filter and an **Alert** filter, which sends an email alert when it finds certificates that are due to expire. You can also configure this policy to run at regular intervals using the policy execution scheduler provided with the API Gateway.

Configuration

Configure the following fields on the **Validate API Gateway Certificate Store** screen:

Name:
Enter an appropriate name for the filter.

Days before expires:
Enter the number of days before the certificates are due to expire.

Check Server's Certificate Store:
Select whether to check the certificates in the API Gateway's Certificate store. This is selected by default.

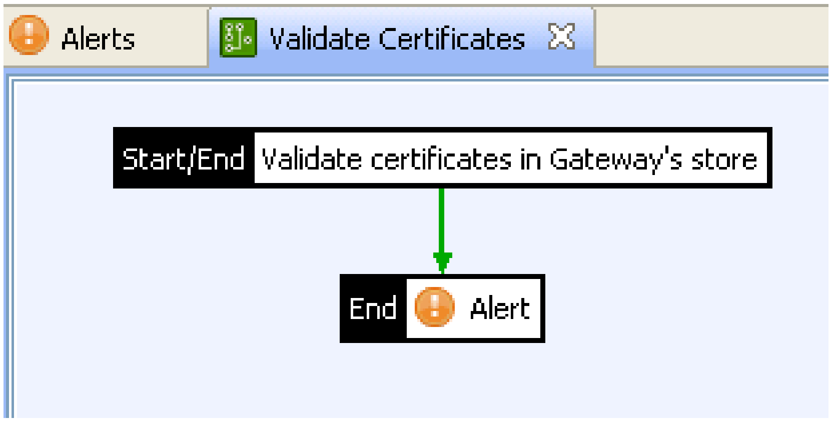
Check Server's Java Keystore:
Select whether to check the certificates in the API Gateway's Java Keystore. This is not selected by default. When selected, you must enter the **Password** for this keystore. The default is password is `changeit`.

Check Java Keystore:
Select whether to check the certificates in the specified Java Keystore. This is not selected by default. When selected, you must configure the following fields:

Keystore Location	Specify the path to this keystore (for example, `/home/oracle/osr-client.jks`).
Password	Enter the password for this keystore.

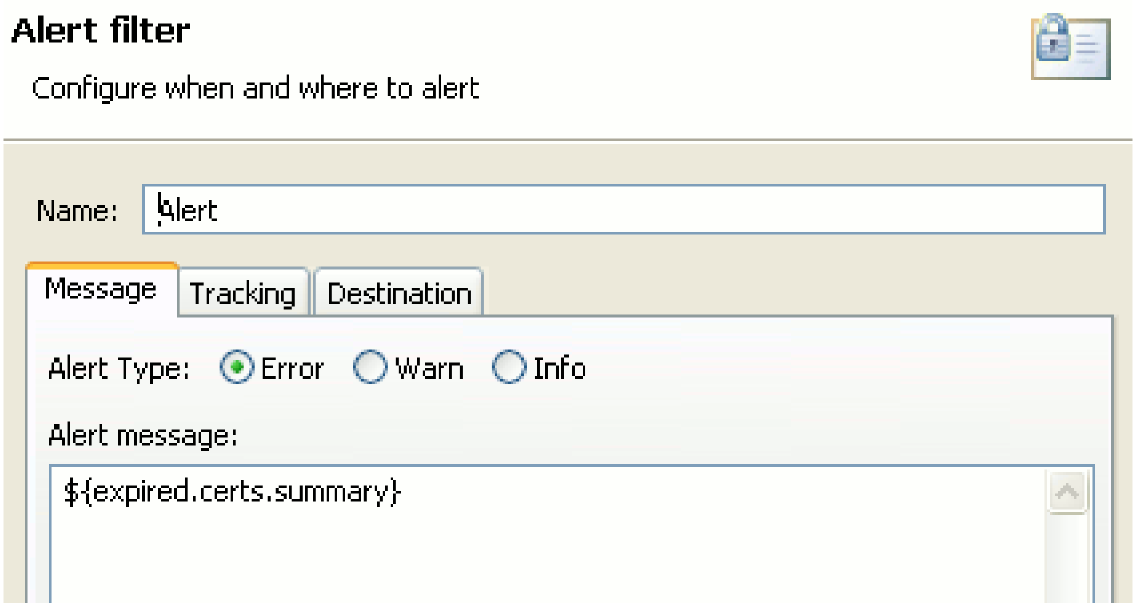
Deployment Example

The following example shows a **Validate Certificates** policy that includes a **Validate Certificates in API Gateway's Store** filter and an **Alert** filter. This policy sends an email alert when it finds certificates that are due to expire:



Configuring an Email Alert

When this filter is successful, and finds certificates that are due to expire, it generates an `expired.certs.summary` attribute, which contains a summary of certificates due to expire. You can then use this attribute in the **Alert** filter to send an email alert to the API Gateway administrators, as shown in the following example:



Name: Alert

Alert Type: ● Error ○ Warn ○ Info

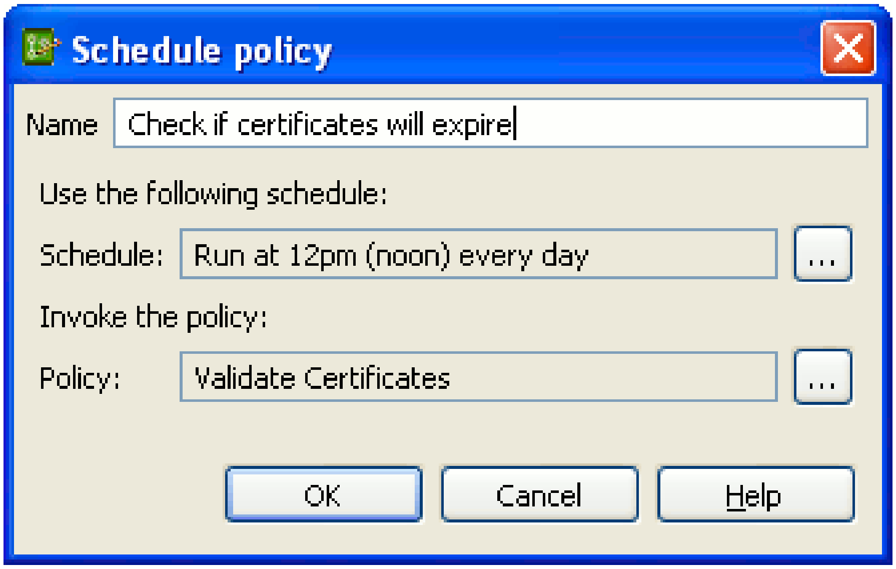
Alert message:

```
${expired.certs.summary}
```

You must also select a pre-configured email alert destination on the **Destination** tab (for example, **Email API Gateway Administrators**). For more details on configuring email alert destinations, see the *System Alerting* topic.

Configuring a Policy Execution Schedule

You can configure this policy to run at regular intervals (for example, once every day) using the policy scheduler provided with the API Gateway. Under the **Listeners** node, right-click the API Gateway instance node, and select **Add policy execution scheduler**. The following example runs the policy at 12 noon every day:



Name: Check if certificates will expire

Use the following schedule:

Schedule: Run at 12pm (noon) every day

Invoke the policy:

Policy: Validate Certificates

For more details, see the *Policy Execution Scheduling* topic.

Example Email Alert
An email alert is sent if any certificates that are due to expire are detected. The contents of the email are obtained from the `expired.certs.summary` message attribute. For example:

```
Oracle API Gateway running on Roadrunner contains certificates that will expire in 730 days.

2 expired certificates in API Gateway certificate store:

1. Cert details:
Cert issued to: CN=CA
Cert issued by: CN=CA
SHA1 fingerprint: 72:04:35:7C:A1:B1:C2:F5:E2:86:75:C4:83:12:9C:70:A8:D6:21:8E
MD5 fingerprint: 82:23:6F:59:F2:8F:C3:95:56:87:70:B5:51:3F:53:05
Subject Key Identifier (SKI): dfABenFoM0r7iJ3E1ZqU7HmKiyY=
Expires on: 2012-04-20

2. Cert details:
Cert issued to: CN=John Doe
Cert issued by: CN=CA
SHA1 fingerprint: 83:32:EB:3F:9C:15:87:FB:81:E1:D5:AC:CC:35:C3:F8:21:BB:DF:CD
MD5 fingerprint: 48:02:F6:3F:B9:64:EB:DA:DF:CF:F9:82:AC:CC:13:AB
Subject Key Identifier (SKI): HabJNMjAsBAWp4AcCq8yZkTEJKQ=
Expires on: 2012-04-20
```

XKMS Certificate Validation

Overview

XML Key Management Specification (XKMS) is an XML-based protocol that enables you to establish the trustworthiness of a certificate over the Internet. The API Gateway can query an XKMS responder to determine whether a given certificate can be trusted.

Configuration

You can configure the following fields on the **Certificate Validation - XKMS** screen.

Name:
Enter an appropriate name for this XKMS filter.

XKMS Connection:
Click the button on the right, and select an XKMS connection in the tree. To add an XKMS connection, right-click the **XKMS Connections** node, and select **Add an XKMS Connection**. Alternatively, you can configure an XKMS connection under the **External Connections** node in the Policy Studio tree. For more details, see the *XKMS Certificate Validation Connection* topic.

XKMS Certificate Validation Connection

Overview

XML Key Management Specification (XKMS) is an XML-based protocol that enables you to establish the trustworthiness of a certificate over the Internet. The API Gateway can query an XKMS responder to determine whether a given certificate can be trusted.

You can add XKMS Connections under the **External Connections** tree node in the Policy Studio. To add a global XKMS Connection, right-click the **XKMS Connections** node, and select **Add an XKMS Connection**.

Configuration

Configure the following fields on the **Certificate Validation - XKMS** screen.

Name:
Enter an appropriate name for this XKMS connection.

URL Group:
Select a group of XKMS responders from the URL Group drop-down list. The API Gateway attempts to connect to the XKMS responders in the selected group in a round-robin fashion. It attempts to connect to the responders with the highest priority first, before connecting to responders with a lower priority.

You can add, edit, or remove URL Groups by selecting the appropriate button. For more information on adding and editing URL groups, see the *Configuring URL Groups* topic.

User Name:
Requests to XKMS responders can be signed by a user to whom the **Sign OCSP or XKMS Requests** privilege has been assigned. Only those users who have been assigned this privilege are displayed in the drop-down list. For more information on assigning privileges to users, see the *API Gateway Users* tutorial.

Signing Key:
Click the **Signing Key** button to open the list of certificates in the Certificate Store. You can then select the key to use to sign requests to XKMS responders. This user must have been granted the Sign OCSP or XKMS Requests privilege.

Cache Attribute

Overview

The **Cache Attribute** filter allows you to configure what part of the message you want to cache. Typically, response messages are cached and so this filter is usually configured *after* the routing filters in a policy. In this case, the `content.body` attribute stores the response message body from the Web Service and so this message attribute should be selected in the **Attribute Name to Store** field.

For more information on how to configure this filter in a caching policy, see the topic on *Global Caches*.

Configuration

Name:
Enter a name for this filter here.

Select Cache to Use:
Click the button on the right, and select the cache to store the attribute value. The list of currently configured caches is displayed in the tree. To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Attribute Key:
The value of the message attribute entered here acts as the key into the cache. In the context of a caching policy, it *must* be the same as the attribute specified in the **Attribute containing key** field on the **Is Cached?** filter.

Attribute Name to Store:
The value of the Oracle message attribute entered here will be cached in the cache specified in the **Cache to use** field above.

Create Key

Overview

The **Create Key** filter is used to identify the part of the message that determines whether a message is unique. For example, you can use the request message body to determine uniqueness so that if two successive identical message bodies are received by the API Gateway, the response for the second request is taken from the cache.

You can also use other parts of the request to determine uniqueness (for example, HTTP headers, client IP address, client SSL certificate, and so on). This means that you can use the **Create Key** filter to create keys for a range of different caching scenarios (for example, caching a user's role, or caching a session for a user).

For more information on how to configure this filter in the context of a caching policy, see the *Global Caches* tutorial. This shows the order in which caching filters such as the **Create Key** filter are placed in an example caching policy.

Configuration

Name:
Enter a suitable name for this filter.

Attribute Name:
Select or enter the name of the message attribute to use to determine whether an incoming request is unique or not. For example, if `http.request.clientcert` (the client SSL certificate) is selected, the API Gateway takes a cached response for successive requests in which the client SSL certificate is the same. Defaults to `content.body`.

Output attribute name:
Select or enter the name of the output message attribute to be used as the key for objects in the cache. Defaults to `message.key`. This attribute contains a hash of the request message, which can then be used as the key for objects in the cache.

Is Cached?

Overview

The **Is Cached?** filter looks up a named cache to see if a specified message attribute has already been cached. A message attribute (usually `message.key`) is used as the key to search for in the cache. If the lookup succeeds, the retrieved value overrides a specified message attribute, which is usually the `content.body` attribute.

For example, if a response message for a particular request has already been cached, the response message overrides the request message body so that it can be returned to the client using the **Reflect** filter.

For more information on how to configure this filter in the context of a caching policy, see the *Global Caches* tutorial.

Configuration

Name:
Enter a suitable name for this filter.

Select Cache to Use:
Click the button on the right, and select the cache to lookup to find the attribute specified in the **Attribute containing key** field below. The list of currently configured caches is displayed in the tree. To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Attribute containing key:
The message attribute entered here is used as the key to lookup in the cache. In the context of a caching policy, the attribute entered here *must* be the same as the attribute specified in the **Attribute key** field on the **Cache Attribute** filter.

Overwrite Attribute Name if Found:
Usually the `content.body` is selected here so that value retrieved from the cache (which is usually a response message) overrides the request `content.body` with the cached response, which can then be returned to the client using the **Reflect** filter.

Removed Cached Attribute

Overview

The **Remove Cached Attribute** filter allows you to delete a message attribute value that has been stored in a cache. Each cache is essentially a map of name-value pairs, where each value is keyed on a particular message attribute. For example, it is possible to store a cache of request messages according to their message ID. In this case the message's `id` attribute would be the key into the cache, which would store the value of the request message's `content.body` message attribute.

In this example, the **Remove Cached Attribute** filter can be used to remove a particular entry from the cache based on the run-time value of a particular message attribute. By specifying the `id` message attribute to remove, the API Gateway will look up the cache based on the value of the `id` message attribute. When it finds a matching message ID in the cache, it will remove the corresponding entry from the cache.

The example described above may be useful in cases where a request message may need to be cached and stored until the request has been fully processed and a response returned to the client. For example, if the request must be routed on to a back-end Web Service, but that Web Service is temporarily unavailable, it may be possible to configure the policy to re-send the cached request instead of forcing the client to retry.

For more information on how to configure a caching policy, see the topic on *Global Caches*.

Configuration

Name:
Enter a name for this filter here.

Select Cache to Use:
Click the button on the right, and select the cache that contains the cached values that have been keyed according to the message attribute specified below. The list of currently configured caches is displayed in the tree. To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Attribute Key:
Enter the message attribute that is used as the key into the cache in this field. At run-time, the API Gateway will populate the value of this message attribute, which will then be used to lookup the cache selected in the table above. If a match is found in the cache, the corresponding entry will be deleted from the cache.

ClamAV Anti-Virus

Overview

The API Gateway can check messages for viruses by connecting to a ClamAV daemon running on network. The ClamAV daemon inspects the message and if the daemon finds a virus, it returns a corresponding response to the API Gateway, which can then block the message, if necessary.

Configuration

Complete the following fields to configure the **ClamAV Anti-Virus** filter:

Name:
Enter an appropriate name for this filter.

ClamAV Daemon Host:
Enter the host name of the machine on which the ClamAV daemon is running.

ClamAV Daemon Port Number:
Enter the port on which the ClamAV daemon is listening.

Content Type Filtering

Overview

The *SOAP Messages with Attachments* specification introduced a standard for transmitting arbitrary files along with SOAP messages as part of a multipart MIME message. In this way, both XML and non-XML data, including binary data, can be encapsulated in a SOAP message. The more recent Direct Internet Message Encapsulation (DIME) specification describes another way of packaging attachments with SOAP messages.

The API Gateway can accept or block multipart messages with certain MIME or DIME content types. For example, you can configure a filter that blocks multipart messages that contain parts that are of type `image/jpeg`.

Allow or Deny Types

The **Content Type Filtering** screen lists the content types that are allowed or denied by this filter.

Allow Content Types:
Use this option if you wish to *accept* most content types, but only want to reject a few specific types. To allow or deny incoming messages based on their content types, complete the following steps:

1. Select the **Allow content types** radio button to allow multipart messages to be routed onwards. If you wish to allow all content types, you do not need to select any of the MIME types in the list.
2. To deny multipart messages with certain MIME or DIME types as parts, select the checkbox next to those types. Multipart messages containing parts of the MIME or DIME types selected here will be rejected.

Deny Content Types:
If you wish to *block* multipart messages containing most content types, but want to allow a small number of content types, select this option. To reject multipart messages based on the content types of their parts, complete the following steps:

1. Select the **Deny content types** radio button to reject multipart messages. If you wish to block all multipart messages, you do not need to select any of the MIME or DIME types in the list.
2. To allow messages with parts of a certain MIME or DIME type, select the checkbox next to those types. Multipart messages with parts of the MIME or DIME types selected here will be allowed. All other MIME or DIME types will be denied.

MIME and DIME types can be added by clicking the **MIME/DIME Registered Types** button. The next section describes how to add, edit, and remove MIME/DIME types.

Configuring MIME/DIME Types

The **MIME/DIME Settings** dialog enables you to configure new and existing MIME types. When a type has been added, you can configure the API Gateway to accept or block multipart messages with parts of this type.

Click the **Add** button to add a new MIME/DIME type, or highlight a type in the table, and select the **Edit** button to edit an existing type. To delete an existing type, select that type in the list, and click the **Remove** button. You can edit or add types using the **Configure MIME/DIME Type** dialog.

Enter a name for the new type in the **MIME or DIME Type** field, and the corresponding file extension in the **Extension** field.

Content Validation

Overview

This tutorial describes how the API Gateway can examine the contents of an XML message to ensure that it meets certain criteria. It uses boolean XPath expressions to evaluate whether or not a specific element or attribute contains has a certain value.

For example, you can configure XPath expressions to make sure the value of an element matches a certain string, to check the value of an attribute is greater (or less) than a specific number, or that an element occurs a fixed amount of times within an XML body.

There are two ways to configure XPath expressions on this screen. Please click the appropriate link below:

- Manual XPath Configuration
- XPath Wizard

Manual XPath Configuration

To manually configure a **Content Validation** rule using XPath:

1. Enter a meaningful name for this XPath content filter.
2. Click the **Add** button to add a new XPath expression. Alternatively, you can select a previously configured XPath expression from the drop-down list.
3. In order to resolve any prefixes within the XPath expression, the namespace mappings (i.e. **Prefix**, **URI**) should be entered in the table.

As an example of how this screen should be configured, consider the following SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sig1">
       ...............
       ...............
       ...............
       ...............
  </dsig:Signature>
 </soap:Header>
 <soap:Body>
  <prod:product xmlns:prod="http://www.company.com">
   <prod:name>SOA Product</prod:name>
   <prod:company>Company</prod:company>
   <prod:description>WebServices Security</prod:description>
  </prod:product>
 </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the `<company>` element contains the value `Company`:
XPath Expression: `//prod:company[text()='Company']`

In this case, you must define a mapping for the *prod* namespace as follows:

Prefix	***URI***
prod	http://www.company.com

In another example, the element to be examined by the XPath expression belongs to a default namespace. Consider the following SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sig1">
      ...............
      ...............
      ...............
      ...............
  </dsig:Signature>
 </soap:Header>
 <soap:Body>
  <product xmlns="http://www.company.com">
   <name>SOA Product</name>
   <company>Company</company>
   <description>Web Services Security</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the `<company>` element contains the value `Company`:
XPath Expression: `//ns:company[text()='Company']`

Because the `<company>` element belongs to the default (`xmlns`) namespace (`http://www.company.com`, you must make up an arbitrary prefix (`ns`) for use in the XPath expression, and assign it to `http://www.company.com`. This is necessary to distinguish between potentially several default namespaces which may exist throughout the XML message. The following mapping illustrates this:

Prefix	***URI***
`ns`	`http://www.company.com`

XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard enables administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned. To launch the **XPath Wizard**, click the **XPath Wizard Button** on the **XPath Expression** dialog.

To use the XPath Wizard, enter (or browse to) the location of an XML file in the **File** field. The contents of the XML file are displayed in the main window of the wizard. Enter an XPath expression in the **XPath** field, and click the **Evaluate** button to run the XPath against the contents of the file. If the XPath expression returns any elements (or returns true), those elements are highlighted in the main window.

If you are not sure how to write the XPath expression, you can select an element in the main window. An XPath expression to isolate this element is automatically generated and displayed in the **Selected** field. If you wish to use this expression, select the **Use this path** button, and click **OK**.

HTTP Header Validation

Overview

The API Gateway can check HTTP header values for threatening content. This ensures that only properly configured name-value pairs appear in the HTTP request headers. *Regular expressions* are used to test HTTP header values. This enables you to make decisions on what to do with the message (for example, if the HTTP header value is X, route to service X).

You can configure the following sections on the **Validate HTTP Headers** screen:

- **Enter Regular Expression**:
  HTTP header values can be checked using regular expressions. You can select regular expressions from the global **White list** or enter them manually. For example, if you know that an HTTP header must have a value of `ABCD`, a regular expression of `^ABCD$` is an exact match test.
- **Enter Threatening Content Regular Expression**:
  You can select threatening content regular expressions from the global **Black list** to run against all HTTP headers in the message. These regular expressions identify common attack signatures (for example, SQL injection attacks).

You can configure the global **White list** and **Black list** libraries of regular expressions under the **Libraries** node in the Policy Studio tree.

Configuring HTTP Header Regular Expressions

The **Enter Regular Expression** table displays the list of configured HTTP header names together with the **White list** of regular expressions that restrict their values. For this filter to run successfully, *all* required headers must be present in the request, and *all* must have values matching the configured regular expressions.

The **Name** column shows the name of the HTTP header. The **Regular Expression** column shows the name of the regular expression that the API Gateway uses to restrict the value of the named HTTP header. A number of common regular expressions are available from the global **White list** library.

Configuring a Regular Expression
You can configure regular expressions by selecting the **Add**, **Edit**, and **Delete** buttons. The **Configure Regular Expression** dialog enables you to add or edit regular expressions to restrict the values of HTTP headers. To configure a regular expression, perform the following steps:

1. Enter the name of the HTTP header in the **Name** field.
2. Select whether this header is **Optional** or **Required** using the appropriate radio button. If it is **Required**, the header *must* be present in the request. If the header is not present, the filter fails. If it is **Optional**, the header does not need to be present for the filter to pass.
3. You can enter the regular expression to restrict the value of the HTTP header manually or select it from the global **White list** library of regular expressions in the **Expression Name** drop-down list. A number of common regular expressions are provided (for example, alphanumeric values, dates, and email addresses).
   You can use selectors representing the values of message attributes to compare the value of an HTTP header with the value contained in a message attribute. Enter the $ character in the **Regular Expression** field to view a list of available attributes. At runtime, the selector is expanded to the corresponding attribute value, and compared to the HTTP header value that you want to check. For more details on selectors, see *Selecting Configuration Values at Runtime*.
4. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

Advanced Settings

The **Advanced** section enables you to extract a portion of the header value which is run against the regular expression. The extracted substring can be Base64 decoded if necessary. This section is specifically aimed towards HTTP Basic authentication headers, which consist of the `Basic ` prefix (with a trailing space), followed by the Base64-encoded username and password. The following is an example of the HTTP Basic authentication header:

```
Authorization: Basic dXNlcjp1c2Vy
```

The Base64-encoded portion of the header value is what you are interested in running the regular expression against. You can extract this by specifying the string that occurs directly before the substring you want to extract, together with the string that occurs directly after the substring.

To extract the Base64-encoded section of the `Authorization` header above, enter `Basic ` (with a trailing space) in the **Start substring** field, and leave the **End substring** field blank to extract the entire remainder of the header value.

Important

You must select the start and end substrings to ensure that the exact substring is extracted. For example, in the HTTP Basic example above, you should enter `Basic ` (with a trailing space) in the **Start substring** field, and *not* `Basic` (with no trailing space).

By specifying the correct substrings, you are left with the Base64-encoded header value (`dXNlcjp1c2Vy`). However, you still need to Base64 decode it before you can run a regular expression on it. Make sure to select the **Base64 decode** checkbox. The Base64-decoded header value is `user:user`, which conforms to the standard format of the `Authorization` HTTP header. This is the value that you need to run the regular expression against.

The following example shows an example of an HTTP Digest authentication header:

```
Authorization: Digest username="user", realm="oracle.com", qop="auth",
algorithm="MD5", uri="/editor", nonce="Id-00000109924ff10b-0000000000000091",
nc="1", cnonce="ae122a8b549af2f0915de868abff55bacd7757ca",
response="29224d8f870a62ce4acc48033c9f6863"
```

You can extract single values from the header value. For example, to extract the `realm` field, enter `realm="` (including the `"` character), in the **Start substring** field and `"` in the **End substring** field. This leaves you with `oracle.com` to run the regular expression against. In this case, there is no need to Base64 decode the extracted substring.

Note

If both **Start substring** and **End substring** fields are blank, the regular expression is run against the entire header value. Furthermore, if both fields are blank and the **Base64 decode** checkbox is selected, the entire header value is Base64 encoded before the regular expression is run against it.

While the above examples deal specifically with the HTTP authentication headers, the interface is generic enough to enable you to extract a substring from other header values.

Configuring Threatening Content Regular Expressions

The regular expressions entered in this section guard against the possibility of an HTTP header containing malicious content. The **Enter Threatening Content Regular Expression** table lists the **Black list** of regular expressions to run to ensure that the header values do not contain threatening content.

For example, to guard against an SQL `DELETE` attack, you can write a regular expression to identify SQL syntax and add it to this list. The **Threatening Content Regular Expressions** are listed in a table. *All* of these expressions are run against *all* HTTP header values in an incoming request. If the expression matches *any* of the values, the filter fails.

Important

If any regular expressions are configured in the Configuring HTTP Header Regular Expressions section, these expressions are run *before* Threatening Content Regular Expressions (TCRE) are run. For example, if you already configured a regular expression to extract the Base64-decoded value of the `Authentication` header value in the example above, the TCRE is run against this value instead of the attribute value that appears in the HTTP header.

You can add threatening content regular expressions using the **Add** button. You can edit or remove existing expressions by selecting them in the drop-down list, and clicking the **Edit** or **Delete** button.

You can enter the regular expressions manually or select them from the global **Black list** library of threatening content regular expressions. This library is pre-populated with a number of regular expressions that scan for common attack signatures. These include expressions to guard against common SQL injection-style attacks (for example, SQL `INSERT`, SQL `DELETE`, and so on), buffer overflow attacks (content longer than 1024 characters), and the presence of control characters in attribute values (ASCII control characters).

Enter or select an appropriate regular expression to restrict the value of the specified HTTP header. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

ICAP Filter

Overview

You can use an **ICAP** filter to send a message to a pre-configured ICAP Server for content adaptation. For example, this includes specific operations such as virus scanning, content filtering, ad insertion, and language translation. For more details, see the topic on *Configuring ICAP Servers*.

Configuration

Configure the following settings:

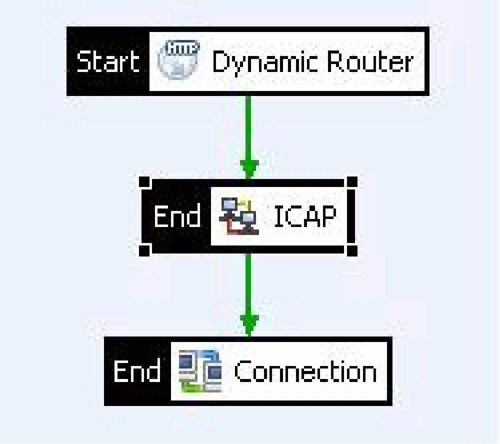
Name:
Enter an appropriate name for the filter.

ICAP Server:
Click the button next to this field, and select a pre-configured ICAP Server in the tree. To add an ICAP Server, right-click the **ICAP Servers** tree node, and select **Add an ICAP Server**. Alternatively, you can configure ICAP Servers under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring ICAP Servers*.

Example Policies

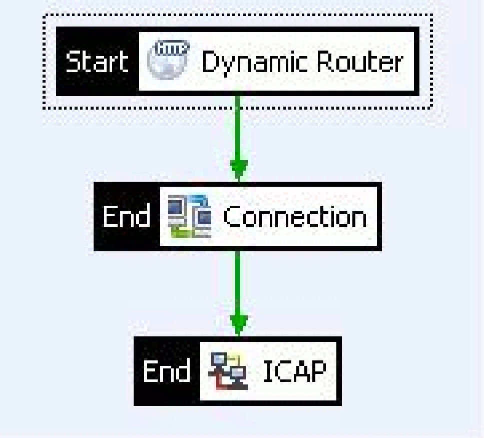
This section shows some example use cases of the **ICAP** filter configured in policies.

Request Modification Mode
The following policy shows an ICAP filter used in Request Modification (REQMOD) mode:



This example policy is essentially an internet proxy but with all incoming messages being sent to an ICAP server for virus-checking before being sent to the destination. All ICAP server-bound messages in this instance are REQMOD requests.

Response Modification Mode
The following policy illustrates an ICAP Filter used in Response Modification (RESPMOD) mode:



This example policy also is an internet proxy but with all responses being sent to an ICAP server for virus-checking after being sent to the destination and before being sent back to the client. All ICAP server-bound messages in this instance are RESPMOD requests.

Further Information

For more details on the REQMOD and RESPMOD modes, see the topic on *Configuring ICAP Servers*.

McAfee Anti-Virus

Overview

The **McAfee Anti-Virus** filter scans incoming HTTP requests and their attachments for viruses and exploits. For example, if a virus is detected in a MIME attachment or in the XML message body, the API Gateway can reject the entire message and return a SOAP Fault to the client. In addition, this filter supports cleaning of messages from infections such as viruses and exploits. It also provides scan type presets for different detection levels, and reports overall message status after scanning.

Note

The **McAfee Anti-Virus** filter is available on Windows and Linux only.

Prerequisites

McAfee virus scanner integration requires the McAfee 5400 Scan Engine.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.dat` files from the McAfee 5400 Scan Engine to the *`install-dir`*`/conf/plugin/mcafee/datv2` directory, except for `config.dat` which must be added to *`install-dir`*`/platform/lib` or *`install-dir`*`\Win32\lib`.
   - Add `.jar` files to the *`install-dir`*`/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`install-dir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

Configuring a McAfee Anti-Virus Filter

To configure the **McAfee Anti-Virus** filter, perform the following steps:

1. Enter an appropriate name in the **Name** field.
2. Select a **Scan type** from the drop-down box. The available options are as follows:

Normal	Processes the entire message detecting exploits and viruses in the message headers, macros, multi-file archives, executables, MIME-encoded/UU-encoded/XX-encoded/BinHex and TNEF/IMC format files. Performs heuristic analysis to find new viruses and potentially unwanted programs. This is the default scan type.

Fast	Detects infections in the top level of each message part, such as exploits that use headers and multiple bodies. The detection is less precise, but the performance is better if the top-level object is infected.
Multi-pass	Combines the **Normal** and **Fast** scan types. The **Fast** scan (pass 1) runs first on the whole message with no cleaning. The scanner stops if it finds an infected object, and if the clean type is set to **No cleaning**, the scanner reports the infection, or otherwise deletes the message. If pass 1 does not detect any virus or exploit, the **Normal** scan (pass 2) runs with the specified clean type and provides more precise detection.
Custom	Enables you to set the **Custom options** described in the next section. This provides compatibility with previous API Gateway versions. **Note** When existing policies are upgraded to the current API Gateway version, the **McAfee Anti-Virus** filter scan type is set to **Custom** and the clean type is set to **No cleaning** for backward compatibility.

3. Select a **Clean type** from the drop-down box. The available options are as follows:

No cleaning	Fails if any infection is detected. This is the default clean type.
Always remove infected parts	Removes the infected message part, and does not try to repair it.
Attempt to repair infected parts	Attempts to repair the found infection (if repairable), otherwise deletes the infected message part.

Configuring Custom Options

When you configure a custom scan type, the following **Custom options** are available:

Decompress Archives:
This instructs the filter to scan each file in an archive for viruses. Types of archived files include the ZIP, JAR, TAR, ARJ, LHA, PKARC, PKZIP, RAR, WinACE, BZip, and Zcompress formats.

Decompress Executables:
Executables are sometimes compressed to decrease overall message size. In such cases, any embedded viruses are also compressed and may be missed by conventional scans. If this option is selected, the filter decompresses the executable before scanning it for viruses.

Fail Any Macros:

A *macro* is a series of commands that can be invoked in a single command or keystroke. While calling the macro can appear to be harmless, the initiated command sequence may be harmful. Macros are usually configured to run automatically when the host document is opened. When this option is selected, the API Gateway fails if any macro is detected in a compound document (whether it matches a virus signature or not). An appropriate SOAP Fault is returned to the client.

Heuristic Program Analysis:
A heuristic virus detection algorithm runs a series of probing tests on a file in an attempt to solicit virus-like behavior from it. Based on the results of these tests, the algorithm can then make an educated guess on whether the file represents a potential threat or not. For example, programs that attempt to modify or delete files, invoke email clients, or replicate themselves all display virus-like behavior and so may be treated as viruses by the scanner.

The major advantage of this type of analysis is that new viruses can be detected. With the signature detection method, the scanner attempts to find a fixed number of known virus signatures in a file. Because the number of known signatures is fixed, new or unknown viruses can not be detected. If this option is selected, the filter runs heuristic analysis on executables only.

Heuristic Macro Analysis:
When this option is enabled, the filter runs heuristic detection analysis on macros contained in any body parts of the message. If any viruses are detected, the message is blocked. If this option is selected, the API Gateway searches for virus signatures in the respective body parts of a MIME message. However, it can only search for *known* viruses using this method.

Note

Macros embedded in MIME parts are also scanned for virus signatures.

Scan Embedded Scripts:
The API Gateway can scan MIME parts, such as HTML documents, for embedded scripts. If this option is selected, the filter scans for embedded scripts.

Scan for Test Files:
When this option is selected, the API Gateway fails if it encounters an anti-virus test file (for example, `eicar.com`). This is a convenient way to check that the anti-virus filter successfully detects known viruses.

Reporting Message Status

When the scan is complete, the **McAfee Anti-Virus** filter reports the overall message status in the `mcafee.status` message attribute, which is generated by the filter. This reflects the overall status of the scan for all message parts, and includes one of the following values:

`NOVIRUS`	No virus or exploit detected in the message.
`INFECTED`	Infection detected in the message.
`REPAIRED`	Message repaired.
`REMOVED`	Some or all message parts successfully removed.
`REPAIRED, REMOVED`	Some message parts successfully repaired and some others removed.

Loading McAfee Updates

When the **McAfee Anti-Virus** filter has been loaded, it searches for virus definitions in the following directory:

```
install-dir\conf\plugin\mcafee\datv2
```

When these have been loaded, it periodically checks for the presence of a directory named as follows:

```
install-dir\conf\plugin\mcafee\datv2.new
```

directory.

If the `datv2.new` directory is found, the scanner is stopped and the `datv2` directory is renamed to `datv2.0`. If a `datv2.0` directory already exists from a previous rollover, a `datv2.1` directory is created instead, and so on, until an unused index is used. This means that the server never deletes the old files, and rolls them out of the way.

When the engine is stopped and restarted, any messages that require scanning are suspended until the restart completes. In addition, an initiated reload is suspended until all currently active scans are completed.

Important

Like all file system scanning approaches, there is an inherent ordering problem. If you create the `datv2.new` directory before copying the files into the directory, the scanner may pick up the new directory before it is ready to be used. For example, on Windows, you may experience problems if you enter the following commands from the *`install-dir`*`\conf\plugin\mcafee` directory:

```
mkdir datv2.new
copy c:\mcafee\newfiles\*.* datv2.new
```

You can use the following commands to prevent this problem:

```
mkdir datv2.tmp
copy c:\mcafee\newfiles\*.* datv2.tmp
rename datv2.tmp datv2.new
```

These create a temporary folder, copy the files into this folder, and rename the temporary folder to `datv2.new`. In this way, the scanner is guaranteed to pick up the virus definition files when it detects the new directory.

On Linux, the same approach applies, but the location of the file and the commands used are different. For example, enter the following commands from the *`install-dir`*`/conf/plugin/mcafee` directory:

```
mkdir datv2.tmp
cp /var/tmp/mcafee/newfiles/*.* datv2.tmp
mv datv2.tmp datv2.new
```

Message Size

Overview

It is sometimes useful to filter incoming messages based, not only on the content of the message, but on external characteristics of the message. To this end, the API Gateway can be configured to reject messages that are greater or less than a specified size.

Configuration

To configure the API Gateway to block messages of a certain size, complete the following fields:

- Enter the size (in bytes) of the smallest message that should be processed in the **At least** field. Messages smaller than this size will be rejected.
- Enter the size (in bytes) of the largest message that should be processed in the **At most** field. Messages larger than the size entered here will be rejected.
- The **Use in Size Calculation** options are used to specify the portion of the message that is to be used when calculating the size of the message.
    - If the **Root body only** option is selected, the API Gateway will calculate the size of the message body excluding all other MIME parts, i.e. attachments.
    - If the **Attachments only** option is selected, the API Gateway will only calculate the size of all attachments to the message. It will exclude the size of the root body payload from its calculation.
    - Finally, if the **Root body and attachments** option is selected, the API Gateway will include the root body together with all other MIME parts when it calculates the size of the message.

Important

The message size measured by the API Gateway does *not* include HTTP headers.

Query String Validation

Overview

The API Gateway can check the request *query string* to ensure that only properly configured name and value pairs appear. *Regular expressions* are used to test the attribute values. This enables you to make decisions on what to do with the message (for example, if the query sting value is X, route to service X)

You can configure the following sections on the **Validate Query String** screen:

- **Enter Regular Expression**:
  Query string values can be checked using regular expressions. You can select regular expressions from the global **White list** or enter them manually. For example, if you know that a query string must have a value of ABCD, a regular expression of ^ABCD$ is an exact match test.
- **Enter Threatening Content Regular Expression**:
  You can select threatening content regular expressions from the global **Black list** to run against all query string names and values. These regular expressions identify common attack signatures (for example, SQL injection attacks).

You can configure the global **White list** and **Black list** libraries of regular expressions under the **Libraries** node in the Policy Studio tree.

Request Query String

The request query string is the portion of the URL that comes after the ? character, and contains the request parameters. It is typically used for HTTP GET requests in which form data is submitted as name-value pairs on the URL. This contrasts with the HTTP POST method where the data is submitted in the body of the request. The following example shows a request URL that contains a query string:

```
http://hostname.com/services/getEmployee?first=john&last=smith
```

In this example, the query string is `first=john&last=smith`. Query strings consist of attribute name-value pairs, and each name-value pair is separated by the & character.

The **Query String Validation** filter can also operate on the form parameters submitted in an HTTP Form POST. Instead of encoding the request parameters in the query string, the client uses the `application/x-www-form-urlencoded` content-type, and submits the parameters in the HTTP POST body, for example:

```
POST /services/getEmployee HTTP/1.1
Host: localhost:8095
Content-Length: 21
SOAPAction: HelloService
Content-Type: application/x-www-form-urlencoded

first=john&last=smith
```

If the API Gateway receives an HTTP request body such as this, the **Query String Validation** filter can validate the form parameters.

Configuring Query String Attribute Regular Expressions

The **Enter Regular Expression** table displays the list of configured query string names together with the white list of regular expressions that restrict their values. For this filter to run successfully, *all* required attributes must be present in the request, and *all* must have the correct value.

The **Name** column shows the name of the query string attribute. The **Regular Expression** column shows the name of the regular expression that the API Gateway uses to restrict the value of the named query string attribute. A number of common regular expressions are available from the global **White list** library.

If the **Allow unspecified names** checkbox is selected, additional unnamed query string attributes are not filtered by the API Gateway. For example, this is useful if you are interested in filtering the content of only a small number of query string attributes but the request may contain many attributes. In such cases, you only need to filter those few attributes, and by selecting this checkbox, the API Gateway ignores all other query string attributes.

Configuring a Regular Expression

You can configure regular expressions by selecting the **Add**, **Edit**, and **Delete** buttons. The **Configure Regular Expression** dialog enables you to add or edit regular expressions to restrict the values request query string attributes. To configure a regular expression, perform the following steps:

1. Enter the name of the query string attribute in the **Name** field.
2. Select whether this request parameter is **Optional** or **Required** using the appropriate radio button. If it is **Required**, the parameter name *must* be present in the request. If the parameter is not present, the filter fails. If it is **Optional**, the attribute does not need to be present for the filter to pass.
3. You can enter the regular expression to restrict the value of the query string attribute manually or select it from the global **White list** library of regular expressions in the **Expression Name** drop-down list. A number of common regular expressions are provided (for example, alphanumeric values, dates, and email addresses).
   You can use selectors representing the values of message attributes to compare the value of the query string attribute with the value contained in a message attribute. Enter the `$` character in the **Regular Expression** field to view a list of available attributes. At runtime, the selector is expanded to the corresponding attribute value, and compared to the query string attribute value that you want to check.
4. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

Advanced Settings

The **Advanced** section enables you to extract a portion of the query string attribute value that is run against the regular expression. The extracted substring can also be Base64 decoded if necessary. The following is an example of a URL containing a query string. The value of the `password` attribute is Base64 encoded, and must be extracted from the query string and decoded before it is run against the regular expression.

```
http://oracle.com/services?username=user&password=dXNlcg0K&dept=eng
```

You can extract the encoded value of the `password=` attribute value by specifying the string that occurs directly before the substring you want to extract, together with the string that occurs directly after the substring. Enter `password=` in the **Start substring** field, and & in the **End substring** field.

Important

You must select the start and end substrings to ensure that the exact substring is extracted. For example, in this example, `password=` (including the equals sign) should be entered in the **Start substring** field, and **not** `password` (without the equals sign).

By specifying the correct substrings, you are left with the Base64-encoded attribute value (`dXNlcg0K`). However, you still need to Base64 decode it before you can run a regular expression on it. Make sure to select the **Base64 decode** checkbox. The Base64-decoded password value is simply `user`. This is the value that you want to run the regular expression against.

By specifying the correct substrings, you are left with the Base64-encoded attribute value (`dXNlcg0K`). However, you still need to Base64 decode it before you can run a regular expression on it. Make sure to select the **Base64 decode** checkbox. The Base64-decoded password value is `user`. This is the value that you need to run the regular expression against.

Note

If both **Start substring** and **End substring** fields are blank, the regular expression is run against the entire attribute value. Furthermore, if both fields are blank and the **Base64 decode** checkbox is selected, the entire attribute value is Base64 encoded before the regular expression is run against it.

Configuring Threatening Content Regular Expressions

The regular expressions entered in this section guard against the possibility of a query string attribute containing malicious content. The **Enter Threatening Content Regular Expression** table lists the **Black list** of regular expressions to run to ensure that the header values do not contain threatening content.

For example, to guard against an SQL `DELETE` attack, you can write a regular expression to identify SQL syntax and add it to this list. The **Threatening Content Regular Expressions** are listed in a table. *All* of these expressions are run against *all* attribute values in the query string. If the expression matches *any* of the values, the filter fails.

Important

If any regular expressions are configured in the Configuring Query String Regular Expressions section, these expressions are run *before* the Threatening Content Regular Expressions (TCRE) are run. For example, if you have already configured a regular expression to extract the Base64-decoded value of the `password` query string attribute as in the example above, the TCRE is run against this value instead of the attribute value that appears in the query string.

You can add threatening content regular expressions using the **Add** button. You can edit or remove existing expressions by selecting them in the drop-down list and clicking the **Edit** or **Delete** button.

You can enter the regular expressions manually or select them from the global **Black list** library of threatening content regular expressions. This library is pre-populated with regular expressions that guard against common attack signatures. These include a expressions to guard against common SQL injection style attacks (for example, SQL `INSERT`, SQL `DELETE`, and so on), buffer overflow attacks (content longer than 1024 characters), and the presence of control characters in attribute values (ASCII Control Character).

Enter or select an appropriate regular expression to restrict the value of the specified query string. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

Schema Validation

Overview

The API Gateway can check that XML messages conform to the structure or format expected by the Web Service by validating those requests against XML Schemas. An XML Schema precisely defines the elements and attributes that constitute an instance of an XML document. It also specifies the data types of these elements to ensure that only appropriate data is allowed through to the Web Service.

For example, an XML Schema might stipulate that all requests to a particular Web Service must contain a `<name>` element, which contains at most a ten character string. If the API Gateway receives a message with an improperly formed `<name>` element, it rejects the message.

You can find the **Schema Validation** filter in the **Content Filtering** category of filters in the Policy Studio. Drag and drop the filter on to the policy where you want to perform schema validation. The **Schema Validation** dialog has three tabs, which are explained in this topic.

Schema to Use

Select one of the following options:

Use Schema from WSDL of Web Service:
Instead of selecting a specific XML schema to run in this filter, you can select this option to dynamically use the appropriate SOAP operation schema from the current Web Service Context. When this option is selected, this filter has an additional required message attribute named `webservice.context`, which must be provided. This enables you to share this filter to perform validation across multiple Web Services.

Select which XML Schema to validate messages with:
This is the default option. Click the button on the right, and select a schema to validate incoming messages in the tree. To add a schema, right-click the **Schemas** tree node, and select **Add Schema**. You can select to load the schema from a file or from a WSDL URL. Alternatively, you can configure schemas under the **Resources** node in the Policy Studio tree. For more details on configuring schemas, see the following topics:

- *Global Schema Cache*
- *Web Service Repository*

Note

If you have a WSDL file that contains an XML Schema, and want to use this schema to validate the message, you can import the WSDL file into the **Web Services Repository**. The **WSDL Import Wizard** automatically creates a **Schema Validation** filter and incorporates it into the auto-generated policy. In this case, the top-level schema in the WSDL, which is imported into the **Web Services Repository**, is selected by default in the filter. In this way, if the schema imports other schemas, they are available to the filter at runtime when validating the message. For more details on importing WSDL files, see the *Web Service Repository* topic.

Part of Message to Match

A portion of the XML message can be extracted using an XPath expression. The API Gateway can then validate this portion against the specified XML Schema. For example, administrators may only want to validate the SOAP Body part of a SOAP message. In this case, they should enter or select an XPath expression that identifies the SOAP Body of the message. This portion should then be validated against an XML Schema that defines the structure of the SOAP Body for that particular message.

Click the **Add** or **Edit** buttons to add or edit an XPath expression using the **Enter XPath Expression** dialog. You can remove expressions by selecting the expression in the **XPath Expression** drop-down and clicking the **Delete** button.

On the **Enter XPath Expression** dialog, there are two ways to configure XPath expressions. For more details, see the following links:

- Manual Configuration
- XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard enables administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned.

Advanced

The following settings are available on the **Advanced** tab:

Allow RPC Schema Validation:
When the **Allow RPC Schema Validation** checkbox is selected, the filter makes a *best attempt* to validate an RPC-encoded SOAP message. An RPC-encoded message is defined in the WSDL as having an operation with the following characteristics:

- The `style` attribute of the `<soap:operation>` element is set to `document`.
- The `use` attribute of the `<soap:body>` element is set to `rpc`.

For details on the possible values for these attributes, see Section 3.5 [http://www.w3.org/TR/wsdl#_soap:body] of the WSDL specification.

The problem with RPC-encoded SOAP messages in terms of schema validation is that the schema contained in the WSDL file does not necessarily fully define the format of the SOAP message, unlike with `document-literal` style messages. With an RPC-encoded operation, the format of the message can be defined by a combination of the SOAP operation name, WSDL message parts, and schema-defined types. As a result, the schema extracted from a WSDL file may not be able to validate a message.

Another problem with RPC-encoded messages is that type information is included in each element that appears in the SOAP message. For such element definitions to be validated by a schema, the type declarations must be removed, which is precisely what the **Schema Validation** filter does if the checkbox is selected on this tab. It removes the type declarations and then makes a *best attempt* to validate the message.

However, as explained earlier, if some of the elements in the SOAP message are taken from the WSDL file instead of the schema (for example, when the SOAP operation name in the WSDL file is used as the wrapper element beneath the SOAP Body instead of a schema-defined type), the schema is **not** able to validate the message.

Inline MTOM Attachments into Message:
Message Transmission Optimization Mechanism (MTOM) provides a way to send binary data to Web Services in standard SOAP messages. MTOM leverages the include mechanism defined by XML Optimized Packaging (XOP), whereby binary data can be sent as a MIME attachment (similar to SOAP with Attachments) to a SOAP message. The binary data can then be referenced from within the SOAP message using the `<xop:Include>` element.

The following SOAP request contains a binary image that has been Base64-encoded so that it can be inserted as the contents of the `<image>` element:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>/aWKKapGGyQ=</image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>
```

When this message is converted to an MTOM message by the API Gateway (for example, using the *Extract MTOM Attachment* filter) the Base64-encoded content from the `<image>` element is replaced with an `<xop:Include>` element. This contains a reference to a newly created MIME part that contains the binary content. The following request shows the resulting MTOM message:

```
POST /services/uploadImages HTTP/1.1
Host: API Gateway Explorer
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
    start="<mymessage.xml@example.org>";
    start-info="text/xml"

--MIME_boundary
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>
     <xop:Include xmlns:xop='http://www.w3.org/2004/08/xop/include'
          href='cid:http://example.org/myimage.gif'/>
   </image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/gif
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/myimage.gif>

// binary octets for image

--MIME_boundary
```

When attempting to validate the MTOM message with an XML Schema, it is crucial that you are aware of the format of the `<image>` element. Will it contain the Base64-encoded binary data, or will it contain the `<xop:include>` element with a reference to a MIME part?

For example, the XML Schema definition for `<image>` element might look as follows:

```
<xsd:element name="image" maxOccurs="1" minOccurs="1"
                type="xsd:base64Binary"
                xmime:expectedContentTypes="*/*"
                xsi:schemaLocation="http://www.w3.org/2005/05/xmlmime"
                xmlns:xmime="http://www.w3.org/2005/05/xmlmime"/>
```

In this case, the XML Schema validator expects the contents of the `<image>` element to be `base64Binary`. However, if the message has been formatted as an MTOM message, the `<image>` element contains a child element, `<xop:Include>` that the schema knows nothing about. This causes the schema validator to report an error and the schema validation fails.

To resolve this issue, select the **Inline MTOM Attachments into Message** checkbox on the **Advanced** tab. At runtime, the schema validator replaces the value of the `<xop:Include>` element with the Base64-encoded contents of the MIME part to which it refers. This means that the message now adheres to the definition of the `<image>` element in the XML Schema (the element contains data of type `base64Binary`).

This standard procedure of interpreting XOP messages is described in Section 3.2 Interpreting XOP Packages [http://www.w3.org/TR/2004/CR-xop10-20040826/#interpreting_xop_packages] of the XML-binary Optimized Packaging (XOP) specification.

Reporting Schema Validation Errors

When a Schema validation check fails, the validation errors are stored in the `xsd.errors` Oracle message attribute. You can return an appropriate SOAP Fault to the client by writing out the contents of this message attribute.

For example, you can do this by configuring a *Set Message* filter to write a custom response message back to the client. Place the **Set Message** filter on the failure path of the **Schema Validation** filter. You can enter the following sample SOAP Fault message in the **Set Message** filter. Notice the use of the `${xsd.errors}` message attribute selector in the `<Reason>` element:

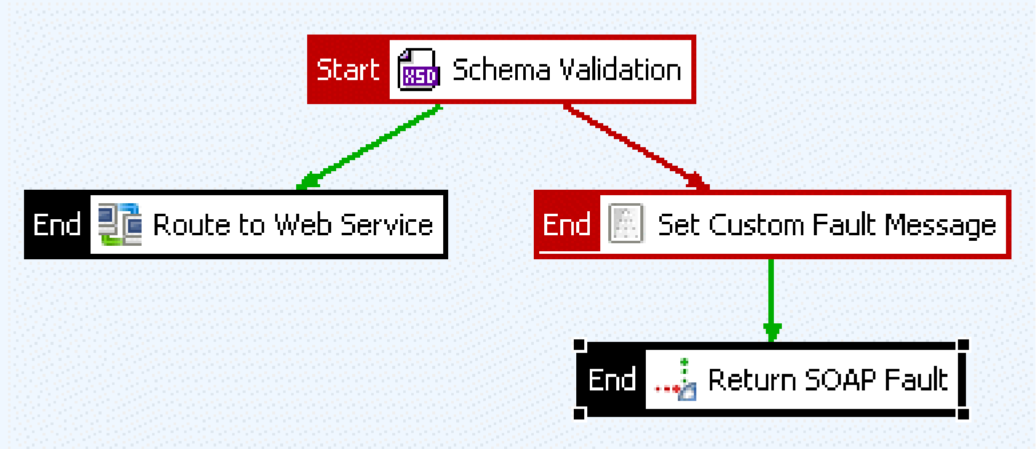
```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
        <env:Subcode>
          <env:Value xmlns:fault="http://www.Oracle.com/soapfaults">
            fault:MessageBlocked
          </env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">
          ${xsd.errors}
        </env:Text>
      </env:Reason>
      <env:Detail xmlns:fault="http://www.Oracle.com/soapfaults"
        fault:type="faultDetails">
      </env:Detail>
    </env:Fault>
  </env:Body>>
</env:Envelope>
```

At runtime, the error reported by the schema validator is set in the message. You can then return the SOAP Fault to the client using a *Reflect Message Filter* filter. The following example shows a SOAP Fault containing a typical schema validation error:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
        <env:Subcode>
          <env:Value xmlns:fault="http://www.Oracle.com/soapfaults">
            fault:MessageBlocked
          </env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
```

```
        <env:Text xml:lang="en">
          [XSD Error: Unknown element 'id' (line: 2, column: 8)]
        </env:Text>
      </env:Reason>
      <env:Detail xmlns:fault="http://www.Oracle.com/soapfaults"
        fault:type="faultDetails">
      </env:Detail>
    </env:Fault>
  </env:Body>>
</env:Envelope>
```

The following screenshot shows how to use the **Set Message** filter to return a customized SOAP Fault in a policy. If the **Schema Validation** filter succeeds, the message is routed on to the target Web Service. However, if the schema validation fails, the **Set Message** filter (named **Set Custom Fault Message**) is invoked. The filter sets the contents of the `xsd.errors` message attribute (the schema validation errors) to the custom SOAP Fault message as shown in the example error. The **Reflect** filter (named **Return SOAP Fault**) then writes the message back to the client.



JSON Schema Validation

Overview

The API Gateway can check that JavaScript Object Notation (JSON) messages conform to the format expected by a Web Service by validating requests against a specified JSON Schema. A JSON Schema precisely defines the items that constitute an instance of an incoming JSON message. It also specifies their data types to ensure that only appropriate data is allowed through to the Web Service.

For example, the following simple JSON Schema requires that all requests sent to a particular Web Service use this format:

```
{"description":"A geographical coordinate",
 "type":"object",
 "properties":{
    "latitude":{"type":"number"},
    "longitude":{"type":"number"}
    }
}
```

If a **JSON Schema Validation** filter is configured with this JSON schema, and the API Gateway receives an incorrectly formed message, the API Gateway rejects that message. For example, the following message would pass because it specifies the coordinates correctly as numbers:

```
{
    "latitude": 55.22,
    "longitude": 117.22
}
```

However, the following message would fail because it specifies the coordinates incorrectly as strings:

```
{
    "latitude": "55.22",
    "longitude": "117.22"
}
```

You can find the **JSON Schema Validation** filter in the **Content Filtering** category of filters in the Policy Studio. Drag and drop the filter on to the policy where you want to perform JSON schema validation.

For more details on JSON schemas, see http://www.json-schema.org.

Configuration

Configure the following settings:

Name:
Enter an appropriate name for this filter.

Select which JSON Schema to validate messages with:
Select one of the following options:

- **Use json schema file**
  Click the button on the right, and select a JSON schema to validate incoming messages from the tree in the dialog. To add a schema, right-click the **JSON Schemas** node, and select **Add Schema** to load the schema from a `.json` file. Alternatively, you can configure schemas under the **Resources** -> **JSON Schemas** node in the main Policy Studio tree. By default, the API Gateway provides the example JSON schemas available from ht-

tp://www.json-schema.org.

- **Use this class**
  Enter the Java classname used to specify the JSON schema (for example, `com.vordel.samples.GeoLocationTest`), and enter the name of message attribute to store the created object (for example, `my.geo.location`). This option enables you to take incoming JSON message data and deserialize it into a Java object.

  For example, to use a Java class for the geographical schema used in the previous section, perform the following steps:

  1. Create the annotated Java class as follows:

     ```
     package com.vordel.samples;
     import java.lang.String;
     import javax.xml.bind.annotation.XmlAttribute;
     import javax.xml.bind.annotation.XmlRootElement;

     @XmlRootElement
     public class GeoLocationTest {
         public GeoLocationTest() { }
         public int latitude;
         public int longitude;
     }
     ```

  2. Place this class in a JAR file, and put it in the API Gateway `ext/lib` directory.
  3. In the **JSON Schema Validation** filter screen, enter the following classname in the **Use this class** field: `com.vordel.samples.GeoLocationTest`.
  4. In the **Store created object in message attribute** field, enter `my.geo.location`.

  Now at runtime when the JSON message arrives, the API Gateway takes the JSON data and tries to instantiate a new `GeoLocationTest` object. If this succeeds, the **JSON Schema Validation** filter passes. However, if the object cannot be instantiated, the filter fails because the incoming data does not conform to the JSON schema specified by the annotated class.

Generating a JSON Schema Using Jython

The API Gateway also provides a Jython script to enable you to generate a JSON schema based on a specified Java annotated class. This script is available in the following directory of your API Gateway installation:

```
INSTALL_DIR/samples/scripts/json/schemagenerator.py
```

Given an annotated Java `.class` file, you can generate a schema from this and output a `.json` schema file. This schema can then be imported into the API Gateway schema library and used subsequently for future validations.

For example, using the Java class from the previous section, you can generate the schema as follows:

- **Windows**

  ```
  run.bat json\schemagenerator.py com.vordel.samples.GeoLocationTest
  MyLocationSchema.json
  ```

- **UNIX/Linux**

  ```
  sh run.sh json/schemagenerator.py com.vordel.samples.GeoLocationTest
  MyLocationSchema.json
  ```

Note

This script requires that you specify the location of your JAR file. You can do this by setting the JYTHONPATH environment variable, for example:

- **Windows**

  ```
  set JYTHONPATH=C:\home\user\mylocation.jar
  ```

- **UNIX/Linux**

  ```
  export JYTHONPATH=/home/user/mylocation.jar
  ```

Alternatively, if you have compiled your classes to `/home/user/classes`, specify the following:

```
export JYTHONPATH=/home/user/classes
```

For more details on using Jython, see http://www.jython.org/.

Sophos Anti-Virus

Overview

The **Sophos Anti-Virus** filter uses the Sophos Anti-Virus Interface (SAVI) to screen messages for viruses. You can configure the behavior of the Sophos library using configuration options available in the **Sophos Anti-Virus** filter.

Important

Because the API Gateway does not ship with any Sophos binaries, the API Gateway must be installed on the same machine as the Sophos AV distribution. On Linux or Solaris, before starting the API Gateway, ensure that the Sophos AV `lib` directory is on your `LD_LIBRARY_PATH`. Similarly, on Windows, this directory must be on the system `PATH>` (the Sophos installation automatically puts this directory on the system path).

Prerequisites

Sophos integration requires the Sophos SAV Interface version 4.8.

API Gateway
When adding third-party binaries to the API Gateway, you must perform the following steps:

1. Add the binary files as follows:
   - Add `.jar` files to the *`InstallDir`*`/ext/lib` directory.
   - Add `.dll` files to the *`InstallDir`*`\Win32\lib` directory.
   - Add `.so` files to the *`InstallDir`*`/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
When adding third-party binaries to the Policy Studio, you must perform the following steps:

1. Add `.jar` files to the *`InstallDir`*`/plugins/thirdparty.runtime.dependencies_6.0.3` directory.
2. Restart the Policy Studio.

General Settings

Specify the following general setting:

Name:
Enter an appropriate name for this filter.

Sophos Configuration Settings

All SAVI configuration options take the form of a name-value pair. Each name is unique and its corresponding value controls specific behavior in the Sophos anti-virus library (for example, decompress `.zip` files to examine their content). You can specify these SAVI configuration settings in the **Sophos configuration settings** section:

Name:
The **Sophos Anti-Virus** filter ships with two sets of default configuration settings: one for UNIX-based platforms, and the other for Windows platforms. Select the appropriate configuration settings for your target platform from the drop-down list.

You can create a new set of configuration options by clicking the **Add** button, and adding the name-value pairs to the table provided. For convenience, you can base a new configuration set on a previously existing one, including the default Windows and UNIX sets. In this way, you can create a new configuration set that *inherits* from the default set, and then adds more configuration options.

To add a new configuration name-value pair, click the **Add** button under the table, and configure the following fields in the dialog:

Name:
Enter a name for the SAVI configuration option. This name must be available in the version of the SAVI library that is used by the API Gateway. Please refer to your SAVI documentation for a complete reference on available options.

Value:
Enter an appropriate value for the SAVI configuration option entered above. Please refer to your SAVI documentation for more information on acceptable values for this configuration option.

Type:
Select the appropriate type of this configuration option from the drop-down list. Please refer your SAVI documentation for more information on the type of the value for this configuration option.

Threatening Content

Overview

The **Threatening Content** filter can run a series of regular expressions that identify different attack signatures against request messages to check if they contain threatening content. Each expression identifies a particular attack signature, which can run against different parts of the request, including the request body, HTTP headers, and the request query string. In addition, you can configure the MIME types on which the **Threatening Content** filter operates.

The threatening content regular expressions are stored in the global **Black list** library, which is displayed under the **Libraries** node in the Policy Studio tree. By default, this library contains regular expressions to identify SQL syntax to guard against SQL injection attacks, DOCTYPE DTD references to avoid against DTD expansion attacks, Java exception stack trace information to prevent call stack information getting returned to the client, and expressions to identify other types of attack signature.

The **Threatening Content** filter is available from the **Content Filtering** category of filters. Drag and drop this filter on to the policy editor, and enter a name for the filter in the **Name** field. The next sections describe how to configure the other tabs on this filter screen.

Scanning Details

To configure the scanning details, complete the following sections:

Additional message parts to scan:
This section configures what parts of the incoming request are scanned for threatening content. By default, the **Threatening Content** filter acts on the request body. However, it can also scan the HTTP headers and the request query string for threatening content. Select the appropriate checkboxes to indicate what additional parts of the request message you want to scan.

Blacklist:
The table lists all the regular expressions that have been added to the global **Black list**. These regular expressions are used to identify threatening content. For example, there are regular expressions to match SQL syntax, ASCII control characters, and XML processing instructions, all of which can be used to attack a Web Service. For more information on how to configure these global regular expressions, see the section called “Black list and White list”.

Select the regular expressions that you want to run against incoming requests using the checkboxes in the table. You can add new expressions using the **Add** button. When adding new regular expressions on the **Add Regular Expression** dialog, the expressions are added to the global **Black list** library.

You can edit or remove existing regular expressions by selecting the expression in the tree, and selecting the **Edit** or **Delete** button.

MIME Types

The **MIME Types** tab lists the MIME types to be scanned for incoming messages. By default, all text- and XML-related types are scanned for threatening content. However, you can select any type from the list.

Similar to the way in which the **Black list** regular expressions are global, so too are the MIME types. You can add these globally by selecting the **Settings** node in the Policy Studio tree, and clicking the **MIME/DIME** tab at the bottom.

You can add new types by selecting the **Add** button and entering a type name and corresponding extension on the **Configure MIME/DIME Type** dialog. You can enter a list of extensions by separating them with spaces. You can edit or delete existing types by selecting the **Edit** and **Delete** buttons.

Throttling

Overview

The **Throttling** filter can protect a service from message flooding. You can configure the filter to only allow a certain number of messages from a specified client in a given time frame through to the protected system. If the number of messages exceeds the specified limit, the filter fails for the excess messages.

Important

The filter still succeeds for incoming messages that meet the specified constraints. For example, if the filter is configured to allow 20 messages through per second, it fails for the 21st message, but passes for the first 20 incoming messages.

The API Gateway's behavior in the case of a breach in the configured constraints is determined by the filter that is next in the failure path for the **Throttling** filter in the policy. Typically, an **Alert**, **Trace**, or **Log** filter is configured as the successor filter in the failure path in the policy.

An example use-case of this filter would be to protect a service that can only handle a maximum of 20 messages per client per second. If the filter detects a higher number of incoming requests, it blocks the messages.

General Settings

Name:
Enter an appropriate name for the filter.

Number of Messages:
If the API Gateway receives more than this number of messages during the time interval specified in the field below, the filter fails. Otherwise, the filter passes.

Time Period:
If the API Gateway receives more than the number of messages specified in the above field in the time interval specified here, the filter fails. Otherwise, the filter passes. The time period depends on the value selected below (seconds, minutes, hours, days, or weeks). For example, if you enter `10` as the **Time Period** and select `Minutes` from the **Time Period Unit** drop-down list below, the time period lasts 10 minutes.

Time Period Unit:
The unit that the time period is measured in must be selected from the following options: `Second`, `Minute`, `Hour`, `Day`, or `Week`. The value selected together with that entered above determines the time period.

Alternatively, you can specify the time period unit using a selector to represent the value of a message attribute, which is looked up and expanded to a value at runtime. Use the following syntax to specify the selector: `${`*`attribute.name`*`}`. For more details on selectors, see *Selecting Configuration Values at Runtime*. The values allowed in the message attribute are `Second`, `Minute`, `Hour`, `Day`, or `Week`.

Note

When one of `Second`, `Minute`, or `Hour` is specified, you do not need to configure the **Time Period Commences on Hour/Day** fields. The time period commences when a message is received and lasts for the time period (for example, 10 minutes). When this time period is up, the message count is reset, and the counter starts again when another message is received. The sections that follow explain how the time period is measured when the **Time Period Unit** is set to `Day` or `Week`.

Time Period Commences on Hour:

This field must be configured if you select either `Day` or `Week` as the **Time Period Unit** above. For example, if you select `Day` above and enter `00:00` in this field, this means that only the specified number of messages can be received in a one day period starting from midnight tonight until midnight the next day.

Time Period Commences on Day:
This field must only be configured if you select `Week` as the **Time Period Unit** above. For example, if you select `Week` and `00:00` in the fields above, and enter `Sunday`, this means that the time period commences next Sunday at midnight and lasts for one week exactly. The time period is reset on midnight of the next Sunday.

Cache Settings

Track per Key:
Select this setting if you wish to configure the API Gateway to keep track of request messages based on a specific key value. In other words, if more messages that match this key are received than are allowed, the filter fails.

Key Value:
You can use the **Track per Key** option to perform message filtering based on a particular key. This key can be used to look up entries in the cache selected below. Defaults to the following:

```
${http.request.clientaddr.getAddress()}
```

The key entered can also be a combination of a fixed string value and/or API Gateway message attribute selector. For example, you could use the following key to keep track of the number of times a particular URI is requested:

```
MSG_COUNT-${http.request.uri}
```

Select Cache to Use:
In cases where multiple API Gateways are deployed for load balancing purposes and you want to maintain a single count of all messages processed by all the API Gateway instances, you can configure a distributed cache to cache request messages.

For example, assume the intention is to prevent a burst of more than 50 messages per second from reaching the back-end Web Service. Also assume that a load balancer is deployed in front of two instances of the API Gateway and round-robins requests between these two instances. By caching request messages in a global distributed cache, which is inherently replicated across all API Gateway instances, the **Throttling** filter can compute the total number of messages in the distributed cache and hence, the total number of messages processed by all API Gateway instances.

The **Throttling** filter uses the pre-configured **Local maximum messages** cache by default. To configure a different cache, click the button on the right, and select from the list of currently configured caches in the tree. To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Using Multiple Throttling Filters

If you want two or more **Throttling** filters to maintain separate message counts, you must use a different key into the cache for each filter, or use different caches for each filter:

- **Using a Different Key per Filter:**
  With this approach you can use a unique **Track per key** value in each **Throttling** filter. The easiest way to do this is to prepend the default `${http.request.clientaddr.getAddress()}` selector value with the filter name, for example:

  ```
  ${filterName} ${http.request.clientaddr.getAddress()}
  ```

  This ensures that each filter maintains its own separate message count in the selected cache.
- **Using a Unique Cache per Filter:**

Alternatively, you can use a unique cache to store the message count of each **Throttling** filter. With this solution, you must configure a separate cache for each **Throttling** filter that you have configured throughout *all* policies running on the API Gateway.

Validate Message Attributes

Overview

Filters configured in a policy before the **Validate Message Attributes** filter can generate message attributes and store them in the message. The **Validate Message Attributes** filter can use regular expressions to check the values of these message attributes. This enables you to make decisions on what to do with the message (for example, if the attribute value is X, route to service X)

You can configure the following sections on the **Validate Message Attributes** screen:

- **Enter Regular Expression**:
  You can configure a list of message attributes so that each is checked against a regular expression from the global **White list** library or against a manually configured expression. This check ensures that the value of the message attribute is acceptable. For example, if you know that a message attribute must have a value of `ABCD`, a regular expression of `^ABCD$` is an exact match test.
- **Enter Threatening Content Regular Expression**:
  You can select threatening content regular expressions from the global **Black list** to run against each message attribute. These regular expressions identify common attack signatures (for example, SQL injection attacks, ASCII control characters, XML entity expansion attacks, and so on).

You can configure the global **White list** and **Black list** libraries of regular expressions under the **Libraries** node in the Policy Studio tree.

Configuring Message Attribute Regular Expressions

The **Enter Regular Expression** table displays the list of configured message attribute names together with the **White list** of regular expressions that restrict their values. For this filter to run successfully, *all* configured attribute checks must have values matching the configured regular expressions.

The **Name** column shows the name of the attribute. The **Regular Expression** column shows the name of the regular expression that the API Gateway uses to restrict the value of the named attribute. A number of common regular expressions are available from the global **White list** library.

Configuring a Regular Expression

You can configure regular expressions by selecting the **Add**, **Edit**, and **Delete** buttons. The **Configure Regular Expression** dialog enables you to add or edit regular expressions to restrict the values of message attributes. To configure a regular expression, perform the following steps:

1. Enter the name of the message attribute in the **Name** field.
2. Select whether this attribute is **Optional** or **Required** using the appropriate radio button. If it is **Required**, the attribute *must* be present in the request. If the attribute is not present, the filter fails. If it is **Optional**, the attribute does not need to be present for the filter to pass.
3. You can enter the regular expression to restrict the value of the attribute manually or select it from the global **White list** library of regular expressions in the **Expression Name** drop-down list. A number of common regular expressions are provided (for example, alphanumeric values, dates, and email addresses).
   You can use a selector representing the value of a message attribute to compare the value of a message attribute with another attribute. Enter the `$` character in the **Regular Expression** field to view a list of available attributes. At runtime, the selector is replaced by the corresponding attribute value and compared to the message attribute value that you want to check. For more details on selectors, see *Selecting Configuration Values at Runtime*.
4. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

Advanced Settings
The **Advanced** section enables you to extract a portion of the attribute value which is run against the regular expression. The extracted substring can also be Base64 decoded if necessary.

Threatening Content Regular Expressions

The regular expressions entered in this section guard against the possibility of a message attribute containing malicious content. The **Enter Threatening Content Regular Expression** table lists the **Black list** of regular expressions that are run against all message attributes.

For example, to guard against a SQL `DELETE` attack, you can write a regular expression to identify SQL syntax and add to this list. The **Threatening Content Regular Expressions** are listed in a table. *All* of these expressions are run against *all* message attributes configured in the **Regular Expression** table above. If the expression matches *any* attribute values, the filter fails.

Important

If any regular expressions are configured in the Message Attribute Regular Expressions section, these expressions are run *before* the Threatening Content Regular Expressions (TCRE) are run. For example, if you have already configured a regular expression to extract the Base64-decoded attribute value, the TCRE is run against this value instead of the attribute value stored in the message.

You can add threatening content regular expressions using the **Add** button. You can edit or remove existing expressions by selecting them in the drop-down list, and clicking the **Edit** or **Delete** button.

You can enter the regular expressions manually or select them from the global **Black list** library of threatening content regular expressions. This library is pre-populated with regular expressions that scan for common attack signatures. These include expressions to guard against common SQL injection-style attacks (for example, SQL `INSERT`, SQL `DELETE`, and so on), buffer overflow attacks (content longer than 1024 characters), and the presence of control characters in attribute values (ASCII control characters).

Enter or select an appropriate regular expression to scan all message attributes for threatening content. You can add a regular expression to the library by selecting the **Add/Edit** button. Enter a **Name** for the expression followed by the **Regular Expression**.

Validate REST Request

Overview

The **Validate REST Request** filter enables you to validate the following aspects of a REST request:

- The HTTP method used in the request.
- Each of the query string parameters against a set of restrictive conditions called a *Request Parameter Restriction*.

For example, a Request Parameter Restriction enables you to specify the expected data type of a named parameter, a regular expression for the parameter value, the minimum and maximum length of a string parameter, the minimum and maximum value of a numeric parameter, and so on.

This filter is found in the **Content Filtering** category in the Policy Studio. For details on how to create a REST request, see the *Create REST Request* filter.

General Configuration

Complete the following fields on the **Validate REST Request** screen:

Name:
Enter an appropriate name for the filter.

HTTP Method:
Enter or select the HTTP method of the incoming message (for example, `POST`, `GET`, `DELETE`, and so on). The HTTP method of the incoming request must match the method specified here.

REST Request Parameter Restrictions

Click the **Add** button to configure restrictions on the values of query string parameters. You can configure the following settings in the **REST Request Parameter Restrictions** dialog:

REST Request Parameter Details
Complete the following fields:

Description	The description entered here is displayed in the **REST Request Parameter Restriction** table on the main filter screen (for example, `Name parameter must be string no longer than 10 characters`). This field is mandatory.
Request Parameter Name	The name of the query string parameter to validate (for example, `name`). This field is mandatory.
Request Parameter Type	The data type of the query string parameter (for example, `string` or `integer`). You can enter a value or select from the drop-down list. This field is mandatory.
Fail if request parameter not found	Select this option if the specified request parameter must be present in the request query string. The filter fails if the parameter is not found.

Request Parameter Restrictions
Complete the following fields:

Min Length	Specifies the minimum number of characters or list items allowed (for example, `0`). The default value of `-1` means that this restriction is ignored.
Max Length	Specifies the exact number of characters or list items allowed (for example, `10`). The default value of `-1` means that this restriction is ignored.
Regular Expression	Specifies the exact sequence of characters that are permitted using a regular expression (for example, `[a-zA-Z\s]*`). **Note** Do not enter the `^` character at the beginning of the expression and the `$` character at the end of the expression. This filter uses the XML Schema Pattern Facet regular expression language, which implicitly anchors all regular expressions at the head and tail.
Enumeration	Specifies a list of permitted values. Click **Add** to enter an item in the list, and Click **OK**. Repeat as necessary to add multiple values.

Advanced Restrictions
Complete the following fields:

Greater than	Specifies that the value entered in the **Minimum Value** field represents an exclusive lower bound (the value must greater than this).
Greater than or Equal to	Specifies that the value entered in the **Minimum Value** field represents an inclusive lower bound (the value must greater than or equal to this).
Minimum Value	Specifies the lower bounds for numeric values (for example, the value must greater than 20).
Less than	Specifies that the value entered in the **Maximum Value** field represents an exclusive lower bound (the value must less than this).
Less than or Equal to	Specifies that the value entered in the **Maximum Value** field represents an inclusive lower bound (the value must less than or equal to this).
Maximum Value	Specifies the upper bounds for numeric values (for example, the value must less than or equal to 30).
Max Total Digits for Number	Specifies the maximum number of digits allowed for a numeric data type. The default value of `-1` means that this restriction is ignored.
Max Digits in Fraction Part of Number	Specifies the exact number of digits allowed in the fraction part of a numeric type. For example, the number 1.23 has two fraction digits (two numbers after the decimal point). The default value of `-1` means that this restriction is ig-

	nored.
Whitespace	Specifies how white space is handled (for example, line feeds, tabs, spaces, and carriage returns). You can enter one of the following values: • `Preserve` means the XML processor preserves (does not remove) any white space characters. • `Replace` means the XML processor replaces any white space characters (line feeds, tabs, and carriage returns) with spaces. • `Remove` means the XML processor removes any white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space).

Fail if unspecified request parameter found:
If a request parameter is found on the incoming query string that has not been specified in the **REST Request Parameter Restrictions** table, this filter fails. You can use this option to guard against processing a query string containing a potentially malicious request parameter (for example, `/uri?number=2&badParam=System.exit(1);`).

Validate Timestamp

Overview

You can use the **Validate Timestamp** filter to validate a timestamp that has been stored in a message attribute by a previous filter in a policy.

For example, you can extract the value of a `wsu:Created` element from a WS-Security token and store it in a created attribute using the **Retrieve from Message** filter in the **Attributes** category. You can then use the **Validate Timestamp** filter to ensure that the created timestamp is not *after* the current time.

Similarly, you can use the **Retrieve from Message** filter to extract the value of the `wsu:Expires` element and store it in a timestamp message attribute. You can use the **Validate Timestamp** filter to check that the timestamp is not *before* the current time.

This ensures that the current time is between the `Created` time and the `Expires` time. By taking into account the drift time (to resolve discrepancies between clock times on the machine that generated the timestamp, and the machine running the API Gateway), this ensures that the current time is after the `Created` time minus the drift time, and before the `Expires` time plus the drift time. The current time is within the following time frame:

```
[Created Time - Drift, Expiry Time + Drift]
```

Important

If you wish to validate the timestamp stored in a WS-Security Username Token or SAML assertion, you can use the **WS-Security Username Token Authentication**, **SAML Authentication**, **SAML Authorization**, or **SAML Attribute** filters to perform this validation. You can use the **Validate Timestamp** filter to validate non-standard timestamps, such as those not transmitted in WS-Security tokens or SAML assertions.

The **Validate Timestamp** filter does not require an entire WS-Utility Timestamp element (unlike the **Insert Timestamp** filter). Instead, this filter requires a simple date-formatted string.

Configuration

Complete the following fields to configure the API Gateway to validate a timestamp that has been stored in a message attribute:

Name:
Enter a name for the filter.

Attribute Containing Timestamp:
Enter the name of the message attribute that contains the value of the timestamp in this field. You must configure a predecessor of this filter to extract the timestamp from the message and store it in the specified attribute (for example, the **Retrieve from Message** filter in the **Attributes** filter.

Format of Timestamp:
Enter the format of the timestamp that is contained in the specified message attribute. The default date/time format is `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`, which can be altered if necessary. For more information on how to use this format, see the Javadoc for the `java.text.SimpleDateFormat` [http://java.sun.com/javase/6/docs/api/index.html] class.

Timezone:
Select the time zone to use to interpret the time stored in the message attribute selected above. The default option is GMT.

Drift (secs):
Specify the drift time to use when determining whether or not the current time falls within a certain time interval. The drift time can be used to account for differences in the clock times of the machine running the API Gateway and the machine on which the timestamp was generated.

Timestamp must be in the past:
The time in the timestamp must be *before* the time at which the server validates the timestamp. This is used for validating a timestamp that represents a `Created` time (the created time must be before the validation time).

Timestamp must be in the future:
The time in the timestamp must be *after* the time at which the server validates the timestamp. This is used for validating a timestamp that represents an `Expires` time (the expiry time must be some time in the future relative to the validation time).

WS-SecurityPolicy Layout

Overview

Web Services can use the WS-Policy specification to advertise the security requirements that clients must adhere to in order to successfully connect and send messages to the service. For example, a typical WS-Policy would mandate that the SOAP request Body be signed and encrypted (using XML Signature and XML Encryption) and that a signed WS-Utility Timestamp must be present in a WS-Security header.

To guarantee that the security tokens used to *protect* the message are added to the request in the most efficient and interoperable manner, WS-Policy uses the `<wsp:Layout>` assertion. The semantics of this assertion are implemented by this filter and are outlined in the configuration details in the next section.

Configuration

To check a SOAP message for a particular WS-Policy layout, complete the following fields:

Name:
Enter an intuitive name for this filter (for example, `Check SOAP Request for Lax Layout`).

Actor:
Enter the name of the SOAP Actor/Role where the security tokens are present.

Select Required Layout Type:
Select the required layout from the following WS-Policy options:

- **Strict:**
  Select this option to check that a SOAP message adheres to the WS-Policy strict layout rules. For more information, see the WS-Policy specification.
- **Lax:**
  Select this option if you want to ensure that the security tokens in the SOAP header have been inserted according to the Lax WS-Policy layout rules. The WS-Policy Lax rules are effectively identical to those stipulated by the SOAP Message Security specification.
- **LaxTimestampFirst:**
  This layout option ensures that the WS-Policy Lax rules have been followed, but also checks to make sure that the WS-Utility Timestamp is the first security token in the WS-Security header.
- **LaxTimestampLast:**
  This option ensures that the WS-Utility Timestamp is the last security token in the WS-Security header and that all other Lax layout rules have been followed.

WS-Security Version:
The layout rules for WS-Security versions 1.0 and 1.1 are slightly different. Select the version of the layout rules that you wish to apply to SOAP requests. For details on the differences between these versions, see the WS-Security specifications.

XML Complexity

Overview

Parsing XML documents is a notoriously processor-intensive activity. This can be exploited by hackers by sending large and complex XML messages to Web Services in a type of denial-of-service attack attempting to overload them. The **XML Complexity** filter can protect against such attacks by performing the following checks on an incoming XML message:

- Checking the total number of nodes contained in the XML message.
- Ensuring that the message does not contain deeply nested levels of XML nodes.
- Making sure that elements in the XML message can only contain a specified maximum number of child elements.
- Making sure that each element can have a maximum number of attributes.

By performing these checks, the API Gateway can protect back-end Web Services from having to process large and potentially complex XML messages.

You can also use the **XML Complexity** filter on the Web Service response to prevent a dictionary attack. For example, if the Web Service is a phone book service, a `name=*` parameter could return all entries.

Configuration

The **XML Complexity** filter should be configured as the first filter in the policy that processes the XML body. This enables this filter to block any excessively large or complex XML message *before* any other filters attempt to process the XML.

To configure the **XML Complexity** filter, complete the following fields:

Name:
Enter an appropriate name for this filter.

Maximum Total Number of Nodes:
Specify the maximum number of nodes that you want to allow in an XML message.

Note

This number does not include text nodes or comments. You can use the *Message Size* filter to stop large text nodes or comments.

Maximum Number of Levels of Descendant Nodes:
Enter the maximum number of descendant nodes that an element is allowed to have. Again, this number does not include text nodes or comments.

Maximum Number of Child Nodes per Node:
Enter the maximum number of child nodes that an element in an XML message is allowed to have.

Maximum Number of Attributes per Node:
Enter the maximum number of attributes that an element is allowed to have.

Add HTTP Header

Overview

The API Gateway can add HTTP headers to a message as it passes through a policy. It can also set a Base64-encoded value for the header. For example, you can use the **Add HTTP Header** filter to add a message ID to an HTTP header. This message ID can then be forwarded to the destination Web Service, where messages can be indexed and tracked by their IDs. In this way, you can create a complete *audit trail* of the message from the time it is received by the API Gateway, until it is processed by the back-end system.

Each message being processed by the API Gateway is assigned a unique transaction ID, which is stored in the `id` message attribute. You can use the `${id}` selector to represent the value of the unique message ID. Then at runtime, this selector is expanded to the value of the `id` message attribute. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Configuration

To configure the **Add HTTP Header** filter, complete the following fields:

Name:
Enter an appropriate name for the filter.

HTTP Header Name:
Enter the name of the HTTP header to add to the message.

HTTP Header Value:
Enter the value of the new HTTP header. You can also enter selectors to represent message attributes. At runtime, the API Gateway expands the selector to the current value of the corresponding message attribute. For example, the `${id}` selector is replaced by the value of the current message ID. Message attribute selectors have the following syntax:
`${messsage_attribute}`

Override existing header:
Select this setting to override the existing header value. This setting is selected by default.

Note

When overriding an existing header, the header can be an HTTP body related header or a general HTTP header. To override an HTTP body related header (for example, `Content-Type`), you must select the **Override existing header** and **Add header to body** settings.

Base64 Encode:
Select this setting to Base64 encode the HTTP header value. For example, you should use this if the header value is an X.509 certificate.

Add header to body:
Select this option to add the HTTP header to the message body.

Add header to HTTP headers attribute:
Select this option to add the HTTP header to the `http.headers` message attribute.

JSON Add Node

Overview

You can use this filter to add a node to a JavaScript Object Notation (JSON) document. The new node is inserted into the location specified by a JSON Path expression. JSON Path is a query language that enables you to select nodes in a JSON document.

For more details on JSON Path, see http://code.google.com/p/jsonpath.

Configuration

You can configure the following settings:

Name:
Enter a suitable name that reflects the role of this filter in the policy.

JSON Path Expression:
Enter the JSON Path expression used to add the node to the JSON document (for example, `$.store`). The Policy Studio prompts you if you enter an unsupported JSON Path expression.

Note

If this expression returns more than one node, the first node is used. If the expression returns no nodes, the filter returns false.

Node Source:
Enter the JSON node to be inserted into the message. For example, the following node source represents a new car:

```
{"make":"Ford",
        "airbags":true,
        "doors":4
        "price":1111.00
}
```

Select one of the following options for the source of the new node:

- **Add as a new item to an array**:
  If you select this option, the new JSON node is added as an item in an array.
- **Add as a new item with field name:**
  If you select this option, the new JSON node is added as a field specified in **Field Name** text box (for example, `car`).
- **Insert previously removed nodes:**
  You can configure a **JSON Remove Node** filter to remove JSON nodes from the message and store them in the `deleted.json.node.list` message attribute. You can then use the **JSON Add Node** filter to reinsert these nodes in a different location in the message, effectively moving the deleted nodes in the message. When selecting this option, you must also select **Save deleted nodes to be reinserted to new location** in the **Remove JSON Node** filter, which runs before the **Add JSON Node** filter in the policy. For more details, see the topic on *JSON Remove Node*.

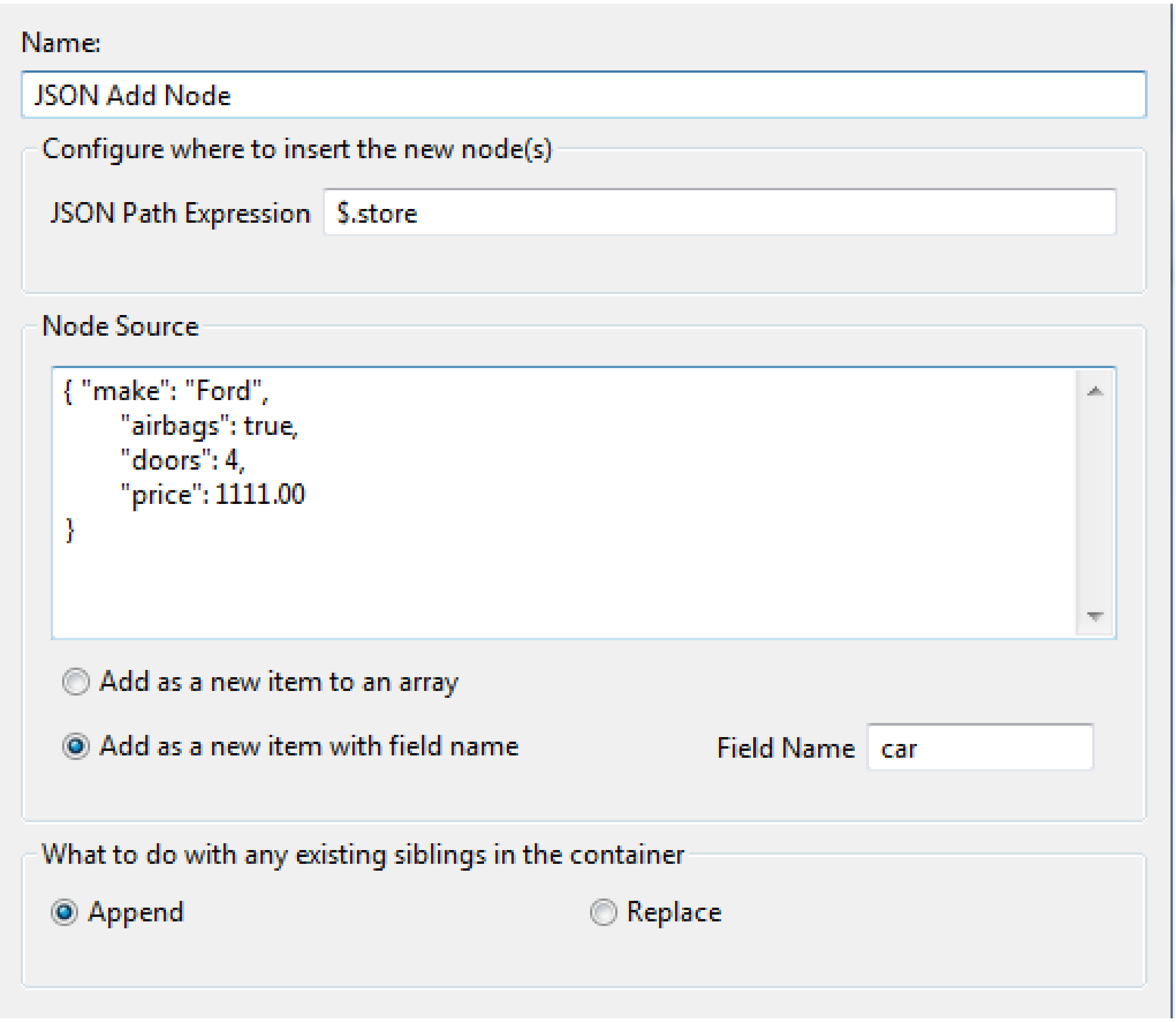
What to do with any existing siblings in the container:
Select one of the following options to determine where the new node is placed relative to the node(s) returned by the JSON Path expression:

- **Append:**
  The new node is appended as a child node of the node returned by the JSON Path expression. If there are already child nodes of the node returned by the JSON expression, the new node is added as the last child node.
- **Replace:**
  The node pointed to by the JSON expression is completely replaced by the new node.

Examples

The following are some examples of using the **JSON Add Node** filter to add and replace JSON nodes.

Adding a Node
The following example shows the settings required to add a car node to the store:



Name: JSON Add Node

Configure where to insert the new node(s)

JSON Path Expression: $.store

Node Source

```
{ "make": "Ford",
        "airbags": true,
        "doors": 4,
        "price": 1111.00
}
```

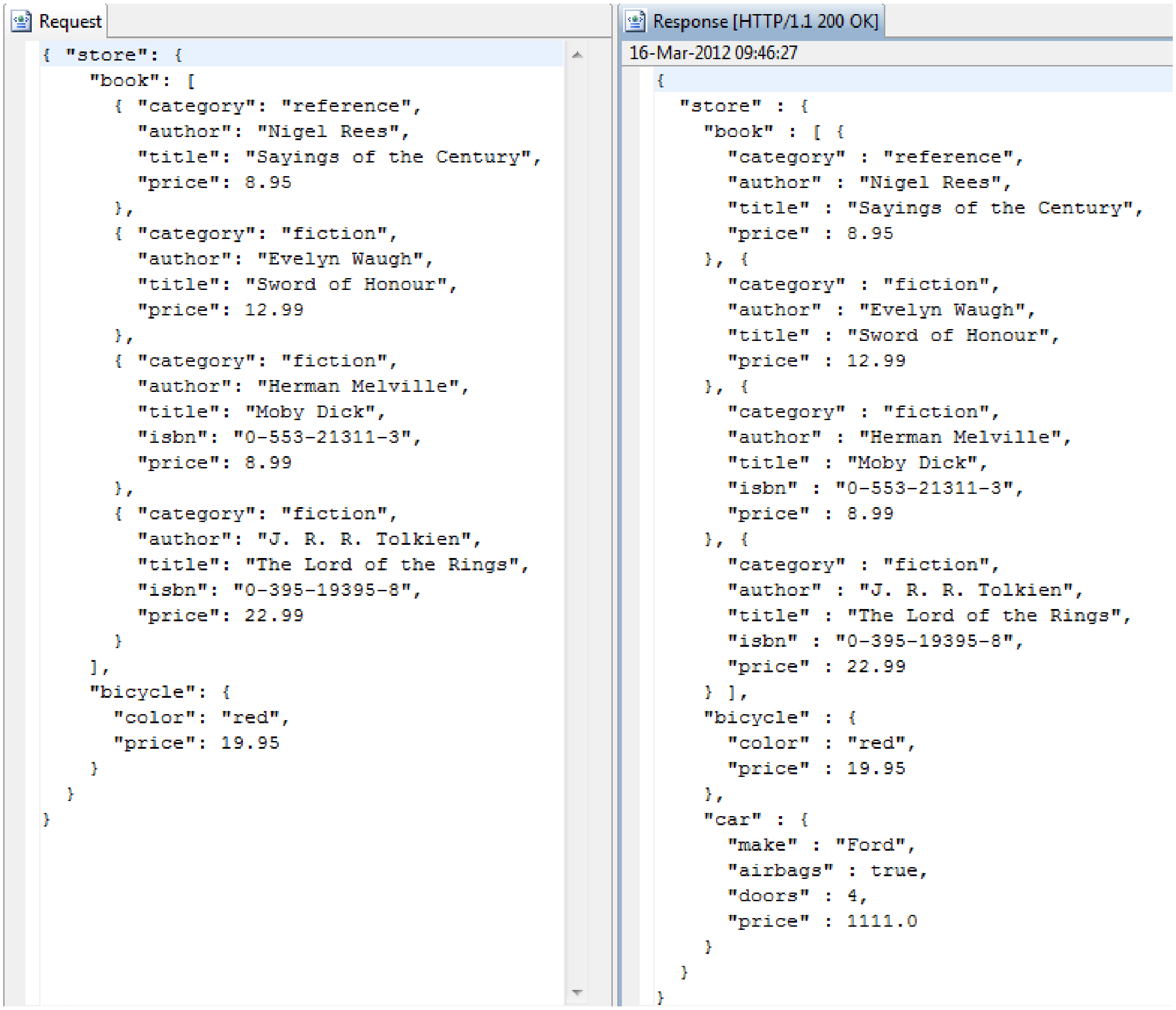
○ Add as a new item to an array

● Add as a new item with field name   Field Name: car

What to do with any existing siblings in the container

● Append   ○ Replace

The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



Request

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

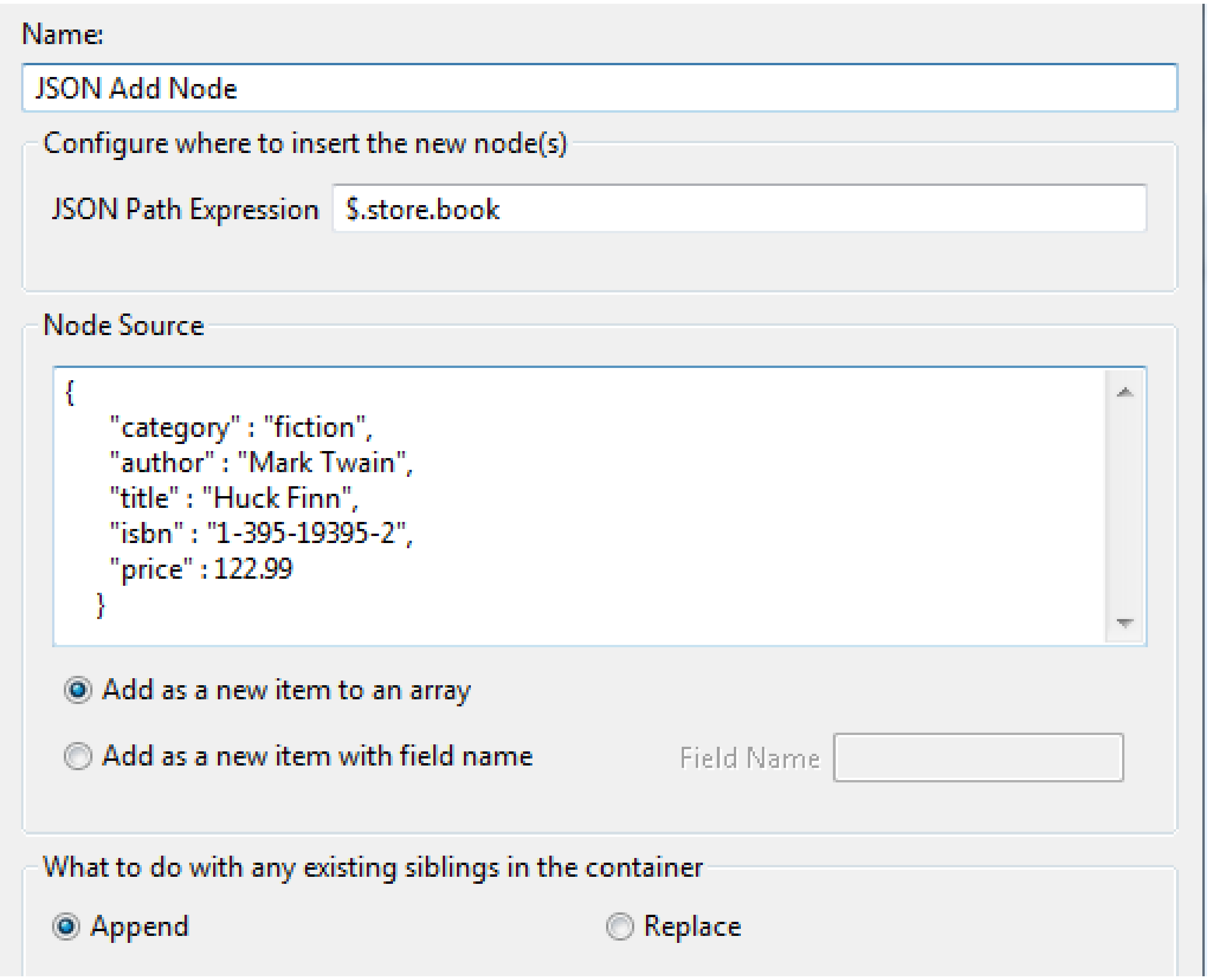
Response [HTTP/1.1 200 OK]

16-Mar-2012 09:46:27

```
{
  "store" : {
    "book" : [ {
      "category" : "reference",
      "author" : "Nigel Rees",
      "title" : "Sayings of the Century",
      "price" : 8.95
    }, {
      "category" : "fiction",
      "author" : "Evelyn Waugh",
      "title" : "Sword of Honour",
      "price" : 12.99
    }, {
      "category" : "fiction",
      "author" : "Herman Melville",
      "title" : "Moby Dick",
      "isbn" : "0-553-21311-3",
      "price" : 8.99
    }, {
      "category" : "fiction",
      "author" : "J. R. R. Tolkien",
      "title" : "The Lord of the Rings",
      "isbn" : "0-395-19395-8",
      "price" : 22.99
    } ],
    "bicycle" : {
      "color" : "red",
      "price" : 19.95
    },
    "car" : {
      "make" : "Ford",
      "airbags" : true,
      "doors" : 4,
      "price" : 1111.0
    }
  }
}
```

Adding an Item to an Array

The following example shows the settings required to add a book to an array:



Name:

JSON Add Node

Configure where to insert the new node(s)

JSON Path Expression: $.store.book

Node Source

```
{
    "category" : "fiction",
    "author" : "Mark Twain",
    "title" : "Huck Finn",
    "isbn" : "1-395-19395-2",
    "price" : 122.99
  }
```

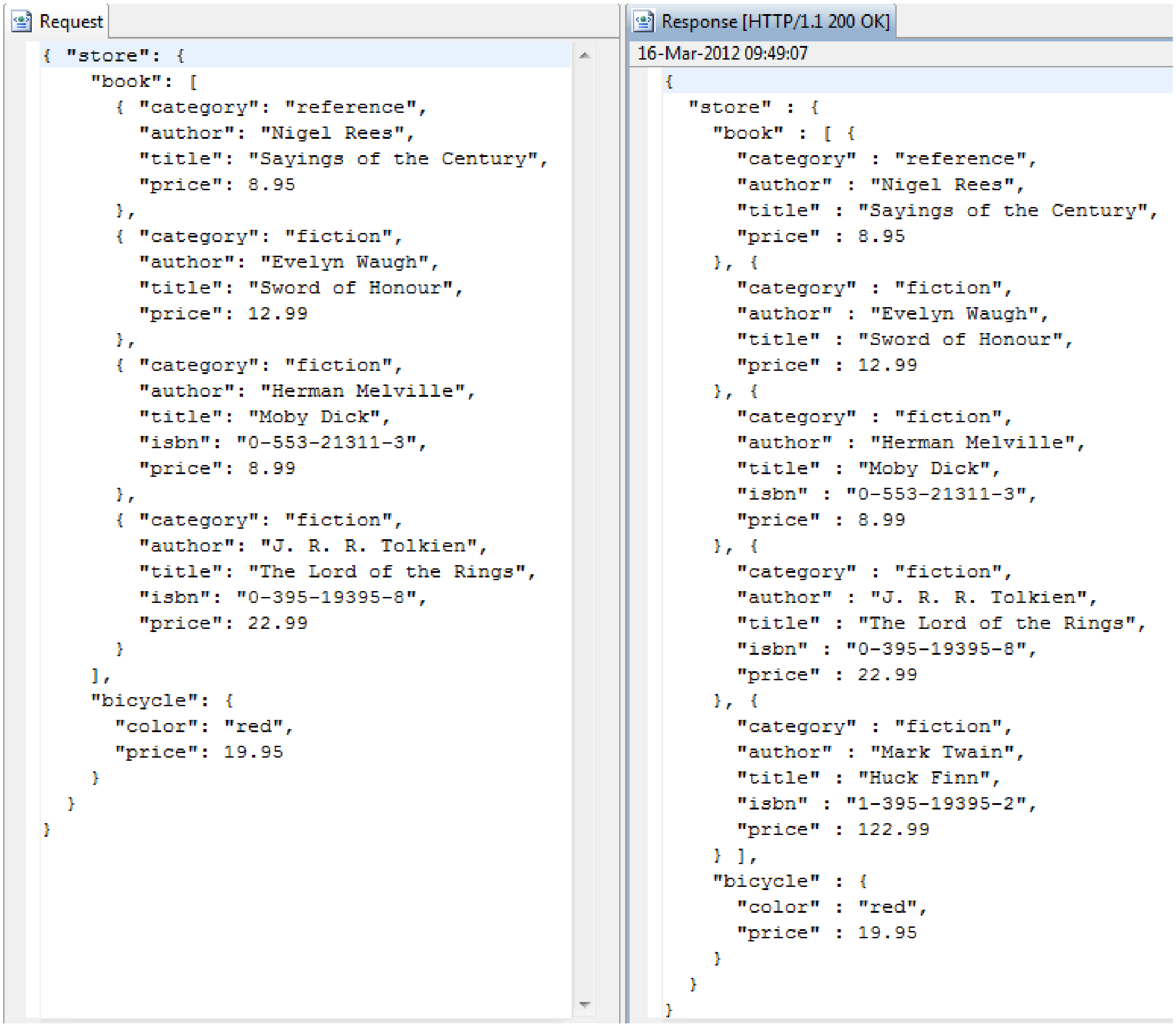
● Add as a new item to an array

○ Add as a new item with field name   Field Name:

What to do with any existing siblings in the container

● Append   ○ Replace

The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



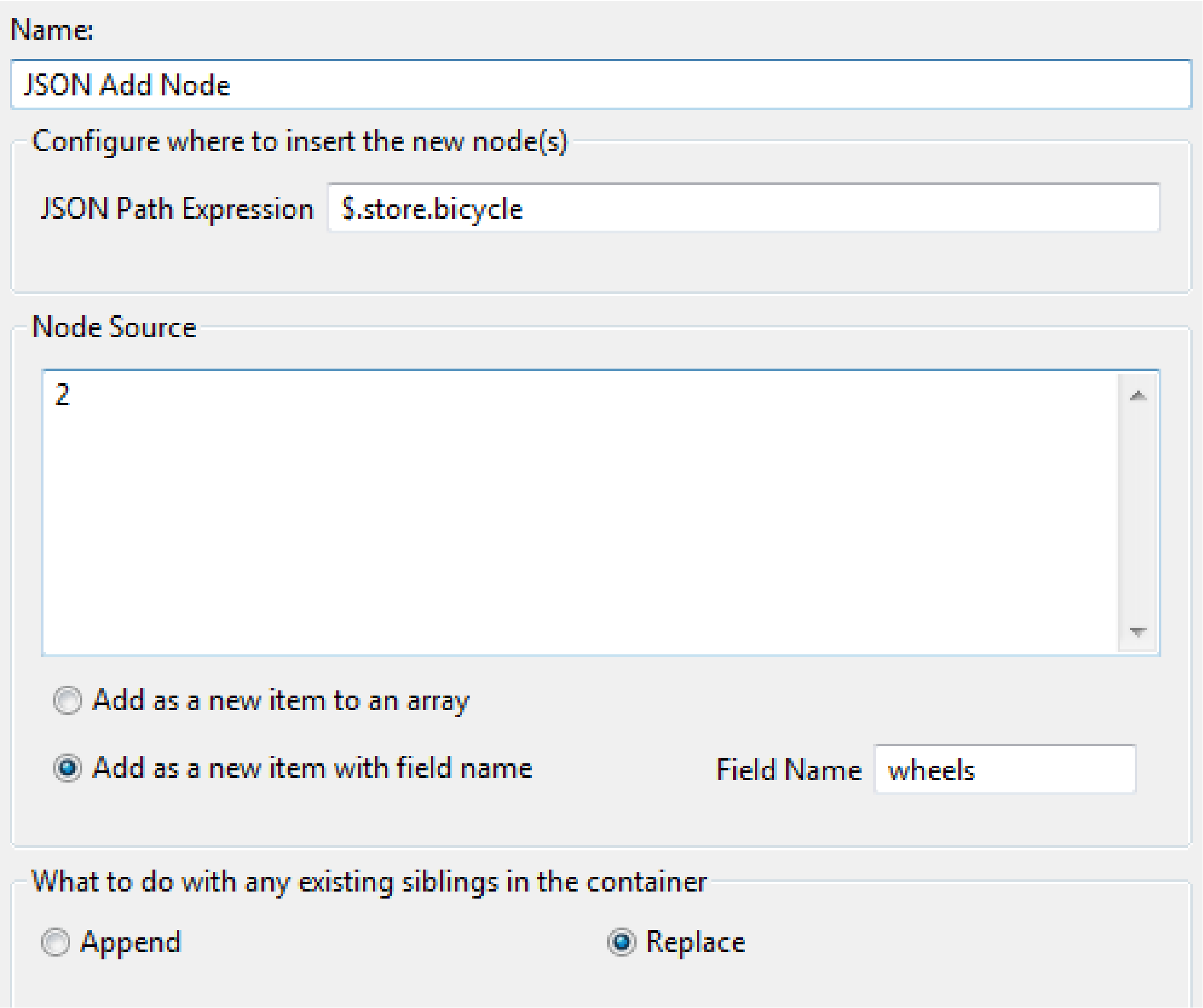
```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

16-Mar-2012 09:49:07

```
{
  "store" : {
    "book" : [ {
      "category" : "reference",
      "author" : "Nigel Rees",
      "title" : "Sayings of the Century",
      "price" : 8.95
    }, {
      "category" : "fiction",
      "author" : "Evelyn Waugh",
      "title" : "Sword of Honour",
      "price" : 12.99
    }, {
      "category" : "fiction",
      "author" : "Herman Melville",
      "title" : "Moby Dick",
      "isbn" : "0-553-21311-3",
      "price" : 8.99
    }, {
      "category" : "fiction",
      "author" : "J. R. R. Tolkien",
      "title" : "The Lord of the Rings",
      "isbn" : "0-395-19395-8",
      "price" : 22.99
    }, {
      "category" : "fiction",
      "author" : "Mark Twain",
      "title" : "Huck Finn",
      "isbn" : "1-395-19395-2",
      "price" : 122.99
    } ],
    "bicycle" : {
      "color" : "red",
      "price" : 19.95
    }
  }
}
```

Adding a Field Replacing Others

The following example shows the settings required to add a field to the bicycle, removing any other fields that may exist:



Name: JSON Add Node

Configure where to insert the new node(s)

JSON Path Expression: $.store.bicycle

Node Source

```
2
```

○ Add as a new item to an array

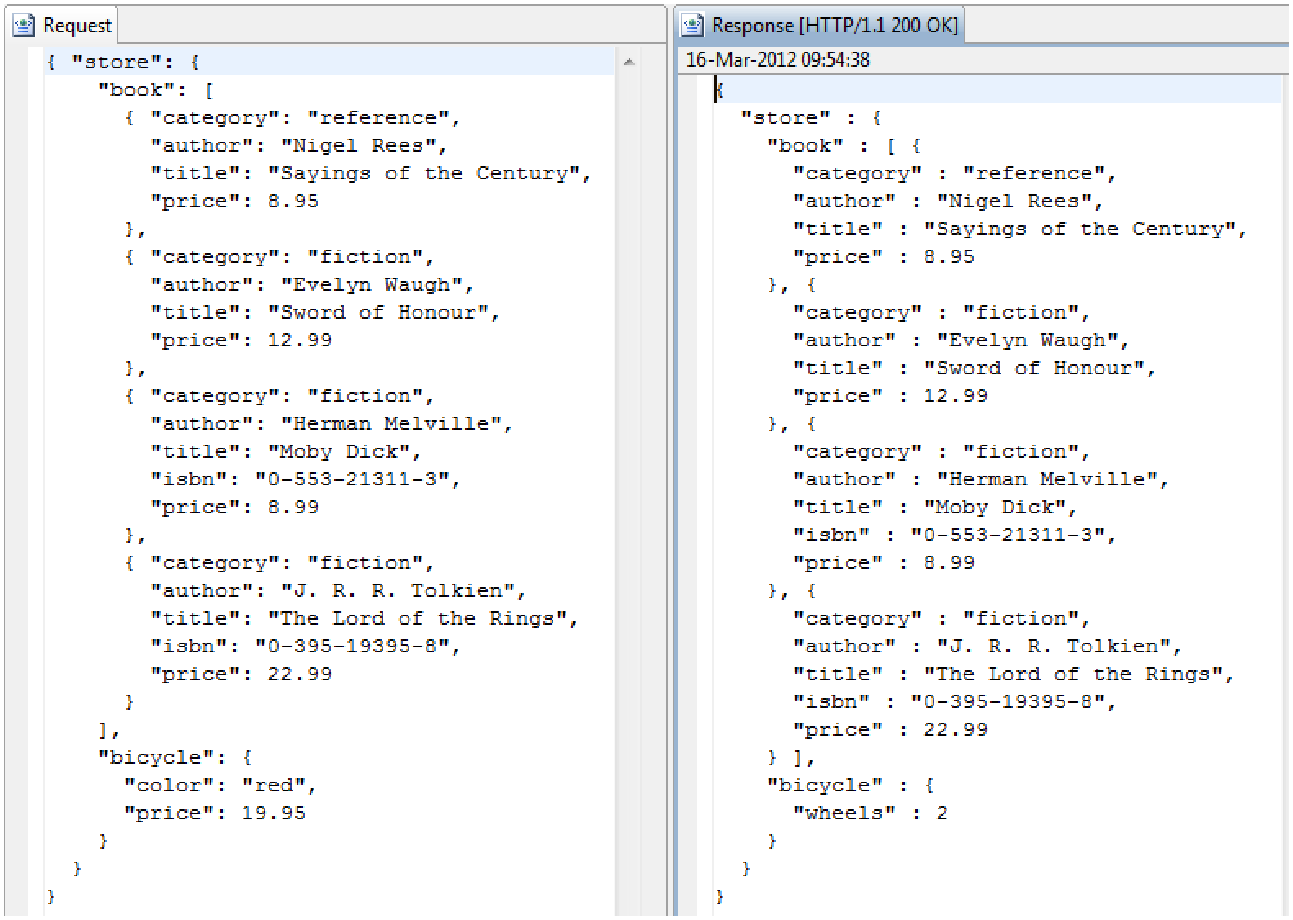
● Add as a new item with field name

Field Name: wheels

What to do with any existing siblings in the container

○ Append ● Replace

The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

```
{
  "store" : {
    "book" : [ {
      "category" : "reference",
      "author" : "Nigel Rees",
      "title" : "Sayings of the Century",
      "price" : 8.95
    }, {
      "category" : "fiction",
      "author" : "Evelyn Waugh",
      "title" : "Sword of Honour",
      "price" : 12.99
    }, {
      "category" : "fiction",
      "author" : "Herman Melville",
      "title" : "Moby Dick",
      "isbn" : "0-553-21311-3",
      "price" : 8.99
    }, {
      "category" : "fiction",
      "author" : "J. R. R. Tolkien",
      "title" : "The Lord of the Rings",
      "isbn" : "0-395-19395-8",
      "price" : 22.99
    } ],
    "bicycle" : {
      "wheels" : 2
    }
  }
}
```

Add XML Node

Overview

You can use this filter to add an XML element, attribute, text, comment, or CDATA section node to an XML message. The new node is inserted into the location specified by an XPath expression or a SOAP actor/role. XPath is a query language that enables you to select nodes in an XML document. A SOAP actor/role provides a way of identifying a particular WS-Security block in a message.

General Configuration

You can configure the following general setting:

Name:
Enter a suitable name that reflects the role of the filter. For example, if the purpose of this filter is to add an `<Address>` element to the message, it would be appropriate to name this filter **Add Address Element**.

Configure where to Insert the New Nodes

You can insert the new node into a location specified by an XPath expression or into a WS-Security header for the specified SOAP actor or role. Select one of the following options:

Insert using XPath:
Select or enter an XPath expression to specify where to insert the new node. If this expression returns more than one node, the first one is used. If the expression returns no nodes, the filter returns false. You can add, edit, or delete XPath expressions using the buttons provided.

Select one of the following options to determine where the new node is placed relative to the node(s) returned by the XPath expression.

- **Append:**
  The new node is appended as a child node of the node returned by the XPath expression. If there are already child nodes of the node returned by the XPath expression, the new node is added as the last child node.
- **Before:**
  The new node is inserted as a sibling node before the node returned by the XPath expression.
- **Replace:**
  The node pointed to by the XPath expression is completely replaced by the new node.

Insert into WS-Security element for SOAP Actor/Role:
Select or enter the name of the SOAP actor/role that specifies the WS-Security element into which the XML node is inserted. A SOAP actor/role provides a way of distinguishing a particular WS-Security block from others that may be present in the message. Actors belong to the SOAP 1.1 specification, and were replaced by roles in SOAP 1.2. For example, this setting is useful if there is no SOAP header or WS-Security element in the message because these are created when this option is specified.

Node Source

Select one of the following options for the source of the new node:

- **Create a new node:**
  If this option is selected, a new node is created and inserted into the location pointed to by the XPath expression configured above.
- **Insert previously removed nodes:**
  You can configure a **Remove XML Node** filter to remove XML nodes from the message and store them in the `de-`

`leted.node.list` message attribute. You can then use the **Add XML Node** filter to re-insert these nodes back into a different location inside the message, effectively moving the deleted nodes in the message. To use this option, select the **Save deleted nodes** option on a **Remove XML Node** filter that is configured to run before the **Add XML Node** filter in the policy.

- **Message attribute:**
  If this option is selected, a new node is created from the contents of the selected message attribute. The expected type of the message attribute is Node of List of Nodes.

Configure New Node Details

Configure the following node details:

Node Type:
Select the type of the new node from the drop-down list.

Important

- You can only append a node to a **Node Type** of `Element` or `Text`.
- If you append to a `Text` node, you must append another `Text` node.
- If you add a new `Attribute` node using the **Replace** option, it must replace an existing `Attribute` node.

Node Content:
This field contains the node to be inserted into the message. The **Node Content** field must contain valid XML when the **Node Type** is set to `Element`. You can also enter wildcards, which are populated at runtime by the API Gateway.

Attribute Node Details

The following attribute-related fields are only enabled when you select `Attribute` from the **Node Type** drop-down list:

Attribute Name:
Enter the name of the attribute in this field.

Attribute Namespace URL:
Enter the namespace of the attribute in this field.

Attribute Namespace Prefix:
Specify the prefix to use to map the element to the namespace entered above in this field. The new attribute is prefixed by this prefix.

Examples

The following are some examples of using the **Add XML Node** filter to replace and add attributes and elements.

Replacing an attribute value
To replace an attribute value, perform the following steps:

1. In the **Configure where to insert the new nodes** section, select **Insert using XPath**.
2. Select a value from the drop-down list (for example, `SOAP Header "mustUnderstand" attribute`).
3. Select the **Replace** option.
4. In the **Configure new node details** section, select `Attribute` from the a **Node Type** list.
5. Enter the **Node Content** in the text box (for example, in this case, `1` or `0`).

6. In the **Attribute Details** section, you must enter the **Attribute Name**.

Adding an attribute

To add an attribute to an element, perform the following steps:

1. In the **Configure where to insert the new nodes** section, select **Insert using XPath**.
2. Select a value from the drop-down list (for example, `SOAP Header Element`).
3. Select the **Append** option.
4. In the **Configure new node details** section, select `Attribute` from the a **Node Type** list.
5. Enter the **Node Content** in the text box (for example, in this case `1` or `0`).
6. In the **Attribute Details** section, you must enter the **Attribute Name**.

Adding an element

To add an element, perform the following steps:

1. In the **Configure where to insert the new nodes** section, select **Insert using XPath**.
2. Select a value from the drop-down list (for example, `SOAP Header Element`).
3. Select the **Append** option.
4. In the **Configure new node details** section, select `Element` from the a **Node Type** list.
5. Enter the **Node Content** in the text box (for example, in this case, the contents of the SOAP header).

Replacing an element

To replace element A with new element B, perform the following steps:

1. In the **Configure where to insert the new nodes** section, select **Insert using XPath**.
2. Select a value from the drop-down list (for example, `SOAP Method Element`).
3. Select the **Replace** option.
4. In the **Configure new node details** section, select `Element` from the a **Node Type** list.
5. Enter the **Node Content** in the text box (for example, in this case, the contents of the SOAP method).

Contivo Transformation

Overview

The **Contivo Transformation** filter uses the Liaison Contivo engine to transform messages into an alternative format. First you use the Contivo Analyst tool to define the mappings required for your transformation. For example, a specific XML element is placed into a specific field in a COBOL message. In Contivo Analyst, when your mappings are defined, you automatically generate Java code, which must be placed on the API Gateway classpath. This topic explains how to add the generated Java code to the API Gateway classpath.

Configuration

Complete the following fields to configure the **Contivo Transformation** filter:

Name:
Enter an appropriate name for this filter.

Transform Class Name:
Contivo Analsyt generates a Java class file that performs the transformation. You must place this class on the API Gateway classpath by copying the associated class file into the `INSTALL_DIR/ext/lib` directory, where `INSTALL_DIR` points to the root of your product installation. Enter the name of the class in this field.

Source FFD:
Contivo Analyst generates a Fixed File Descriptor (FFD) that describes the format of the *source* message to be transformed by the Contivo engine. Browse to the location of this file using the **Browse** button.

Target FFD:
Contivo Analyst also generates a Fixed File Descriptor (FFD) for the *target* message, which is the message generated by the Contivo engine. Browse to the location of this file using the **Browse** button.

Transformed Content Type:
Enter the Content-type of the target message used by the API Gateway when routing the output of the transformation on to the target Web Service.

Multipart Bodypart Conversion

Overview

The **Multipart Bodypart Conversion** filter is typically used to compress a MIME message before FTPing a message to an FTP server. A simple policy containing a **Multipart Bodypart Conversion** filter as a predecessor of an **FTP Upload** filter could be written to achieve this functionality.

The **Multipart Bodypart Conversion** filter outputs a `content.body` message attribute, which represents the last part processed. For example, in a three part multi-type message, the value of the `content.body` attribute is the third and last part in the series.

Configuration

The following fields must be configured on this filter screen:

Name:
Enter a suitable name for this filter.

Multipart Content Type:
Enter the MIME content type that you want to compress the multipart message to. For example, to compress a multipart message to a ZIP file, enter `multipart/x-zip` in this field.

Create Cookie

Overview

An HTTP cookie is data sent by a server in an HTTP response to a client. The client can then return an updated cookie value in subsequent requests to the server. For example, this enables the server to store user preferences, manage sessions, track browsing habits, and so on.

The **Create Cookie** filter is used to create a `Set-Cookie` header or a `Cookie` header. The `Set-Cookie` header is used when the server instructs the client to store a cookie. The `Cookie` header is used when a client sends a cookie to a server. This filter adds the appropriate HTTP cookie header to the message header, and saves the cookie as a message attribute.

For more details, see the topic on the *Get Cookie* filter.

Configuration

Configure the following fields on the **Create Cookie Filter Configuration** screen:

Filter Name:
Enter an appropriate name to display for this filter.

HTTP Header Type:
Select the HTTP cookie header type that you wish to create: **Set-Cookie (Server)** or **Cookie (Client)**. When this is set to **Set-Cookie (Server)**, all attributes from the **Cookie Details** list are used. When this is set to **Cookie (Client)**, only the **Cookie Value** attribute is used.

Cookie Details
You can configure the following settings for the cookie:

Setting	***Description***
Cookie Name	The name of the cookie.
Cookie Value	The value of the cookie.
Domain	The domain name for this cookie.
Path	The path on the server to which the browser returns this cookie.
Max age	The maximum age of the cookie in days, hours, minutes, and/or seconds.
Secure	Whether sending this cookie is restricted to a secure protocol. This setting is not selected by default, which means that it can be sent using any protocol.
HTTPOnly	Whether the browser should use cookies over HTTP only. This setting is not selected by default.

Create REST Request

Overview

Representational State Transfer (REST) is a client-server architectural style used to represent the state of application resources in distributed systems. Typically, servers expose resources using a URI, and clients access these resources using HTTP verbs such as HTTP GET, HTTP POST, HTTP DELETE, and so on.

The **Create REST Request** filter enables you to create HTTP requests to RESTful Web Services. You can also configure the query string parameters that are sent with the REST request. For example, for an HTTP GET request, the parameters are URL-encoded and appended to the request URI as follows:

```
GET /translate_a/t?client=t&sl=en&tl=ga&text=Hello
```

For an HTTP POST request, the parameters are URL-encoded and added to the request body as follows:

```
POST /webservices/tempconvert.asmx/CelsiusToFahrenheit
Host: ww.w3schools.com
Accept-charset: en
Celsius=200
```

This filter is found in the **Conversion** category in the Policy Studio. For details on how to extract REST request attributes from a message, see the *Extract REST Request Attributes* filter. For details on how to validate a REST request, see the *Validate REST Request* filter.

Configuration

Complete the following fields on the **Create REST Request** screen:

Name:
Enter an appropriate name for the filter.

HTTP Method:
Enter or select an HTTP method from the drop-down list (for example, `POST`, `GET`, `DELETE`, and so on).

REST Request Parameters:
You can add query string parameters to the REST request. These are simple name-value pairs (for example, `Name=Joe Bloggs`). To add query string parameters, click the **Add** button, and enter the name-value pair in the **Configure REST Request Parameters** dialog. Repeat to add multiple parameters.

This filter generates the `http.querystring` and `http.raw.querystring` message attributes to store the query string. For example, you can then append contents of the `http.raw.querystring` message attribute to a **Connect to URL** or **Rewrite URL** filter using a message attribute selector (for example, `${http.raw.querystring}`). For more details on selectors, see *Selecting Configuration Values at Runtime*.

Add attributes stored in attribute lookup list to REST request:
If you have populated the `attribute.lookup.list` message attribute using a previous filter in a policy, you can select this setting to include these message attributes in the serialized query string that is written to the request.

Set HTTP Verb

Overview

You can use the **Set HTTP Verb** filter to explicitly set the HTTP verb in the message that is sent from the API Gateway. By default, all messages are routed onwards using the HTTP verb that the API Gateway received in the request from the client. If the message originated from a non-HTTP client (for example, JMS), the messages are routed using the HTTP POST verb.

Configuration

Complete the following fields on the **Set HTTP Verb** filter screen:

Name:
Enter a name for the filter.

HTTP Verb:
Specify the HTTP verb to use in the message that is routed onwards.

Insert MTOM Attachment

Overview

Message Transmission Optimization Mechanism (MTOM) provides a way to send binary data to Web Services in standard SOAP messages. MTOM leverages the include mechanism defined by XML Optimized Packaging (XOP) whereby binary data can be sent as a MIME attachment (similar to SOAP with Attachments) to a SOAP message. The binary data can then be referenced in the SOAP message using the `<xop:Include>` element.

The following MTOM message contains a binary image encapsulated in a MIME part:

Important

The MIME part that contains the binary image is referenced in the SOAP request body using the `<xop:Include>` element. The `href` attribute of this element refers to the `Content-ID` HTTP header of the MIME part.

```
POST /services/uploadImages HTTP/1.1
Host: API Gateway Explorer
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
    start="<mymessage.xml@example.org>";
    start-info="text/xml"

--MIME_boundary
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>
     <xop:Include xmlns:xop='http://www.w3.org/2004/08/xop/include'
          href='cid:http://example.org/myimage.gif'/>
   </image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/gif
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/myimage.gif>

// binary octets for image

--MIME_boundary
```

When the API Gateway receives this request, the **Insert MTOM Attachment** filter can be used to read the binary data in the MIME parts pointed to by the `<xop:Include>` elements embedded in the SOAP request. The binary data is then Base64-encoded and inserted into the message in place of the `<xop:Include>` elements. The resulting message is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>/aWKKapGGyQ=</image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>
```

Configuration

Complete the following fields to configure this filter:

Name:
Enter a name for the filter.

XPath Location:
Use an XPath expression to point to the location of the `<xop:Include>` element that refers to the binary attachment. The specified XPath expression can point to multiple `<xop:Include>` elements if necessary. For example, an XPath expression of `//xop:Include` returns *all* `<xop:Include>` elements in the SOAP Envelope. For more information, see the *Configuring XPath Expressions* topic.

Remove attachments once they have been included in the message:
Select this option if you wish to remove the MIME parts that contain the actual binary content from the message after they have been inserted into the message.

JSON to XML

Overview

You can use the **JSON to XML** filter to convert a JavaScript Object Notation (JSON) document to an XML document. For details on the mapping conventions used, see: https://github.com/beckchr/staxon/wiki/Mapping-Convention

Configuration

To configure the **JSON to XML** filter, specify the following fields:

Name:
Enter a suitable name that reflects the role of the filter.

Virtual root element:
If the incoming JSON document has multiple root elements, enter a virtual root element to be added to the output XML document. This is required because multiple root elements are not valid in XML. Otherwise, the XML parser will fail. For more details, see the section called “Examples”.

Insert processing instructions into the output XML representing JSON array boundaries:
Select this option if you wish to enable round-trip conversion back to JSON. This inserts the necessary processing instructions into the output XML. This option is not selected by default. For more details, see the section called “Examples”.

Note

This option is recommended if you wish to convert back to the original JSON array structures. This information would be lost during the translation back to XML.

For more details, see the topic on *XML to JSON*.

Convert JSON object names to valid XML element names:
Select this option if you wish to convert your JSON object names to XML element names. This option is not selected by default.

Important

You should ensure that your JSON object names are also valid XML element names. If this is not possible, this option analyses each object name and automatically performs the conversion. This has a performance overhead and is not recommended if you wish to convert back to the original JSON.

Examples

This section shows examples of using **JSON to XML** filter options.

Multiple Root Elements
For example, the following incoming JSON message has multiple root elements:

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address":
     {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber":
     [
         {
           "type": "home",
           "number": "212 555-1234"
         },
         {
           "type": "fax",
           "number": "646 555-4567"
         }
     ]
 }
```

If you enter `customer` in the **Virtual root element** field, this results in the following output XML:

```
<?xml version="1.0" encoding="utf-8"?>
<customer>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <age>25</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state>NY</state>
        <postalCode>10021</postalCode>
    </address>
    <phoneNumber>
        <type>home</type>
        <number>212 555-1234</number>
    </phoneNumber>
    <phoneNumber>
        <type>fax</type>
        <number>646 555-4567</number>
    </phoneNumber>
</customer>
```

Inserting processing instructions into the output XML
For example, take the following incoming JSON message:

```
{
  "customer" : {
    "first-name" : "Jane",
    "last-name" : "Doe",
    "address" : {
      "street" : "123 A Street"
    },
    "phone-number" : [ {
      "@type" : "work",
      "$" : "555-1111"
    }, {
      "@type" : "cell",
      "$" : "555-2222"
    } ]
  }
}
```

When the **Insert processing instructions into the output XML representing JSON array boundaries** option is selected, the output XML is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<customer>
    <first-name>Jane</first-name>
    <last-name>Doe</last-name>
    <address>
        <street>123 A Street</street>
    </address>
    <?xml-multiple phone-number?>
    <phone-number type="work">555-1111</phone-number>
    <phone-number type="cell">555-2222</phone-number>
</customer>
```

Extract MTOM Attachment

Overview

Message Transmission Optimization Mechanism (MTOM) provides a way to send binary data to Web Services within standard SOAP messages. MTOM leverages the include mechanism defined by XML Optimized Packaging (XOP) whereby binary data can be sent as a MIME attachment (similar to SOAP with Attachments) to a SOAP message. The binary data can then be referenced in the SOAP message using the `<xop:Include>` element.

The following MTOM message contains a binary image that has been Base64-encoded so that it can be inserted as the contents of the `<image>` element:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>/aWKKapGGyQ=</image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>
```

When the API Gateway receives this request, the **Extract MTOM Content** filter can be used to extract the Base64-encoded content from the `<image>` element, replace it with an `<xop:Include>` element, which contains a reference to a newly created MIME part that contains the binary content. The following request shows the resulting MTOM message:

```
POST /services/uploadImages HTTP/1.1
Host: API Gateway Explorer
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
    start="<mymessage.xml@example.org>";
    start-info="text/xml"

--MIME_boundary
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <uploadGraphic xmlns="www.example.org">
   <image>
     <xop:Include xmlns:xop='http://www.w3.org/2004/08/xop/include'
          href='cid:http://example.org/myimage.gif'/>
   </image>
  </uploadGraphic>
 </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/gif
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/myimage.gif>

// binary octets for image
```

```
--MIME_boundary
```

Important

It is important to note the following:

- The Base64-encoded contents of the `<image>` element have been replaced by the `<xop:Include>` element.
- The `<xop:Include>` element points to a MIME part using the `href` attribute.
- The value of the `href` attribute corresponds to the value of the `Content-ID` HTTP header of the MIME part that contains the binary octets of the actual image file.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

XPath Location:
Use an XPath expression to locate the encoded data elements. For example, in the sample SOAP request message above, you would configure an XPath expression to point to the `<image>` element. For more information, see the *Configuring XPath Expressions* topic.

Load File

Overview

The **Load File** filter enables you to load the contents of the specified file, and set them as message content to be processed. When the contents of the file are loaded, they can be passed to the core message pipeline for processing by the appropriate message filters. For example, the **Load File** filter might be used in cases where an external application drops XML files on to the file system to be validated, modified, and potentially routed on over HTTP, JMS, or simply stored to a directory where the application can access them again.

For example, this sort of protocol mediation can be useful when integrating legacy systems. Instead of making drastic changes to the legacy system by adding an HTTP engine, the API Gateway can load files from the file system, and route them on over HTTP to another back-end system. The added benefit is that messages are exposed to the full compliment of message processing filters available in the API Gateway. This ensures that only properly validated messages are routed on to the target system.

Configuration

To configure the **Load File** filter, specify the following fields:

Name	Name of the filter to be displayed in a policy. Defaults to **Load File**.
File	Specify the path to the file that the content is loaded from (for example, `c:/workspace/example_file.xml`).
File contents	Choose one of the following options: • **Raw file** • **HTTP message including HTTP headers** This enables you to load the contents of a raw file, or to load archived HTTP messages from disk.
Output directory	Specify the directory to write the output to at the end of processing (for example, `c:/my_app/output_dir`).
Finishing processing	Specify the action to take when processing is complete: • **Do nothing** • **Delete loaded file** • **Move loaded file to following directory** If you select **Move loaded file to following directory**, the **Directory** field is enabled to allow you to specify the directory that the file is moved to. The default is **Do nothing**.

Remove Attachments

Overview

This filter can be used to remove ***all*** attachments from either a request or a response message, depending on where the filter is placed in the policy.

Configuration

Enter a name for this filter in the **Name** field.

Remove HTTP Header

Overview

The API Gateway can strip a named HTTP header from the message as it passes through a policy. This is especially useful in cases where end-user credentials are passed to the API Gateway in an HTTP header. After processing the credentials, you can use the **Remove HTTP Header** filter to strip the header from the message to ensure that it is not forwarded on to the destination Web Service.

Configuration

To configure the **Remove HTTP Header** filter, perform the following steps:

1. Enter an appropriate name for this filter in the **Name** field.
2. Specify name of the HTTP header to remove in the **HTTP Header Name** field.
3. Select the **Fail if header is not present** checkbox to configure the API Gateway to abort the filter if the message does not contain the named HTTP header. Headers can be added to the message using the *Add HTTP Header* conversion filter.

JSON Remove Node

Overview

You can use this filter to remove a JSON node from a JSON message. You can specify the node to remove using a JSON Path expression. The JSON Path query language enables you to select nodes in a JSON document.

For more details on JSON Path, see http://code.google.com/p/jsonpath.

Configuration

To configure this filter, specify the following fields:

Name:
Enter a suitable name that reflects the role of the filter.

JSON Path Expression:
Enter a JSON Path expression to specify the node to remove (for example, `$.store.bicycle`). The Policy Studio prompts if you enter an unsupported JSON Path expression.

Note

If the specified expression returns more than one node, all returned nodes are removed.

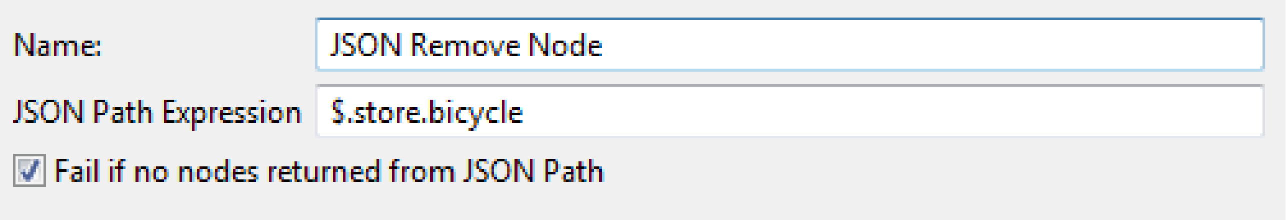
Fail if no nodes returned from JSON Path:
When this option is selected, and the JSON Path expression returns no nodes, the filter returns false. If this option is *not* selected, and the JSON Path returns no nodes, the filter returns true, and no nodes are removed. This option is not selected by default.

Save deleted nodes to be reinserted to new location:
Select this option if you want to move JSON nodes from one location in the message to another. The deleted nodes are stored in the `deleted.json.node.list` message attribute. You can then use the **JSON Add Node** filter to insert the deleted nodes into a different location in the message. For more details, see the topic on *JSON Add Node*.

Examples

The following are some examples of using the **JSON Remove Node** filter.

Removing a Node
The following example shows removing a bicycle from the store:

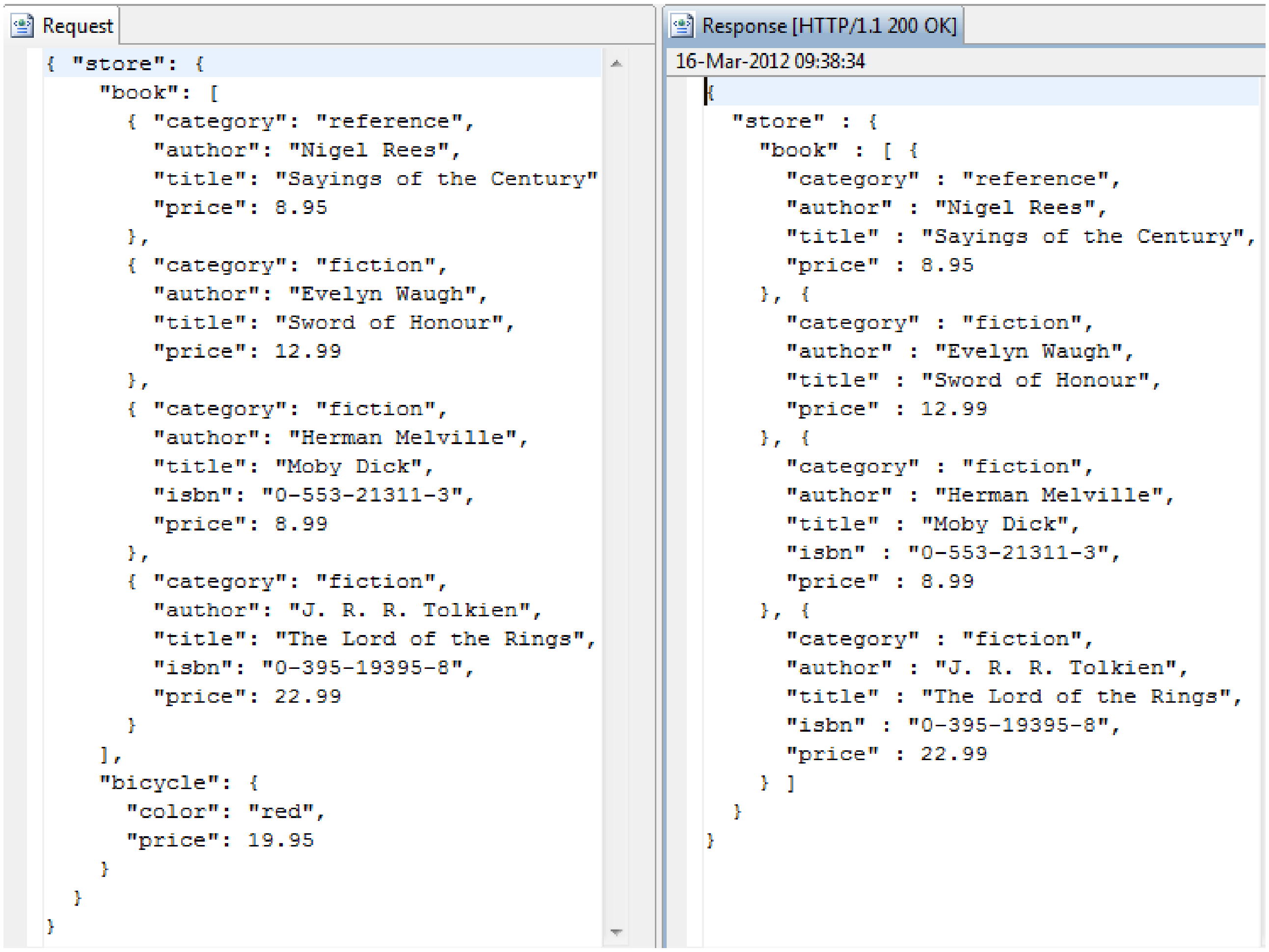


Name: JSON Remove Node

JSON Path Expression $.store.bicycle

☑ Fail if no nodes returned from JSON Path

The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



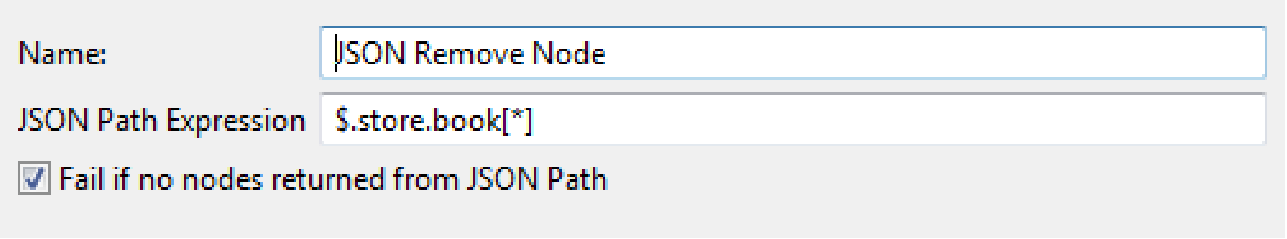
```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century"
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

16-Mar-2012 09:38:34

```
{
  "store" : {
    "book" : [ {
      "category" : "reference",
      "author" : "Nigel Rees",
      "title" : "Sayings of the Century",
      "price" : 8.95
    }, {
      "category" : "fiction",
      "author" : "Evelyn Waugh",
      "title" : "Sword of Honour",
      "price" : 12.99
    }, {
      "category" : "fiction",
      "author" : "Herman Melville",
      "title" : "Moby Dick",
      "isbn" : "0-553-21311-3",
      "price" : 8.99
    }, {
      "category" : "fiction",
      "author" : "J. R. R. Tolkien",
      "title" : "The Lord of the Rings",
      "isbn" : "0-395-19395-8",
      "price" : 22.99
    } ]
  }
}
```

Removing all Items in an Array

The following example shows removing all books in an array:

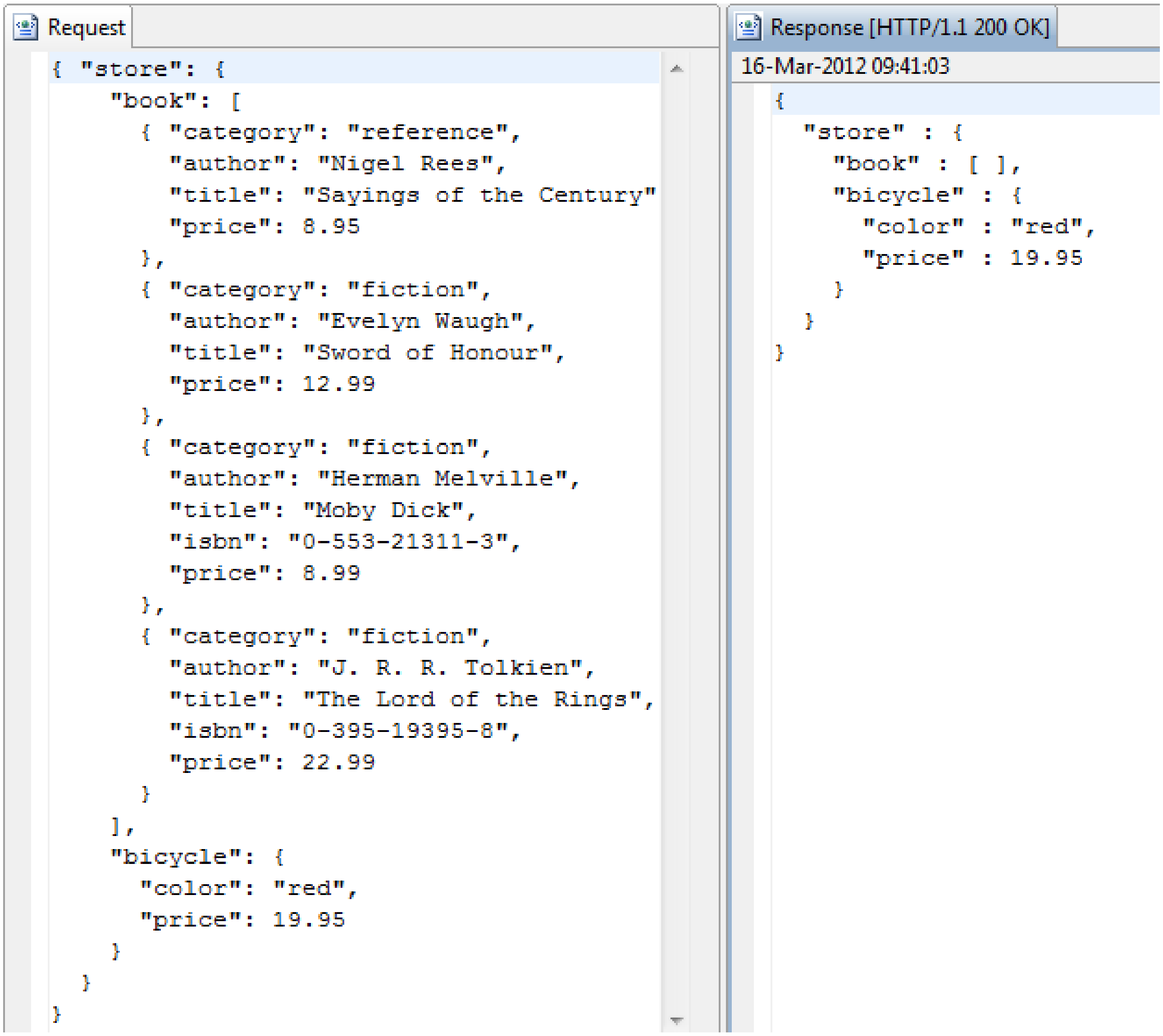


Name: JSON Remove Node

JSON Path Expression $.store.book[*]

☑ Fail if no nodes returned from JSON Path

The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century"
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

16-Mar-2012 09:41:03

```
{
  "store" : {
    "book" : [ ],
    "bicycle" : {
      "color" : "red",
      "price" : 19.95
    }
  }
}
```

Remove XML Node

Overview

You can use this filter to remove an XML element, attribute, text, or comment node from an XML message. You can specify the node to remove using an XPath expression. The XPath query language enables you to select nodes in an XML document.

Configuration

To configure this filter, specify the following fields:

Name:
Enter a suitable name that reflects the role of the filter. For example, if the purpose of this filter is to remove an `<ID>` element from the message, it would be appropriate to name this filter **Remove ID Element**.

XPath Location:
Specify an XPath expression to indicate the node to remove. When the expression is configured correctly, you can remove an element, attribute, text, or comment node. If this expression returns more than one node, all returned nodes are removed.

You can select XPath expressions from the drop-down list, and edit or add expressions by clicking the relevant button. The following are some example expressions:

Name	*XPath Expression*	*Prefix*	*URI*
The First WSSE Security element	`//wsse:Security[1]`	`wsse`	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`
Text Nodes in SOAP Body	`/soap:Envelope/soap:Body/text()`	`soap`	`http://schemas.xmlsoap.org/soap/envelope/`

Fail if no nodes returned from XPath:
If this option is selected, and the XPath expression returns no nodes, the filter returns false. If this option is *not* selected, and the XPath returns no nodes, the filter returns true, and no nodes are removed.

Save deleted nodes to be re-inserted to new location:
You can use this option in cases where you want to move XML nodes from one location in the message to another. By selecting this option, the deleted nodes are stored in the `deleted.node.list` message attribute. You can then use the **Add XML Node** filter to insert the deleted nodes back into a different location in the message. For more details, see the *Add XML Node* filter.

Restore Message

Overview

You can use this filter to restore message content at runtime using a specified selector expression. You can restore the contents of a request message or a response message, depending on where the filter is placed in the policy.

For example, you could use this filter to restore original message content if you needed to manipulate the message for authentication or authorization. Typically, this filter is used with the **Store Message** filter, which is first used to store the original message content. For more details, see *Store Message*.

Configuration

Name:
Enter a suitable name for this filter.

Selector Expression to retrieve message:
Enter the selector expression used to restore the message content. Defaults to `${store.content.body}`). For more details on selector expressions, see *Selecting Configuration Values at Runtime*.

Store Message

Overview

You can use this filter to store message content in a specified message attribute. You can store the contents of a request message or a response message, depending on where the filter is placed in the policy.

For example, you could use this filter to store the original message content for reuse later if you need to manipulate the message for authentication or authorization. Typically, this filter is used with the **Restore Message** filter, which is then used to restore the original message content. For more details, see *Restore Message*.

Configuration

Name:
Enter a suitable name for this filter.

Attribute to store message:
Enter the name of the message attribute used to store the message content. Defaults to `store.content.body`.

Set Message

Overview

The **Set Message** filter replaces the body of the message. The replacement data can be plain text, HTML, XML, or any other text-based markup.

You can use selectors representing the values of message attributes in the replacement text to insert message-specific data into the message body. For example, you can insert the authenticated user's ID into a `<Username>` element by using a `${authentication.subject.id}` selector as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <Username>${authentication.subject.id}</Username>
 </soap:Header>
 <soap:Body>
  <getQuote xmlns="oracle.com">
   <ticker>ORM.L</ticker>
  </getQuote>
 </soap:Body>
</soap:Envelope>
```

Assuming the `oracle` user authenticated successfully to the API Gateway, the message body is set as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <Username>oracle</Username>
 </soap:Header>
 <soap:Body>
  <getQuote xmlns="oracle.com">
   <ticker>ORM.L</ticker>
  </getQuote>
 </soap:Body>
</soap:Envelope>
```

For more details on selectors, see *Selecting Configuration Values at Runtime*.

You can also use the **Set Message** filter to customize SOAP faults that are returned to clients in the case of a failure or exception in the policy. For a detailed explanation of how to use this filter to customize SOAP faults, see the *SOAP Fault* topic.

Configuration

Perform the following steps to configure the **Set Message** filter:

1. Enter a name for this filter in the **Name** field.
2. Specify the content type of the new message body in the **Content-type** field. For example, if the new message body is HTML markup, enter `text/html` in the **Content-Type** field.
3. Enter the new message body in the **Message Body** text area. You can use selectors, as shown in the example above, to ensure that current message attribute values are inserted into the message body at the appropriate places. You can also load the message contents from a file by browsing to the location of the file using the **Browse** button.

XSLT Transformation

Overview

Extensible Stylesheet Language Transformations (XSLT) is a declarative, XML-based language used to transform XML documents into other XML documents. An XSL stylesheet is used to transform an XML document into another document type. The stylesheet defines how elements in the XML source document should appear in the resulting XML document.

The API Gateway can convert XML data to other data formats using XSL files. For example, an incoming XML message adhering to a specific XML schema can be converted to an XML message adhering to a different schema before it is sent to the destination Web Service.

This type of conversion is especially valuable in the Web Services arena, where a Web Service might receive SOAP requests from various types of clients, such as browsers, applications, and mobile phones. Each client might send up a different type of SOAP request to the Web Service. Using stylesheets, the API Gateway can then convert each type of request to the same format. The requests can then be processed in the same fashion.

Enter an appropriate name for this filter in the **Name** field, and configure the following tabs.

Stylesheet Location

Select an XSL stylesheet from the **Stylesheet Location** drop-down list, which is populated with the contents of the Stylesheet Library. You can import a new stylesheet into the library by clicking the **View/Import** button, and the **Add** button on the **Stylesheet Library** dialog.

You can modify existing stylesheets in the **XSLT Contents** text area, and then update them in the API Gateway configuration by clicking the **Update** button.

Stylesheet Parameters

You can pass parameters to an XSL stylesheet using specified values in `<xsl:param>` elements. These values are then used in the templates defined throughout the stylesheet.

Using the **XSLT Transformation** filter, you can pass the values of message attributes to the configured stylesheet. For example, you can take the value of the `authentication.subject.id` message attribute, pass it to the configured XSL stylesheet, and then output this value to the result produced by the conversion.

To use this feature, select the **Use Message Attributes as Stylesheet Parameters** checkbox, and specify the message attribute to pass to the stylesheet by clicking the **Add** button.

The following example from an XSL stylesheet that uses parameters shows how to configure this:

```
<xsl:param name="authentication.subject.id"/>
<xsl:param name="authentication.issuer.id"/>
```

To pass the corresponding message attribute values to the stylesheet, you must add the `authentication.subject.id` and `authentication.issuer.id` message attributes to the **Message Attributes to use** table.

Important

The name of the specified parameter must be a valid API Gateway message attribute name, and there *must* be an equivalent parameter name in the stylesheet.

Advanced

Complete the following fields on this tab:

Provider Class Name:
Enter the fully qualified name of the XSLT provider class of the XSLT library that you want to use. This class *must* be added to the API Gateway's classpath. If this field is blank, the default provider is used.

The simplest way to add a provider class to the API Gateway's classpath is to drop the required JAR file into the `PRODUCT_HOME/ext/lib` directory, where `PRODUCT_HOME` refers to the root of your API Gateway installation.

Result will be XML:
You can convert an incoming XML message to other data formats. Select this option if the result of the XSLT conversion is always XML. If not, the content-type of the result document depends on the output method of the XSLT stylesheet. For example, if the stylesheet specifies an output method of HTML (`<xsl:output method="html">`), this field should be left blank so that the API Gateway can forward on the HTML output document to the target Web Service.

Do not change the content type header:
You can select whether to change the HTTP `Content-Type` header in this XSLT transformation. This setting is selected by default, so the content type is preserved.

XML to JSON

Overview

You can use the **XML to JSON** filter to convert an XML document to a JavaScript Object Notation (JSON) document. For details on the mapping conventions used, see:https://github.com/beckchr/staxon/wiki/Mapping-Convention

Configuration

To configure the **XML to JSON** filter, specify the following fields:

Name:
Enter a suitable name to reflect the role of this filter.

Automatically insert JSON array boundaries:
Select this option to attempt to automatically reconstruct JSON arrays from the incoming XML document. This option is selected by default.

Note

If the incoming XML document includes the `<?xml multiple>` processing instruction, the JSON array is reconstructed regardless of this option setting. If the XML document does not contain `<?xml multiple>`, and this option is selected, the filter makes an attempt at guessing what should be part of the array by examining the element names.

For more details and example `<?xml multiple>` processing instructions, see the topic on *JSON to XML*.

Generate Key

Overview

The **Generate Key** filter enables you to generate an asymmetric key pair, or a symmetric key. The generated keys are placed in message attributes, which are then available to be consumed by other filters.

An example use case for this filter is to use it in conjunction with the **Security Token Service Client** filter. For example, you wish to request a SAML token with a symmetric proof-of-possession key from an STS. You need to provide the key material to the STS as a binary secret, which is the private key of an asymmetric key pair. You can use an asymmetric private key generated on-the-fly instead of from the Certificate Store with an associated certificate. You must configure the **Generate Key** filter in a **Security Token Service Client** filter policy that runs before the WS-Trust request is created. You can then configure the **Security Token Service Client** filter to consume the generated asymmetric private key. For more details, see the *Security Token Service Client* topic.

Note

An asymmetric key pair generated by the **Generate Key** filter can also be used by the **Security Token Service Client** filter when a proof-of-possession key of type `PublicKey` is requested. The generated public key can be used as the `UseKey` in the request to the STS.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

Key Type:
Select the key type from the drop-down list. Defaults to `RSA Asymmetric Key Pair`.

Key Size:
Enter the key size in bits. Defaults to `2048` bits.

PGP Decrypt

Overview

You can use the **PGP Decrypt** filter to decrypt a Pretty Good Privacy (PGP) encrypted message. This filter decrypts an incoming message using the specified PGP private key, and creates a new message body using the specified content type. The decrypted message can be processed by the API Gateway, and then encrypted again using the **PGP Encrypt** filter.

An example use case for this filter would be when files are sent to the API Gateway over Secure Shell File Transfer Protocol (SFTP) in PGP encrypted format. The API Gateway can use the **PGP Decrypt** filter to decrypt the message, and then use Threat Detection filters to perform virus scanning. The clean files can be PGP encrypted again using the **PGP Encrypt** filter before being sent over SFTP to their target destination. For more details, see the *PGP Encrypt* filter.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for this filter.

PGP Private Key to be retrieved from one of the following locations:
Select one of the following options:

- **Use the following private key from the PGP Key Pair list**
  Click the browse button on the right, and select a PGP key pair configured in the Certificate Store. For details on configuring PGP key pairs, see the topic on *Certificates and Keys*.
- **Look up the private key using the following alias**
  Enter the alias name of the PGP private key used in the Certificate Store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression that specifies the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **The following message attribute will contain the private key**
  Enter a selector expression that specifies the name of the message attribute that contains the private key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.private.key}`).

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Content type:
Enter the `Content-Type` of the unencrypted message data. Defaults to `application/octet-stream`.

PGP Encrypt

Overview

You can use the **PGP Encryption** filter to generate a Pretty Good Privacy (PGP) encrypted message. This filter enables you to configure the PGP public key used when encrypting the message. You can also configure advanced options such as whether the message outputs ASCII armor, or whether it uses a symmetrically encrypted integrity protected data packet to protect against modification attacks.

For example, using the default options, the **PGP Encryption** filter creates a PGP encrypted message such as the following:

```
-----BEGIN PGP MESSAGE-----
Version: BCPG v1.46

hQIOA3ePizxHLIA8EAgAmVNAgJO7TXI9vWCJHZS27r4FIfZIYWNc0+MiQ3H+LZrW
29Wageetg5N7cFAbRpG28iKYSE5O0uFMThuWuhnMZ/GtRwMogiRsNyBY0Cq0LKaG
7oIbkjWE1BHdWXQLWW44zYl8ekTWJ4ZPNCemTtHyULB9QwuWx5b6QfAyh1jvFrSN
ub9mQzU8caY7xQrVgWii1tBFOzTcGw6/Vb7AtMZfwGGjqmzYLT5pLozWUKB0gZe1
/7wpWDsHsn+53lrRXdoqwvAhY2AnOLPyrrVsykXS38YtIh9N5D+uCCvgmICej9Ok
iieh1hgGnuzw3VdQ6n3TS0t/Xk3shB95I3IkXU1l4ggAjPSnf9qEuN2u8dsRooNR
J6nYWs/OGwBSj0/MtssoRAcEVYu93tITUXqybduq8CATHGD8at4WRiTLOcndgJTp
0aUU+aOi3l3SsnrlpPSKIu18K9AqFCE+lafdXKlqU2OaGMBbsU22Vy6SkDgSXuGg
EOG0KYHRdrAntHJiO3qrJRTd8BDrPc5PZWUwDfCUWuQRMJiJVp0bxrK8Qzz/Ni7T
XgRSL1cYzAQRsQAbne69On+5n+NWO1Qcx9SnSimBtPOQXJfff+a+Wb45ABj5TdYr
4PCd2OJ0uOapSWfiSA5mZ1sB9hEAR0FidXs1iAploun1qggNYZK94BGGXblnTCzR
ccnARKqWmWanr1VVnp6fs9WI6I3zkiCGTJlQMNa0UKZdEPe1wJb7NHgJFMKrwN1X
3rSVyUWiovnYYMlDGBHG/RWGsTSd0LT7VugtIByefCI2G7WevgLUJbOq+U/0Sh6A
82oMNuWbXbDTp3pfZae/SHqOyEdDp5zsGqZ/F4M7CQFx63XCIBsFA6JRj6GdYqYf
dewuej3WJtRDdHmikjb3o7Utl8fFhKjA9GdEZueG9ls+XcAx21iBT656HRof8wio
oSca8ui3SYbhZ+0uzwImDJ0054P3Xr24+iwI4vlKjiQNY23GjXsVa2rQn6VHT60o
CYo08tDYBH4gyetLAqczCVyh6sff9SqX
=qkB0
-----END PGP MESSAGE-----
```

For an example use case, see the *PGP Decrypt* filter.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

PGP Public Key to be retrieved from one of the following locations:
Select one of the following options:

- **Use the following public key from the PGP Key Pair list**
  Click the browse button on the right, and select a PGP key pair configured in the Certificate Store. For details on configuring PGP key pairs, see the topic on *Certificates and Keys*.
- **Look up the public key using the following alias**
  Enter the alias name of the PGP public key used in the Certificate Store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression that specifies the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **The following message attribute will contain the public key**
  Enter a selector expression that specifies the name of the message attribute that contains the public key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.public.key}`).

For more details on selectors, see *Selecting Configuration Values at Runtime*.

ASCII Armor Output:
Select whether to output the binary message data as ASCII Armor. ASCII Armor is a special text format used by PGP to convert binary data into printable ASCII text. ASCII Armored data is especially suitable for use in email messages, and is also known as Radix-64 encoding. This option is selected by default.

Symmetric Encrypted Integrity Protected Data Packet:
Select whether the message uses a Symmetrically Encrypted Integrity Protected Data packet. This is a variant of the Symmetrically Encrypted Data packet, and is used detect modifications to the encrypted data. This option is not selected by default.

SMIME Decryption

Overview

The **SMIME Decryption** filter can be used to decrypt an encrypted Secure/Multipurpose Internet Mail Extensions (SMIME) message.

Configuration

Complete the following fields to configure this filter:

Name:
Enter a name for the filter in the **Name** field.

Use Certificate to Decrypt:
Check the box next to the certificate that you want to use to decrypt the encrypted PKCS#7 message with. The private key associated with this certificate will be used to actually decrypt the message.

SMIME Encryption

Overview

You can use the **SMIME Encryption** filter to generate an encrypted Secure/Multipurpose Internet Mail Extensions (SMIME) message. This filter enables you to configure the certificates of the recipients of the encrypted message. You can also configure advanced options such as ciphers and Base64 encoding.

General Configuration

Complete the following general field:

Name:
Enter an appropriate name for the filter.

Recipients

The **Recipients** tab enables you to configure the certificates of the recipients of the encrypted SMIME message. Select one of the following options:

Use the following certificates:
This is the default option. Select the certificates of the recipients of the encrypted message. The public keys associated with these certificates are used to encrypt the data so that it can only be decrypted using the associated private keys.

Certificate in attribute:
Alternatively, enter the message attribute that contains the certificate of the recipients of the encrypted message. Defaults to the `certificate` message attribute.

Advanced

The **Advanced** tab includes the following settings:

Cipher:
Enter the cipher that you want to use to encrypt the message data. Defaults to the `DES-EDE3-CBC` cipher.

Content-Type:
Enter the `Content-Type` of the message data. Defaults to `application/pkcs7-mime`.

Base64 encode:
Select whether to Base64 encode the message data. This option is not selected by default.

XML-Decryption

Overview

The **XML Decryption** filter is responsible for decrypting data in XML messages based on the settings configured in the **XML-Decryption Settings** filter.

The **XML-Decryption Settings** filter generates the `decryption.properties` message attribute based on configuration settings. The **XML-Decryption** filter uses these properties to perform the decryption of the data.

Configuration

Enter an appropriate name for the filter in the **Name** field.

Auto-generation using the XML Decryption Wizard

Because the **XML Decryption** filter must always be paired with an **XML Decryption Settings** filter, the Policy Studio provides a wizard that can generate both of these filters at the same time. To use the wizard, right-click a policy node under the **Policies** node in the Policy Studio tree, and select **XML Decryption Settings**.

Configure the fields on the **XML Decryption Settings** dialog as explained in the *XML-Decryption Settings* topic. When finished, an **XML Decryption Settings** filter is created along with an **XML Decryption** filter.

XML-Decryption Settings

Overview

The API Gateway can decrypt an XML encrypted message on behalf of its intended recipients. XML Encryption is a W3C standard that enables data to be encrypted and decrypted at the application layer of the OSI stack, thus ensuring complete end-to-end confidentiality of data.

You should use the **XML-Decryption Settings** in conjunction with the **XML-Decryption** filter, which performs the decryption. The **XML-Decryption Settings** generates the `decryption.properties` message attribute, which is required by the **XML-Decryption** filter.

Important

The output of a successfully executed decryption filter is the original unencrypted message. Depending on whether the **Remove EncryptedKey used in decryption** has been enabled, all information relating to the encryption key can be removed from the message. For more details, see Options section.

XML Encryption Overview

XML Encryption facilitates the secure transmission of XML documents between two application endpoints. Whereas traditional transport-level encryption schemes, such as SSL and TLS, can only offer point-to-point security, XML Encryption guarantees complete end-to-end security. Encryption takes place at the application-layer and so the encrypted data can be encapsulated in the message itself. The encrypted data can therefore remain encrypted as it travels along its path to the target Web Service. Furthermore, the data is encrypted such that only its intended recipients can decrypt it.

To understand how the API Gateway decrypts XML encrypted messages, you should first examine the format of an XML Encryption block. The following example shows a SOAP message containing information about Oracle:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <getCompanyInfo xmlns="www.oracle.com">
   <name>Company</name>
   <description>XML Security Company</description>
  </getCompanyInfo>
 </s:Body>
</s:Envelope>
```

After encrypting the SOAP Body, the message is as follows:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext" s:actor="Enc">
   <!-- Encapsulates the recipient's key details -->
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
           Id="00004190E5D1-7529AA14" MimeType="text/xml">
    <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#rsa-1_5">
     <enc:KeySize>256</enc:KeySize>
    </enc:EncryptionMethod>
    <enc:CipherData>
     <!-- The session key encrypted with the recipient's public key -->
     <enc:CipherValue>
         AAAAAJ/lK ... mrTF8Egg==
     </enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>sample</dsig:KeyName>
     <dsig:X509Data>
      <!-- The recipient's X.509 certificate -->
      <dsig:X509Certificate>
          MIIEZzCCA0 ... fzmc/YR5gA
      </dsig:X509Certificate>
     </dsig:X509Data>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
     <enc:DataReference URI="#00004190E5D1-5F889C11"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
         Id="00004190E5D1-5F889C11" MimeType="text/xml"
         Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData>
   <!-- The SOAP Body encrypted with the session key -->
   <enc:CipherValue>
       E2ioF8ib2r ... KJAnrX0GQV
   </enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
<s:Envelope>
```

The most important elements are as follows:

- `EncryptedKey`: The `EncryptedKey` element encapsulates all information relevant to the encryption key.
- `EncryptionMethod`: The `Algorithm` attribute specifies the algorithm that is used to encrypt the data. The message data (`EncryptedData`) is encrypted using the Advanced Encryption Standard (AES) *symmetric cipher*, but the session key (`EncryptedKey`) is encrypted with the RSA *asymmetric* algorithm.
- `CipherValue`: The value of the encrypted data. The contents of the `CipherValue` element are always Base64 encoded.
- `KeyInfo`: Contains information about the recipient and his encryption key, such as the key name, X.509 certificate, and Common Name.
- `ReferenceList`: This element contains a list of references to encrypted elements in the message. The `ReferenceList` contains a `DataReference` element for each encrypted element, where the value of a URI attribute points to the `Id` of the encrypted element. In the previous example, you can see that the `DataReference` URI attribute contains the value `#00004190E5D1-5F889C11`, which corresponds with the `Id` of the `EncryptedData` element.
- `EncryptedData`: The XML element(s) or content that has been encrypted. In this case, the SOAP `Body` element has been encrypted, and so the `EncryptedData` block has replaced the SOAP `Body` element.

Now that you have seen how encrypted data can be encapsulated in an XML message, it is important to discuss how this data gets encrypted in the first place. When you understand how data is encrypted, the fields that must be configured to decrypt this data become easier to understand.

When a message is encrypted, only the intended recipient(s) of the message can decrypt it. By encrypting the message with the recipient's public key, the sender can be guaranteed that only the intended recipient can decrypt the message using his private key, to which he has sole access. This is the basic principle behind *asymmetric cryptography*.

In practice, however, encrypting and decrypting data with a public-private key pair is notoriously CPU-intensive and time consuming. Because of this, asymmetric cryptography is seldom used to encrypt large amounts of data. The following steps exemplify a more typical encryption process:

1. The sender generates a one-time *symmetric* (or session) key which is used to encrypt the data. Symmetric key encryption is much faster than asymmetric encryption and is far more efficient with large amounts of data.
2. The sender encrypts the data with the symmetric key. This same key can then be used to decrypt the data. It is therefore crucial that only the intended recipient can access the symmetric key and consequently decrypt the data.
3. To ensure that nobody else can decrypt the data, the symmetric key is encrypted with the recipient's *public key*.
4. The data (encrypted with the symmetric key) and session key (encrypted with the recipient's public key) are then sent together to the intended recipient.
5. When the recipient receives the message he, decrypts the encrypted session key using his *private key*. Because the recipient is the only one with access to the private key, he is the only one who can decrypt the encrypted session key.
6. Armed with the decrypted session key, the recipient can decrypt the encrypted data into its original plaintext form.

Now that you understand how XML Encryption works, it is now time to learn how to configure the API Gateway to decrypt XML encrypted messages. The following sections describe how to configure the **XML Decryption Settings** filter to decrypt encrypted XML data.

Node(s) to Decrypt

An XML message may contain several `EncryptedData` blocks. This section specifies which encryption blocks are to be decrypted. There are two available options:

- Decrypt All Encrypted Nodes
- Use XPath to Select Encrypted Nodes

Decrypt All:
The API Gateway attempts to decrypt *all* `EncryptedData` blocks contained in the message.

Use XPath:
This option enables the administrator to explicitly choose the `EncryptedData` block that the API Gateway should decrypt.

For example, the following skeleton SOAP message contains two `EncryptedData` blocks:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
    ...
 <s:Header>
 <s:Body>
  <!-- 1st EncryptedData block -->
  <e:EncryptedData xmlns:e="http://www.w3.org/2001/04/xmlenc#"
          Encoding="iso-8859-1" Id="ENC_1" MimeType="text/xml"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
     ...
  </e:EncryptedData>
  <!-- 2nd EncryptedData block -->
  <e:EncryptedData xmlns:e="http://www.w3.org/2001/04/xmlenc#"
          Encoding="iso-8859-1" Id="ENC_2" MimeType="text/xml"
          Type="http://www.w3.org/2001/04/xmlenc#Element">
     ...
  </e:EncryptedData>
 </s:Body>
</s:Envelope>
```

The `EncryptedData` blocks are selected using XPath. You can use the following XPath expressions to select the respective `EncryptedData` blocks:

EncryptedData Block	***XPath Expression***
1st	`//enc:EncryptedData[@Id='ENC_1']`
2nd	`//enc:EncryptedData[@Id='ENC_2']`

Click the **Add**, **Edit**, or **Delete** buttons to add, edit, or remove an XPath expression.

Decryption Key

This section specifies the key to use to decrypt the encrypted nodes. As discussed in the section called "XML Encryption Overview", data encrypted with a public key can only be decrypted with the corresponding private key. The **Decryption Key** settings enable you to specify the private (decryption) key from the `<KeyInfo>` element of the XML Encryption block, or the certificate stored in the Oracle message attribute can be used to lookup the private key of the intended recipient of the encrypted data in the Certificate Store.

Find via KeyInfo in Message:
Select this option if you wish to determine the decryption key to use from the `KeyInfo` section of the `EncryptedKey` block. The `KeyInfo` section contains a reference to the public key used to encrypt the data. You can use this `KeyInfo` section reference to find the relevant private key (from the Oracle Certificate Store) to use to decrypt the data.

Find via certificate from Selector Expression:
Select this option if you do not wish to use the `KeyInfo` section in the message. Enter a selector expression that contains a certificate, (for example, `${certificate}`) whose corresponding private key is stored in the Oracle Certificate Store . Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, a Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Extract nodes from Selector Expression:
Specify whether to extract nodes from a specified selector expression (for example, `${node.list}`). This setting is not selected by default.

Typically, a **Find Certificate** filter is used in a policy to locate an appropriate certificate and store it in the `certificate` message attribute. When the certificate has been stored in this attribute, the **XML Decryption Settings** filter can use this certificate to lookup the Certificate Store for a corresponding private key for the public key stored in the certificate. To do this, select the `certificate` attribute from the drop-down list.

Options

The following configuration options are available in this section:

Fail if no encrypted data found:
If this option is selected, the filter fails if no `<EncryptedData>` elements are found within the message.

Remove the EncryptedKey used in decryption:
Select this option to remove information relating to the decryption key from the message. When this option is selected, the `<EncryptedKey>` block is removed from the message.

Important

In cases where the `<EncryptedKey>` block has been included in the `<EncryptedData>` block, it is removed regardless of whether this setting has been selected.

Default Derived Key Label:
If the API Gateway consumes a `<DerivedKeyToken>`, the default value entered is used to recreate the derived key that is used to decrypt the encrypted data.

Algorithm Suite Required:
Select the WS-Security Policy *Algorithm Suite* that must have been used when encrypting the message. This check ensures that the appropriate algorithms were used to encrypt the message.

Auto-generation using the XML Decryption Wizard

Because the **XML Decryption Settings** filter must always be paired with an **XML Decryption** filter, it makes sense to have a wizard that can generate both of these filters at the same time. To use the wizard, right-click the name of the policy in the tree view of the Policy Studio, and select the **XML Decryption Settings** menu option.

Configure the fields on the **XML Decryption Settings** dialog as explained in the previous sections. When finished, an **XML Decryption Settings** filter is created along with an **XML Decryption** filter.

XML-Encryption

Overview

The **XML-Encryption** filter is responsible for encrypting parts of XML messages based on the settings configured in the **XML-Encryption Settings** filter.

The **XML-Encryption Settings** filter generates the `encryption.properties` message attribute based on configuration settings. The **XML-Encryption** filter uses these properties to perform the encryption of the data.

Configuration

Enter a suitable name for the filter in the **Name** field.

Auto-generation using the XML Encryption Settings Wizard

Because the **XML Encryption** filter must always be used in conjunction with the **XML Encryption Settings** and **Find Certificate** filters, the Policy Studio provides a wizard that can generate these three filters at the same time. To use this wizard, right-click a policy node under the **Policies** node in the Policy Studio tree, and select the **XML Encryption Settings** menu option.

For more information on how to configure the **XML Encryption Settings Wizard** see the *XML Encryption Wizard* topic.

XML-Encryption Settings

Overview

The API Gateway can XML encrypt an XML message so that only certain specified recipients can decrypt the message. XML Encryption is a W3C standard that enables data to be encrypted and decrypted at the application layer of the OSI stack, thus ensuring complete end-to-end confidentiality of data.

The **XML-Encryption Settings** should be used in conjunction with the **XML-Encryption** filter, which performs the encryption. The **XML-Encryption Settings** generates the `encryption.properties` message attribute, which is required by the **XML-Encryption** filter.

XML Encryption Overview

XML Encryption facilitates the secure transmission of XML documents between two application endpoints. Whereas traditional transport-level encryption schemes, such as SSL and TLS, can only offer point-to-point security, XML Encryption guarantees complete end-to-end security. Encryption takes place at the application-layer, and so the encrypted data can be encapsulated in the message itself. The encrypted data can therefore remain encrypted as it travels along its path to the target Web Service.

Before explaining how to configure the API Gateway to encrypt XML messages, it is useful to examine an XML encrypted message. The following example shows a SOAP message containing information about Oracle:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <getCompanyInfo xmlns="http://www.oracle.com">
   <name>Company</name>
   <description>XML Security Company</description>
  </getCompanyInfo>
 </s:Body>
</s:Envelope>
```

After encrypting the SOAP Body, the message is as follows:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext" s:actor="Enc">
   <!-- Encapsulates the recipient's key details -->
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
           Id="00004190E5D1-7529AA14" MimeType="text/xml">
    <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#rsa-1_5">
     <enc:KeySize>256</enc:KeySize>
    </enc:EncryptionMethod>
    <enc:CipherData>
     <!-- The session key encrypted with the recipient's public key -->
     <enc:CipherValue>
         AAAAAJ/lK ... mrTF8Egg==
     </enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>sample</dsig:KeyName>
     <dsig:X509Data>
      <!-- The recipient's X.509 certificate -->
      <dsig:X509Certificate>
          MIIEZzCCA0 ... fzmc/YR5gA
      </dsig:X509Certificate>
     </dsig:X509Data>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
     <enc:DataReference URI="#00004190E5D1-5F889C11"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
         Id="00004190E5D1-5F889C11" MimeType="text/xml"
         Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData>
   <!-- The SOAP Body encrypted with the session key -->
   <enc:CipherValue>
       E2ioF8ib2r ... KJAnrX0GQV
   </enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
<s:Envelope>
```

The most important elements are as follows:

- `EncryptedKey`:
  The `EncryptedKey` element encapsulates all information relevant to the encryption key.
- `EncryptionMethod`:
  The `Algorithm` attribute specifies the algorithm used to encrypt the data. The message data (`EncryptedData`) is encrypted using the Advanced Encryption Standard (AES) *symmetric cipher*, but the session key (`EncryptedKey`) is encrypted with the RSA *asymmetric* algorithm.
- `CipherValue`:
  The value of the encrypted data. The contents of the `CipherValue` element are always Base64 encoded.
- `DigestValue`:
  Contains the Base64-encoded message-digest.
- `KeyInfo`:
  Contains information about the recipient and his encryption key, such as the key name, X.509 certificate, and Common Name.
- `ReferenceList`: This element contains a list of references to encrypted elements in the message. It contains a `DataReference` element for each encrypted element, where the value of a URI attribute points to the `Id` of the encrypted element. In the previous example, the `DataReference` URI attribute contains the value `#00004190E5D1-5F889C11`, which corresponds with the `Id` of the `EncryptedData` element.
- `EncryptedData`:
  The XML elements or content that has been encrypted. In this case, the SOAP `Body` element has been encrypted, and so the `EncryptedData` block has replaced the SOAP `Body` element.

Now that you have seen how encrypted data can be encapsulated in an XML message, it is important to discuss how the data is encrypted. When a message is encrypted, it is encrypted in such a manner that only the intended recipients of the message can decrypt it. By encrypting the message with the recipient public key, the sender can be guaranteed that only the intended recipient can decrypt the message using his private key, to which he has sole access. This is the basic principle behind *asymmetric cryptography*.

In practice, however, encrypting and decrypting data with a public-private key pair is a notoriously CPU-intensive and time consuming affair. Because of this, asymmetric cryptography is seldom used to encrypt large amounts of data. The following steps show a more typical encryption process:

1. The sender generates a one-time *symmetric* (or session) key which is used to encrypt the data. Symmetric key encryption is much faster than asymmetric encryption, and is far more efficient with large amounts of data.
2. The sender encrypts the data with the symmetric key. This same key can then be used to decrypt the data. It is therefore crucial that only the intended recipient can access the symmetric key and consequently decrypt the data.
3. To ensure that nobody else can decrypt the data, the symmetric key is encrypted with the recipient's *public key*.
4. The data (encrypted with the symmetric key), and session key (encrypted with the recipient's public key), are then sent together to the intended recipient.
5. When the recipient receives the message, he decrypts the encrypted session key using his *private key*. Because the recipient is the only one with access to the private key, only he can decrypt the encrypted session key.
6. Armed with the decrypted session key, the recipient can decrypt the encrypted data into its original plaintext form.

Now that you understand the structure and mechanics of XML Encryption, you can configure the API Gateway to encrypt egress XML messages. The next section describes how to configure the tabs on the **XML Encryption Settings** screen.

Encryption Key

The settings on this tab determine the key to use to encrypt the message, and how this key is referred to in the encrypted data. The following configuration options are available:

Important

A symmetric key is used to encrypt the data. This symmetric key is then encrypted (asymmetrically) with the recipient's public key. In this way, only the recipient can decrypt the symmetric encryption key with its private key. When the recipient has access to the unencrypted encryption key, it can decrypt the data.

Generate Encryption Key:
Select this option to generate a symmetric key to encrypt the data with.

Encryption Key from Selector Expression:
If you have already used a symmetric key in a previous filter (for example, a **Sign Message** filter), you can reuse that key to encrypt data by selecting this option and specifying a selector expression to obtain the key (for example, `${symmetric.key}`). Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, a Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Include Encryption Key in Message:
Select this option if you want to include the encryption key in the message. The encryption key is encrypted for the recipient so that only the recipient can access the encryption key. You may choose not to include the symmetric key in the message if the API Gateway and recipient have agreed on the symmetric encryption key using some other means.

Specify Method of Associating the Encryption Key with the Encrypted Data:
This section enables you to configure the method by which the encrypted data references the key used to encrypt it. The following options are available:

- ***Point to Encryption Key with Security Token Reference:***
  This option creates a `<SecruityTokenReference>` in the `<EncryptedData>` that points to an `<EncryptedKey>`.
- ***Embed Symmetric Key Inside Encrypted Data:***
  Place the `<xenc:EncryptedKey>` inside the `<xenc:EncryptedData>` element.
- ***Specify Encryption Key via Carried Keyname:***
  Place the encrypted key's carried keyname inside the `<dsig:KeyInfo>`/ `<dsig:KeyName>` of the `<xenc:EncryptedData>`.
- ***Specify Encryption Key via Retrieval Method:***
  Refer to a symmetric key via a retrieval method reference from the `<xenc:EncryptedData>`.

- ***Symmetric Key Refers to Encrypted Data:***
  The symmetric key refers to <xenc:EncryptedData> using a reference list.

Use Derived Key:
Select this option if you want to derive a key from the symmetric key configured above to encrypt the data. The <enc:EncryptedData> has a <wsse:SecurityTokenReference> to the <wssc:DerivedKeyToken>. The <wssc:DerivedKeyToken> refers to the <enc:EncryptedKey>. Both <wssc:DerivedKeyToken> and <enc:EncryptedKey> are placed inside a <wsse:Security> element.

Key Info

This tab configures the content of the <KeyInfo> section of the generated <EncryptedData> block. Configure the following fields on this tab:

Do Not Include KeyInfo Section:
This option enables you to omit all information about the certificate that contains the public key that was used to encrypt the data from the <EncryptedData> block. In other words, the <KeyInfo> element is omitted from the <EncryptedData> block. This is useful where a downstream Web Service uses an alternative method to decide what key to use to decrypt the message. In such cases, adding certificate information to the message may be regarded as an unnecessary overhead.

Include Certificate:
This is the default option, which places the certificate that contains the encryption key inside the <EncryptedData>. The following example, shows an example of a <KeyInfo> that has been produced using this option:

```
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</enc:EncryptedData>
```

Expand Public Key:
The details of the public key used to encrypt the data are inserted into a <KeyValue> block. The <KeyValue> block is only inserted when this option is selected.

```
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
   ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
      MIIE ....... EQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
  <dsig:KeyValue>
   <dsig:RSAKeyValue>
    <dsig:Modulus>
       AMfb2tT53GmMiD
       ...
       NmrNht7iy18=
    </dsig:Modulus>
    <dsig:Exponent>AQAB</dsig:Exponent>
   </dsig:RSAKeyValue>
  </dsig:KeyValue>
 </dsig:KeyInfo>
</enc:EncryptedData>
```

Include Distinguished Name:
If this checkbox is selected, the Distinguished Name of the certificate that contains the public key used to encrypt the data is inserted in an <X509SubjectName> element as shown in the following example:

```
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample,C=IE...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</enc:EncryptedData>
```

Include Key Name:
This option enables you insert a key identifier, or <KeyName>, to allow the recipient to identify the key to use to decrypt the data. Enter an appropriate value for the <KeyName> in the **Value** field. Typical values include Distinguished Names (DName) from X.509 certificates, key IDs, or email addresses. Specify whether the specified value is a **Text value** of a **Distinguished name attribute** by selecting the appropriate radio button.

```
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
   ...
 <dsig:KeyInfo>
  <dsig:KeyName>test@oracle.com</dsig:KeyName>
 </dsig:KeyInfo>
</enc:EncryptedData>
```

Put Certificate in an Attachment:
The API Gateway supports SOAP messages with attachments. By selecting this option, you can save the certificate containing the encryption key to the file specified in the input field. This file can then be sent along with the SOAP message as a SOAP attachment.

From previous examples, it is clear that the user's certificate is usually placed inside a <KeyInfo> element. However, in this example, the certificate is contained in an attachment, and not in the <EncryptedData>. Clearly, you need a way to reference the certificate from the <EncryptedData> block, so that the recipient can determine what key it should use to decrypt the data. This is the role of the <SecurityTokenReference> block.

The <SecurityTokenReference> block provides a generic mechanism for applications to retrieve security tokens in cases where these tokens are not contained in the SOAP message. The name of the security token is specified in the URI attribute of the <Reference> element.

```
<enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
   ...
 <dsig:KeyInfo>
  <wsse:SecurityTokenReference xmlns:wsse="http://schemas.xmlsoap.org/ws/...">
   <wsse:Reference URI="c:\myCertificate.txt"/>
  </wsse:SecurityTokenReference>
 </dsig:KeyInfo>
</enc:EncryptedData>
```

When the message is sent, the certificate attachment is given a `Content-Id` corresponding to the `URI` attribute of the `<Reference>` element. The following example shows the wire format of the complete multipart MIME SOAP message. It should help illustrate how the `<Reference>` element refers to the `Content-ID` of the attachment:

```
POST /adoWebSvc.asmx HTTP/1.0
Content-Length: 3790
User-Agent: API Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
Content-Type: text/xml; charset="utf-8";
SOAPAction=getQuote

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   ...
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
    ...
  <dsig:KeyInfo>
   <ws:SecurityTokenReference xmlns:ws="http://schemas.xmlsoap.org/ws/...">
    <ws:Reference URI="c:\myCertificate.txt"/>
   </ws:SecurityTokenReference>
  </dsig:KeyInfo>
 </enc:EncryptedData>
    ...
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: c:\myCertificate.txt
Content-Type: text/plain; charset="US-ASCII"

MIIEZDCCA0ygAwIBAgIBAzANBgkqhki
....
7uFveG0eL0zBwZ5qwLRNp9aKD1fEQgJ
------=Multipart-SOAP-boundary-
```

Security Token Reference:
A `<wsse:SecurityTokenReference>` element can be used to point to the security token used to encrypt the data. If you wish to use a `<wsse:SecurityTokenReference>`, enable this option, and select a Security Token Reference type from **Reference Type** drop-down list.

The `<wsse:SecurityTokenReference>`, (in the `<dsig:KeyInfo>`), may contain a `<wsse:Embedded>` security token. Alternatively, the `<wsse:SecurityTokenReference>`, (in the `<dsig:KeyInfo>`), may refer to a certificate using a `<dsig:X509Data>`. Select the appropriate button, **Embed** or **Refer**, depending on whether you want to use an embedded security token or a referred one.

If you have configured the `SecurityContextToken (sct)` mechanism from the **Security Token Reference** drop-down list, you can select to use an **Attached SCT** or an **Unattached SCT**. The default option is to use an **Attached SCT**, which should be used in cases where the SCT refers to a security token inside the `<wsse:Security>` header. If the SCT is located outside the `<wsse:Security>` header, you should select the **Unattached SCT** option.

You can make sure to include a `<BinarySecurityToken>` (BST) that contains the certificate (that contains the encryption key) in the message by selecting the **Include BinarySecurityToken** option. The BST is inserted into the WS-Security header regardless of the type of Security Token Reference selected from the dropdown.

Select **Include TokenType** if you want to add the `TokenType` attribute to the `SecurityTokenReference` element.

Important

When using the Kerberos Token Profile standard, and the API Gateway is acting as the initiator of a secure transaction, it can use Kerberos session keys to encrypt a message. The `KeyInfo` must be configured to use a Security Token Reference with a `ValueType` of `GSS_Kerberosv5_AP_REQ`. In this case, the Kerberos token is contained in a `<BinarySecurityToken>` in the message.

If the API Gateway is acting as the recipient of a secure transaction, it can also use the Kerberos session keys to encrypt the message returned to the client. However, in this case, the `KeyInfo` must be configured to use a Security Token Reference with `ValueType` of `Kerberosv5_APREQSHA1`. When this is selected, the Kerberos token is not contained in the message. The Security Token Reference contains a SHA1 digest of the original Kerberos token received from the client, which identifies the session keys to the client.

When using the WS-Trust for SPENGO standard, the Kerberos session keys are not used directly to encrypt messages because a security context with an associated symmetric key is negotiated. This symmetric key is shared by both client and service and can be used to encrypt messages on both sides.

Recipients

XML Messages can be encrypted for multiple recipients. In such cases, the symmetric encryption key is encrypted with the public key of each intended recipient and added to the message. Each recipient can then decrypt the encryption key with their private key and use it to decrypt the message.

The following SOAP message has been encrypted for 2 recipients (*oracle_1* and *oracle_2*). The encryption key has been encrypted twice: once for *oracle_1* using its public key, and a second time for *oracle_2* using its public key:

Important

The data itself is only encrypted once, while the encryption key must be encrypted for each recipient. For illustration purposes, only those elements relevant to the above discussion have been included in the following XML encrypted message.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <Security xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext"
      s:actor="Enc Keys">
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
        Id="0000418BBB61-A692675C" MimeType="text/xml">
    ...
    <enc:CipherData>
     <!-- Enc key encrypted with oracle_1's public key and base64-encoded -->
     <enc:CipherValue>AAAAAExx1A ... vuAhCgMQ==</enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>oracle_1</dsig:KeyName>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
      <enc:ReferenceList>
   <enc:DataReference URI="#0000418BBB61-D4495D9B"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
   <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
        Id="#0000418BBB61-D4495D9B" MimeType="text/xml">
    ...
    <enc:CipherData>
     <!-- Enc key encrypted with oracle_2's public key and base64-encoded -->
     <enc:CipherValue>AAAAABZH+U ... MrMEEM/Ps=</enc:CipherValue>
    </enc:CipherData>
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <dsig:KeyName>oracle_2</dsig:KeyName>
    </dsig:KeyInfo>
    <enc:CarriedKeyName>Session key</enc:CarriedKeyName>
    <enc:ReferenceList>
   <enc:DataReference URI="#0000418BBB61-D4495D9B"/>
    </enc:ReferenceList>
   </enc:EncryptedKey>
  </Security>
 </s:Header>
 <enc:EncryptedData xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
      Id="0000418BBB61-D4495D9B" MimeType="text/xml"
      Type="http://www.w3.org/2001/04/xmlenc#Element">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04xmlenc#aes256-cbc">
   <enc:KeySize>256</enc:KeySize>
  </enc:EncryptionMethod>
  <enc:CipherData>
   <!-- SOAP Body encrypted with symmetric enc key and base64-encoded -->
   <enc:CipherValue>WD0TmuMk9 ... GzYFeq8SM=</enc:CipherValue>
  </enc:CipherData>
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:KeyName>Session key</dsig:KeyName>
  </dsig:KeyInfo>
 </enc:EncryptedData>
</s:Envelope>
```

There are two <EncryptedKey> elements, one for each recipient. The <CipherValue> element contains the symmetric encryption key encrypted with the recipient's public key. The encrypted symmetric key must be Base64-encoded so that it can be represented as the textual contents of an XML element.

The <EncryptedData> element contains the encrypted data, along with information about the encryption process, including the encryption algorithm used, the size of the encryption key, and the type of data that was encrypted (for example, whether an element or the contents of an element was encrypted).

Click the **Add** button to add a new recipient for which the data will be encrypted. Configure the following fields on the **XML Encryption Recipient** dialog:

Recipient Name:
Enter a name for the recipient. This name can then be selected on the main **Recipients** tab of the filter.

Actor:
The <EncryptedKey> for this recipient is inserted into the specified SOAP actor/role.

Use Key in Message Attribute:
Specify the message attribute that contains the recipient's public key that is used to encrypt the data. By default, the certificate attribute is used. Typically, this attribute is populated by the **Find Certificate** filter, which retrieves a certificate from any one of a number of locations, including the Certificate Store, an LDAP directory, HTTP header, or from the message itself.

If you want to encrypt the message for multiple recipients, you must configure multiple **Find Certificate** filters (or some other filter that can retrieve certificates). Each **Find Certificate** filter retrieves a certificate for a single recipient and store it in a unique message attribute.

For example, a **Find Certificate** filter called **Find Certificate for Recipient1** filter could locate Recipient1's certificate from the Certificate Store and store it in a certificate_recip1 message attribute. You would then configure a second **Find Certificate** filter called **Find Certificate for Recipient2**, which could retrieve Recipient2's certificate from the Certificate Store and store it in a certificate_recip2 message attribute.

On the **Recipients** tab of the **XML Encryption Settings** filter, you would then configure two recipients. For the first recipient (Recipient1), you would enter certificate_recip1 as the location of the encryption key, while for the second recipient (Recipient2), you would specify `certificate_recip2` as the location of the encryption key.

Note

If the API Gateway fails to encrypt the message for any of the recipients configured on the **Recipients** tab, the filter will fail.

What to Encrypt

This tab is used to identify parts of the message that must be encrypted. Each encrypted part will be replaced by an `<EncryptedData>` block, which contains all information required to decrypt the block.

You can use *any* combination of **Node Locations**, **XPaths**, and the nodes contained in a **Message Attribute** to specify the nodes that are required to be encrypted. Please refer to the *Locate XML Nodes* filter for more information on how to use these node selectors.

Important

Note the difference between encrypting the element and encrypting the element content. When encrypting the element, the entire element is replaced by the `<EncryptedData>` block. This is not recommended, for example, if you wish to encrypt the SOAP Body because if this element is removed from the SOAP message, the message may no longer be considered a valid SOAP message.

Element encryption is more suitable when encrypting security blocks, (for example, WS-Security Username tokens and SAML assertions) that may appear in a WS-Security header of a SOAP message. In such cases, replacing the element content (for example, a `<UsernameToken>` element) with an `<EncryptedData>` block will not affect the semantics of the WS-Security header.

If you wish to encrypt the SOAP Body, you should use element content encryption, where the children of the element are replaced by the `<EncryptedData>` block. In this way, the message can still be validated against the SOAP schema.

When using **Node Locations** to identify nodes that are to be encrypted, you can configure whether to encrypt the element or the element contents on the **Locate XML Nodes** dialog. To encrypt the element, select the **Encrypt Node** radio button. Alternatively, to encrypt the element contents, select the **Encrypt Node Content** radio button.

If you are using XPath expressions to specify the nodes that are to be signed, be careful not to use an expression that returns a node and all its contents. The **Encrypt Node** and **Encrypt Node Content** options are also available when configuring XPath expressions on the **Enter XPath Expression** dialog.

Advanced

The **Advanced** tab on the main **XML-Encryption Settings** screen enables you to configure some of the more complicated settings regarding XML-Encryption. The following settings are available:

Algorithm Suite Tab:
The following fields can be configured on this tab:

Algorithm Suite:
WS-Security Policy defines a number of *algorithm suites* that group together a number of cryptographic algorithms. For example, a given algorithm suite uses specific algorithms for asymmetric encryption, symmetric encryption, asymmetric key wrap, and so on. Therefore, by specifying an algorithm suite, you are effectively selecting a whole suite of cryptographic algorithms to use.

If you want to use a particular WS-Security Policy algorithm suite, you can select it here. The **Encryption Algorithm** and **Key Wrap Algorithm** fields are automatically populated with the corresponding algorithms for that suite.

Encryption Algorithm:
The encryption algorithm selected is used to encrypt the data. The following algorithms are available:

- AES-256
- AES-192
- AES-128
- Triple DES

Key Wrap Algorithm:
The key wrap algorithm selected here is used to wrap (encrypt) the symmetric encryption key with the recipient's public key. The following key wrap algorithms are available:

- KwRsa15
- KwRsaOaep

Settings Tab:
The following advanced settings are available on this tab:

Generate a Reference List in WS-Security Block:
When this option is selected, a `<xenc:ReferenceList>` that holds a reference to all encrypted data elements is generated. The `<xenc:ReferenceList>` element is inserted into the WS-Security block indicated by the specified actor.

Insert Reference List into EncryptedKey:
When this option is selected, a `<xenc:ReferenceList>` that holds a reference to all encrypted data elements is generated. The `<xenc:ReferenceList>` element is inserted into the `<xenc:EncryptedKey>` element.

Layout Type:
Select the WS-SecurityPolicy layout type that you want the generated tokens to adhere to. This includes elements such as the `<EncryptedData>`, `<EncryptedKey>`, `<ReferenceList>`, `<BinarySecurityToken>`, and `<DerivedKeyToken>` tokens, among others.

Fail if no Nodes to Encrypt:
Select this option if you want the filter to fail if any of the nodes specified on the **What to Encrypt** tab are found in the message.

Insert Timestamp:
This option enables you to insert a WS-Security Timestamp as an encryption property.

Indent:
This option enables you to format the inserted `<EncryptedData>` and `<EncryptedKey>` blocks by indenting the elements.

Insert CarriedKeyName for EncryptedKey:
Select this option to insert a `<CarriedKeyName>` element into the generated `<EncryptedKey>` block.

Auto-generation using the XML Encryption Settings Wizard

Because the **XML Encryption Settings** filter must always be used in conjunction with the **XML Encryption** and **Find Certificate** filters, the Policy Studio provides a wizard that can generate these three filters at the same time. Right-click a policy under the **Policies** node in the Policy Studio, and select **XML Encryption Settings**.

For more information on how to configure the **XML Encryption Settings Wizard** see the *XML Encryption Wizard* topic.

XML Encryption Wizard

Overview

The following filters are involved in encrypting a message using XML Encryption:

Filter	***Role***
Find Certificate	Specifies the certificate that contains the public key to use in the encryption. The data is encrypted such that it can only be decrypted with the corresponding private key.
XML Encryption Settings	Specifies the recipient of the encrypted data, what data to encrypt, what algorithms to use, and other such options that affect the way the data is encrypted.
XML Encryption	Performs the actual encryption using the certificate selected in the **Find Certificate** filter, and the options set in the **XML Encryption Settings** filter.

While these filters can be configured independently of each other, it makes sense to configure them all at the same time because they must play a role in the policy that XML-Encrypts messages. You can do this using the **XML Encryption Wizard**. The wizard is available by right-clicking the name of the policy in the tree view of the Policy Studio, and selecting the **XML Encryption Settings** menu option. The next section describes how to configure the settings on this dialog.

Configuration

The first step in configuring the **XML Encryption Wizard** is to select the certificate that contains the public key to use to encrypt the data. When the data has been encrypted with this public key, it can only be decrypted using the corresponding private key. Select the relevant certificate from the list of **Certificates in the Trusted Certificate Store**.

When the wizard is completed, the information configured on this screen results in the auto-generation of a **Find Certificate** filter. This filter is automatically configured to use the selected certificate from the Certificate Store. For more details, see the *Find Certificate* tutorial.

After clicking the **Next** button on the first screen of the wizard, the configuration options for the **XML Encryption Settings** filter are displayed. For more details, see the *XML-Encryption Settings* topic.

When you have completed all the steps in the wizard, a policy is created that comprises a **Find Certificate**, **XML Encryption Settings**, and **XML Encryption** filter. You can insert other filters into this policy as required, however, the order of the encryption filters must be maintained as follows:

1. Find Certificate
2. XML Encryption Settings
3. XML Encryption

XML Signature Generation

Overview

The API Gateway can sign both SOAP and non-SOAP XML messages. Attachments to the message can also be signed. The resulting XML Signature is inserted into the message for consumption by a downstream Web Service. At the Web Service, the signature can be used to authenticate the message sender and/or verify the integrity of the message.

Enter a name for the filter in the **Name** field. You can use the following tabs to configure various aspects of the generated XML Signature.

Signing Key

You can use either a symmetric or an asymmetric key to sign the message content. Select the appropriate radio button and configure the fields on the corresponding tab.

Asymmetric Key

With an asymmetric signature, the signatory's private key (from a public-private key pair) is used to sign the message. The corresponding public key is then used to verify the signature. The following fields are available for configuration on this tab:

Private Key in Certificate Store:
To use a signing key from the Certificate Store, select the **Key in Store** radio button and, click the **Signing Key** button. Select a certificate that has the required signing key associated with it. The signing key can also be stored on a Hardware Security Module (HSM). For more information on storing signing keys, see the the *Certificates and Keys* topic.

The *Distinguished Name* of the selected certificate will appear in the `X509SubjectName` element of the XML Signature as follows:

```
<dsig:X509SubjectName>
   CN=Sample,OU=R&D,O=Company Ltd.,L=Dublin 4,ST=Dublin,C=IE
</dsig:X509SubjectName>
```

Private Key from Selector Expression:
Alternatively, the signing key may have already have been used by another filter and stored in a message attribute. To reuse this key, select the **Private Key from Selector Expression** radio button, and enter the selector expression in the field provided (for example, `${asymmetric.key}`). Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Symmetric Key

With a symmetric signature, the same key is used to sign and verify the message. Typically the client generates the symmetric key and uses it to sign the message. The key must then be transmitted to the recipient so that they can verify the signature. It would be unsafe to transmit an unprotected key along with the message so it is usually encrypted (or wrapped) with the recipient's public key. The key can then be decrypted with the recipient's private key and can then be used to verify the signature. The following configuration options are available on this screen:

Generate Symmetric Key, and Save in Message Attribute:
If you select this option, the API Gateway generates a symmetric key, which is included in the message before it is sent to the client. By default, the key is saved in the `symmetric.key` message attribute.

Symmetric Key from Selector Expression:
If a previous filter (for example, a **Sign Message** filter) has already used a symmetric key, you can to reuse this key as proof that the API Gateway is the holder-of-key entity. Enter the name of the selector expression in the field provided,

which defaults to `${symmetric.key}`. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, a Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Include Encrypted Symmetric Key in Message:
As described earlier, the symmetric key is typically encrypted for the recipient and included in the message. However, it is possible that the initiator and recipient of the transaction have agreed on a symmetric key using some out-of-bounds mechanism. In this case, it is not necessary to include the key in the message. However, the default option is to include the encrypted symmetric key in the message. The `<KeyInfo>` section of the Signature points to the `<EncryptedKey>`.

Encrypt with Key in Store:
Select this option to encrypt the symmetric key with a public key from the Certificate Store. Click the **Signing Key** button and then select the certificate that contains the public key of the recipient. By encrypting the symmetric key with this public key, you are ensuring that only the recipient that has access to the corresponding private key will be able to decrypt the encrypted symmetric key.

Encrypt with Key from Selector Expression:
You can also use a key stored in a message attribute to encrypt (or wrap) the symmetric key. Select this radio button and enter the selector expression to obtain the public key you want to use to encrypt the symmetric key with. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, a Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Use Derived Key:
A `<wssc:DerivedKeyToken>` token can be used to derive a symmetric key from the original symmetric key held in and `<enc:EncryptedKey>`. The derived symmetric key is then used to actually sign the message, as opposed to the original symmetric key. It must be derived again during the verification process using the parameters in the `<wssc:DerivedKeyToken>`. One of these parameters is the symmetric key held in `<enc:EncryptedKey>`. The following example shows the use of a derived key:

```
<enc:EncryptedKey Id="Id-0000010b8b0415dc-0000000000000000">
  <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <dsig:KeyInfo>
       ...
  </dsig:KeyInfo>
  <enc:CipherData>
</enc:EncryptedKey>

<wssc:DerivedKeyToken wsu:Id="Id-0000010bd2b8eca1-0000000000000017"
      Algorithm="http://schemas.xmlsoap.org/ws/2005/02/sc/dk/p_sha1">
  <wsse:SecurityTokenReference wsu:Id="Id-0000010bd2b8ed5d-0000000000000018">
    <wsse:Reference URI="#Id Id-0000010b8b0415dc-0000000000000000"
    ValueType="..../oasis-wss-soap-message-security-1.1#EncryptedKey"/>
  </wsse:SecurityTokenReference>
  <wssc:Generation>0</wssc:Generation>
  <wssc:Length>32</wssc:Length>
  <wssc:Label>WS-SecureConverstaionWS-SecureConverstaion</wssc:Label>
  <wssc:Nonce>h9TTWKRylCOz87+mc1/7Pg==</wssc:Nonce>
</wssc:DerivedKeyToken>

<dsig:Signature Id="Id-0000010b8b0415dc-0000000000000004">
  <dsig:SignedInfo>
    <dsig:CanonicalizationMethod
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <dsig:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <dsig:Reference>...</dsig:Reference>
  </dsig:SignedInfo>
  <dsig:SignatureValue>...dsig:SignatureValue>
  <dsig:KeyInfo>
    <wsse:SecurityTokenReference wsu:Id="Id-0000010b8b0415dc-0000000000000006">
      <wsse:Reference
          URI="# Id-0000010bd2b8eca1-0000000000000017"
          ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk"/>
    </wsse:SecurityTokenReference>
  </dsig:KeyInfo>
</dsig:Signature>
```

Symmetric Key Length:
This option enables the user to specify the length of the key to use when performing symmetric key signatures. It is important to realize that the longer the key, the stronger the encryption.

Key Info

This tab configures how the `<KeyInfo>` block of the generated XML Signature is displayed. Configure the following fields on this tab:

Do Not Include KeyInfo Section:
This option enables you to omit all information about the signatory's certificate from the signature. In other words, the `KeyInfo` element is omitted from the signature. This is useful where a downstream Web Service uses an alternative method of authenticating the signatory, and wishes to use the signature for the sole purpose of verifying the integrity of the message. In such cases, adding certificate information to the message may be regarded as an unnecessary overhead.

Include Certificate:
This is the default option which places the signatory's certificate inside the XML Signature itself. The following example, shows an example of an XML Signature which has been created using this option:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
    ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Expand Public Key:
The details of the signatory's public key are inserted into a `KeyValue` block. The `KeyValue` block is only inserted when this option is selected.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample...</dsig:X509SubjectName>
   <dsig:X509Certificate>
      MIIE ....... EQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
  <dsig:KeyValue>
   <dsig:RSAKeyValue>
    <dsig:Modulus>
       AMfb2tT53GmMiD
       ...
       NmrNht7iy18=
    </dsig:Modulus>
    <dsig:Exponent>AQAB</dsig:Exponent>
   </dsig:RSAKeyValue>
  </dsig:KeyValue>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Distinguished Name:
If this checkbox is selected, the Distinguished Name of the signatory's X.509 certificate is inserted in an `<X509SubjectName>` element as shown in the following example:

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 <dsig:KeyInfo>
  <dsig:X509Data>
   <dsig:X509SubjectName>CN=Sample,C=IE...</dsig:X509SubjectName>
   <dsig:X509Certificate>
       MIIEZDCCA0yg
       ....
       RNp9aKD1fEQgJ
   </dsig:X509Certificate>
  </dsig:X509Data>
 </dsig:KeyInfo>
</dsig:Signature>
```

Include Key Name:
This option allows you insert a key identifier, or `KeyName`, to allow the recipient to identify the signatory. Enter an appropriate value for the `KeyName` in the **Value** field. Typical values include Distinguished Names (DName) from X.509 certificates, key IDs, or email addresses. Specify whether the specified value is a **Text value** of a **Distinguished name attribute** by checking the appropriate radio button.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <dsig:KeyName>test@oracle.com</dsig:KeyName>
 </dsig:KeyInfo>
</dsig:Signature>
```

Put Certificate in an Attachment:
The API Gateway supports SOAP messages with attachments. By selecting this option, you can save the signatory's certificate to the file specified in the input field. This file can then be sent along with the SOAP message as a SOAP attachment.

From previous examples, it is clear that the user's certificate is usually placed inside a `KeyInfo` element. However, in this example, the certificate is actually contained within an attachment, and not within the XML Signature itself. Clearly, we need a way to reference the certificate from the XML Signature, so that validating applications can process the signature correctly. This is the role of the `SecuriyTokenReference` block.

The `SecurityTokenReference` block provides a generic way for applications to retrieve security tokens in cases where these tokens are not contained within the SOAP message. The name of the security token is specified in the `URI` attribute of the `Reference` element.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
   ...
 <dsig:KeyInfo>
  <wsse:SecurityTokenReference xmlns:wsse="http://schemas.xmlsoap.org/ws/...">
   <wsse:Reference URI="c:\myCertificate.txt"/>
  </wsse:SecurityTokenReference>
 </dsig:KeyInfo>
```

```
</dsig:Signature>
```

When the message is actually sent, the certificate attachment will be given a "Content-Id" corresponding to the `URI` attribute of the `Reference` element. The following example shows what the complete multipart MIME SOAP message looks like as it is sent over the wire. It should help illustrate how the `Reference` element actually refers to the "Content-ID" of the attachment:

```
POST /adoWebSvc.asmx HTTP/1.0
Content-Length: 3790
User-Agent: API Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
Content-Type: text/xml; charset="utf-8";
SOAPAction=getQuote

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   ...
 <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
    ...
  <dsig:KeyInfo>
   <ws:SecurityTokenReference xmlns:ws="http://schemas.xmlsoap.org/ws/...">
    <ws:Reference URI="c:\myCertificate.txt"/>
   </ws:SecurityTokenReference>
  </dsig:KeyInfo>
 </dsig:Signature>
    ...
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: c:\myCertificate.txt
Content-Type: text/plain; charset="US-ASCII"

MIIEZDCCA0ygAwIBAgIBAzANBgkqhki
....
7uFveG0eL0zBwZ5qwLRNp9aKD1fEQgJ
------=Multipart-SOAP-boundary-
```

Security Token Reference:
A `<wsse:SecurityTokenReference>` element can be used to point to the security token used in the generation of the signature. Select this option if you wish to use this element. The type of the reference must be selected from the **Reference Type** dropdown.

The `<wsse:SecurityTokenReference>`, (within the `<dsig:KeyInfo>`), may contain a `<wsse:Embedded>` security token. Alternatively, the `<wsse:SecurityTokenReference>`, (within the `<dsig:KeyInfo>`), may refer to a certificate via a `<dsig:X509Data>`. Select the appropriate button, **Embed** or **Refer**, depending on whether you want to use an embedded security token or a referred one.

You can make sure to include a `<BinarySecurityToken>` (BST) that contains the certificate used to wrap the symmetric key in the message by selecting the **Include BinarySecurityToken** option. The BST is inserted into the WS-Security header regardless of the type of Security Token Reference selected from the dropdown.

Important

When using the Kerberos Token Profile standard and the API Gateway is acting as the initiator of a secure transaction, it can use Kerberos session keys to sign a message. The `KeyInfo` must be configured to use a Security Token Reference with a `ValueType` of `GSS_Kerberosv5_AP_REQ`. In this case, the Kerberos token is contained in a `<BinarySecurityToken>` in the message.

If the API Gateway is acting as the recipient of a secure transaction, it can also use the Kerberos session keys to sign the message returned to the client. However, in this case, the `KeyInfo` must be configured to use a Security Token Reference with `ValueType` of `Kerberosv5_APREQSHA1`. When this `ValueType` is selected, the Kerberos token is not contained in the message. The Security Token Reference contains a SHA1 digest of the original Kerberos token received from the client, which identifies the session keys to the client.

Note

When using the WS-Trust for SPENGO standard, the Kerberos session keys are not used directly to sign messages since a security context with an associated symmetric key is negotiated. This symmetric key is shared by both client and service and can be used to sign messages on both sides.

What to Sign

This tab is used to identify parts of the message that must be signed. Each signed part will be referenced from within the generated XML Signature. You can use *any* combination of **Node Locations**, **XPaths**, **XPath Predicates**, and the nodes contained in a **Message Attribute** to specify what must be signed. For details on the settings on these tabs, see the *Locate XML Nodes* filter.

XML Signing Mechanisms
It is important to consider the mechanisms available for referencing signed elements from within an XML Signature. For example, With WSU Ids, an Id attribute is inserted into the root element of the nodeset that is to be signed. The XML Signature then references this Id to indicate to verifiers of the Signature the nodes that were signed. The use of WSU Ids is the default option because these are WS-I compliant.

Alternatively, a generic Id attribute (not bound to the WSU namespace) can be used to dereference the data. The Id attribute is inserted into the top-level element of the nodeset that is to be signed. The generated XML Signature can then reference this Id to indicate what nodes were signed.

When XPath transforms are used, an XPath expression that points to the root node of the nodeset that is signed will be inserted into the XML Signature. When attempting to verify the Signature, this XPath expression must be run on the message to retrieve the signed content.

Id Attribute:
Select the Id attribute used to dereference the signed element in the `dsig:Signature`. The available options are as follows:

- *wsu:Id*
  The default option references the signed data using a `wsu:Id` attribute. A `wsu:Id` attribute is inserted into the root node of the signed nodeset. This Id is then referenced in the generated XML Signature as an indication of which nodes were signed. For example:

```
<soap:Envelope xmlns:soap="...">
  <soap:Header>
  <wsse:Security xmlns:wsse="...">
    <dsig:Signature xmlns:dsig="..." Id="Id-00000112e2c98df8-0000000000000004">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-00000112e2c98df8-0000000000000003">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>xChPoiWJJrrPZkbXN8FPB8S4U7w=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>KG4N .... /9dw==</dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-00000112e2c98df8-0000000000000005">
        <dsig:X509Data>
          <dsig:X509Certificate>
            MIID ... ZiBQ==
          </dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
  </wsse:Security>
 </soap:Header>
 <soap:Body xmlns:wsu="..." wsu:Id="Id-00000112e2c98df8-0000000000000003">
  <vs:getProductInfo xmlns:vs="http://ww.oracle.com">
   <vs:Name>API Gateway</vs:Name>
   <vs:Version>11.1.2.1.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

In the above example, a `wsu:Id` attribute has been inserted into the `<soap:Body>` element. This `wsu:Id` attribute is then referenced by the `URI` attribute of the `<dsig:Reference>` element in the actual Signature. When the Signature is being verified, the value of the `URI` attribute can be used to locate the nodes that have been signed.

- *Id*
  Select the `Id` option to use generic Ids (not bound to the WSU namespace) to dereference the signed data. Under this schema, the `URI` attribute of the `<Reference>` points at an Id attribute, which is inserted into the top-level node of the signed nodeset. In the following example, the Id specified in the Signature matches the Id attribute inserted into the `<Body>` element, indicating that the Signature applies to the entire contents of the SOAP Body:

```
<soap:Envelope xmlns:soap="....">
  <soap:Header>
    <dsig:Signature xmlns:dsig="...."
                Id="Id-0000011a101b167c-0000000000000013">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-0000011a101b167c-0000000000000012">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>JCy0JoyhVZYzmrLrl92nxfr1+zQ=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>......<dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-0000011a101b167c-0000000000000014">
        <dsig:X509Data>
          <dsig:X509Certificate>......</dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
 </soap:Header>
 <soap:Body Id="Id-0000011a101b167c-0000000000000012">
```

```
  <product version="11.1.2.1.0">
    <name>API Gateway</name>
    <company>oracle</company>
    <description>SOA Security and Management</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

- *ID*
  Select this option to use generic IDs (not bound to the WSU namespace) to dereference the signed data. Under this schema, the URI attribute of the `Reference` points at an ID attribute, which is inserted into the top-level node of the signed nodeset. In the following example, the URI specified in the Signature Reference node matches the ID attribute inserted into the `Body` element, indicating that the Signature applies to the entire contents of the SOAP Body:

```
<soap:Envelope xmlns:soap="....">
  <soap:Header>
    <dsig:Signature xmlns:dsig="....">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-0000011a101b167c-0000000000000012">
          <dsig:Transforms>
            <dsig:Transform
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>JCy0JoyhVZYzmrLrl92nxfr1+zQ=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
       <dsig:SignatureValue>......<dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-0000011a101b167c-0000000000000014">
        <dsig:X509Data>
          <dsig:X509Certificate>......</dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
 </soap:Header>
 <soap:Body ID="Id-0000011a101b167c-0000000000000012">
  <product version="11.1.2.1.0">
    <name>API Gateway</name>
    <company>Oracle</company>
    <description>SOA Security and Management</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

- *xml:id*
  Select this option to use an `xml:id` to dereference the signed data. Under this schema, the URI attribute of the `Reference` points at an `xml:id` attribute, which is inserted into the top-level node of the signed nodeset. In the following example, the URI specified in the Signature Reference node matches the `xml:id` attribute inserted into the `Body` element, indicating that the Signature applies to the entire contents of the SOAP Body:

```
<soap:Envelope xmlns:soap="....">
  <soap:Header>
    <dsig:Signature xmlns:dsig="...."
                Id="Id-0000011a101b167c-0000000000000013">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
```

```
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-0000011a101b167c-0000000000000012">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>JCy0JoyhVZYzmrLrl92nxfr1+zQ=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>......<dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-0000011a101b167c-0000000000000014">
        <dsig:X509Data>
          <dsig:X509Certificate>......</dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
 </soap:Header>
 <soap:Body ID="Id-0000011a101b167c-0000000000000012">
  <product version=11.1.2.1.0>
    <name>API Gateway</name>
    <company>Oracle</company>
    <description>SOA Security and Management</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

- *No id (use with enveloped signature and XPath 'The Entire Document')*
  Select this option to sign the entire document. In this case, the URI attribute on the `Reference` node of the Signature is `“”`, which means that no id is used to refer to what is being signed. The `“”` URI means that the full document is signed. A signature of this type must be an enveloped signature. On the **Advanced** -> **Options** tab, select **Create enveloped signature**. To sign the full document, on the **What to Sign** -> **XPaths** tab, select the XPath named The `entire document`.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="....">
    <soap:Header>
        <wsse:Security
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
               oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
                Id="Id-0001346926985531-ffffffff28f6103-1">
                <dsig:SignedInfo>
                    <dsig:CanonicalizationMethod
                      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                    <dsig:SignatureMethod
                      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                    <dsig:Reference URI="">
                        <dsig:Transforms>
                            <dsig:Transform
                              Algorithm="http://www.w3.org/2000/09/
                                   xmldsig#enveloped-signature" />
                            <dsig:Transform
                              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                        </dsig:Transforms>
                        <dsig:DigestMethod
                            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                        <dsig:DigestValue>
                            BAz3140AFAfBL/DIj9y+16TEJIU=
                        </dsig:DigestValue>
                    </dsig:Reference>
                </dsig:SignedInfo>
                <dsig:SignatureValue>.......</dsig:SignatureValue>
                <dsig:KeyInfo Id="Id-0001346926985531-ffffffff28f6103-2">
                    <dsig:X509Data>
                        <dsig:X509Certificate>........</dsig:X509Certificate>
                    </dsig:X509Data>
                </dsig:KeyInfo>
            </dsig:Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
       <product version=11.1.2.1.0>
           <name>API Gateway</name>
           <company>Oracle</company>
           <description>SOA Security and Management</description>
       </product>
    </soap:Body>
</soap:Envelope>
```

Use SAML Ids for SAML Elements:
This option is only relevant if a SAML assertion is required to be signed. If this option is selected, and the signature is to cover a SAML assertion, an `AssertionID` attribute is inserted into a SAML version 1.1 assertion, or an `ID` attribute is inserted into a SAML version 2.0 assertion. The value of this attribute is then referenced from within a <Reference> block of the XML Signature. This option is selected by default.

Add and Dereferece Security Token Reference for SAML:
This option is only relevant if a SAML assertion is required to be signed. This setting signs the SAML assertion using a Security Token Reference and an STR-Transform. The `Signature` points to the id of the `wsse:SecurityTokenReference`, and applies the STR-Transform. When signing the SAML assertion, this means to sign the XML that the `wsse:SecurityTokenReference` points to, and not the `wsse:SecurityTokenReference`. This option is unselected by default.

The following shows an example SOAP header:

```
<soap:Envelope xmlns:soap="....">
  <soap:Header>
     <wsse:Security xmlns:wsse="...." xmlns:wsu="....";
         <dsig:Signature xmlns:dsig=".....">
             <dsig:SignedInfo>
               <dsig:CanonicalizationMethod
                   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
               <dsig:SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
               <dsig:Reference
                   URI="#Id-0001347292983847-00000000530a9b1a-1">
                   <dsig:Transforms>
                       <dsig:Transform
                          Algorithm="http://docs.oasis-open.org/wss/2004/01/
                          oasis-200401-wss-soap-message-security-1.0#STR-Transform">
                          <wsse:TransformationParameters>
                             <dsig:CanonicalizationMethod
                               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                          </wsse:TransformationParameters>
                       </dsig:Transform>
                   </dsig:Transforms>
                   <dsig:DigestMethod
                       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                   <dsig:DigestValue>
                       6/aLwABWfS+9UiX7v39sLJw5MaQ=
                   </dsig:DigestValue>
               </dsig:Reference>
             </dsig:SignedInfo>
             <dsig:SignatureValue>
```

```
                ......
             </dsig:SignatureValue>
             <dsig:KeyInfo Id="Id-0001347292983847-00000000530a9b1a-3">
                 <dsig:X509Data>
                     <dsig:X509Certificate>
                       .....
                     </dsig:X509Certificate>
                 </dsig:X509Data>
             </dsig:KeyInfo>
         </dsig:Signature>
         <wsse:SecurityTokenReference
            wsu:Id="Id-0001347292983847-00000000530a9b1a-1">
            <wsse:KeyIdentifier
                ValueType="http://docs.oasis-open.org/wss/
                  oasis-wss-saml-token-profile-1.0#SAMLAssertionID">
                Id-948d50f1504e0f3703e00000-1
            </wsse:KeyIdentifier>
         </wsse:SecurityTokenReference>
         <saml:Assertion xmlns:saml="...."
             IssueInstant="2012-09-10T16:03:03Z"
             Issuer="CN=AAA Certificate Services, O=Comodo CA Limited,
               L=Salford, ST=Greater Manchester, C=GB"
             MajorVersion="1" MinorVersion="1">
             <saml:Conditions NotBefore="2012-09-10T16:03:02Z"
                 NotOnOrAfter="2012-12-18T16:03:02Z" />
             <saml:AuthenticationStatement
                AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
                AuthenticationInstant="2012-09-10T16:03:03Z">
                <saml:Subject>
                    <saml:NameIdentifier
                      Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
                      admin
                    </saml:NameIdentifier>
                    <saml:SubjectConfirmation>
                        <saml:ConfirmationMethod>
                            urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
                        </saml:ConfirmationMethod>
                    </saml:SubjectConfirmation>
                </saml:Subject>
             </saml:AuthenticationStatement>
         </saml:Assertion>
      </wsse:Security>
  </soap:Header>
    ....
</soap:Envelope>
```

Where to Place Signature

Append Signature to Root or SOAP Header:
If the message is a SOAP message, the signature will be inserted into the SOAP `Header` element when this radio button is selected. The XML Signature will be inserted as an immediate child of the SOAP `Header` element. The following example shows a skeleton SOAP message which has been signed using this option:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <ws:Security xmlns:ws="http://schemas.xmlsoap.org/..." s:actor="test">
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/..." id="Sample">
     ...
   </dsig:Signature>
  </ws:Security>
 </s:Header>
 <s:Body>
  ...
 </s:Body>
</s:Envelope>
```

If, on the other hand, the message is just plain XML, the signature will be inserted as an immediate child of the root element of the XML message. The following example shows a non-SOAP XML message which has been signed using this option:

```
<PurchaseOrder>
 <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
 </dsig:Signature>

 <Items>
 ...
 </Items>
</PurchaseOrder>
```

Place in WS-Security Element for SOAP Actor/Role:
By selecting this option, the XML Signature will be inserted into the WS-Security element identified by the specified SOAP *actor* or *role*. A SOAP actor/role is simply a way of distinguishing a particular WS-Security block from others which may be present in the message. Actors belong to the SOAP 1.1 specification, but were replaced in SOAP 1.2 by roles. Conceptually, however, they are identical.

Enter the name of the SOAP actor or role of the WS-Security block in the dropdown. The following SOAP message contains an XML Signature within a WS-Security block identified by the "test" actor:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <ws:Security xmlns:ws="http://schemas.xmlsoap.org/..." s:actor="test">
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/..." id="Sample">
       ...
   </dsig:Signature>
  </ws:Security>
 </s:Header>
 <s:Body>
  ...
 </s:Body>
</s:Envelope>
```

Use XPath Location:
This option is useful in cases where the signature must be inserted into a non-SOAP XML message. In such cases, it is possible to insert the signature into a location pointed to by an XPath expression. Select or add an XPath expression in the field provided, and then specify whether the API Gateway should insert the signature *before* the location to which the XPath expression points, or *append* it to this location.

Advanced

The **Advanced** tab enables you to set the following:

- Additional elements from the message to be signed.
- Algorithms and ciphers used to sign the message parts.
- Various advanced options on the generated XML Signature.

Additional

The **Additional** tab allows you to select additional elements from the message that are to be signed. It is also possible to insert a WS-Security Timestamp into the XML Signature, if necessary.

Additional Elements to Sign:
The options here allow you to select other parts of the message that you may wish to sign.

- **Sign KeyInfo Element of Signature:**
  The <KeyInfo> block of the XML Signature can be signed to prevent people cut-and-pasting a different <KeyInfo> block into the message, which may point to some other key material, for example.
- **Sign Timestamp:**
  As stated earlier, timestamps are used to prevent replay attacks. However, to guarantee the end-to-end integrity of the timestamp, it is necessary to sign it.

  ### Note

  This option is only enabled when you have elected to insert a Timestamp into the message using the relevant fields on the **Timestamp Options** panel below.
- **Sign Attachments:**
  In addition to signing some or all of the contents of the SOAP message, it is also possible to sign attachments to the SOAP message. To sign all attachments, check the **Include Attachments** checkbox.
  A signed attachment is referenced in an XML Signature using the *Content-Id* or *cid* of the attachment. The `URI` attribute of the `Reference` element corresponds to this Content-Id. The following example shows how an XML Signature refers to a sample attachment. It shows the wire format of the message and its attachment as they are sent to the destination Web Service. Multiple attachments will result in successive `Reference` elements.

```
POST /myAttachments HTTP/1.0
Content-Length: 1000
User-Agent: API Gateway
Accept-Language: en
Content-Type: multipart/related; type="text/xml";
              boundary="----=Multipart-SOAP-boundary"

------=Multipart-SOAP-boundary
Content-Id: soap-envelope
SOAPAction: none
Content-Type: text/xml; charset="utf-8"

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
    <dsig:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
    <dsig:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <dsig:Reference URI="cid:moredata.txt">...</dsig:Reference>
   </dsig:SignedInfo>
  </dsig:Signature>
 </s:Header>
 <s:Body>
    ...
 </s:Body>
</s:Envelope>

------=Multipart-SOAP-boundary
Content-Id: moredata.txt
Content-Type: text/plain; charset="UTF-8"

Some more data.
```

```
------=Multipart-SOAP-boundary--
```

Transform:
This dropdown is only available when you have selected the **Sign Attachments** box above. It determines the transform used to reference the signed attachments.

Timestamp Options:
It is possible to insert a timestamp into the message to indicate when exactly the signature was generated. Consumers of the signature can then validate the signature to ensure that it is not of date.

The following options are available:

- **No Timestamp:**
  No timestamp is inserted into the signature.
- **Embed in WSSE Security:**
  The `wsu:Timestamp` is inserted into a `wsse:Security` block. The `Security` block is identified by the SOAP actor/role specified on the **Signature** tab.
- **Embed in Signature Property:**
  The `wsu:Timestamp` is placed inside a signature property element in the `dsig:Signature`.

The **Expires In** fields enable the user to optionally specify the `wsu:Expires` for the `wsu:Timestamp`. If all fields are left at `0`, no `wsu:Expires` element is placed inside the `wsu:Timestamp`.

The following examples shows a `wsu:Timestamp` that has been inserted into a `wsse:Security` block:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
   <wsse:Security>
     <wsu:Timestamp wsu:Id="Id-0000011294a0311e-000000000000003d">
      <wsu:Created>2007-05-16T11:22:45Z</wsu:Created>
      <wsu:Expires>2007-05-23T11:22:45Z</wsu:Expires>
    </wsu:Timestamp>
    <dsig:Signature ...>
    ...
    </dsig:Signature ...>
   </wsse:Security>
 </s:Header>
 <s:Body>
    ...
 </s:Body>
</s:Envelope>
```

Algorithm Suite

The fields on this tab determine the combination of cryptographic algorithms and ciphers that are used to sign the message parts.

Algorithm Suite:
WS-Security Policy defines a number of *algorithm suites* that group together a number of cryptographic algorithms. For example, a given algorithm suite will use specific algorithms for asymmetric signing, symmetric signing, asymmetric key wrap, and so on. Therefore, by specifying an algorithm suite, you are effectively selecting a whole suite of cryptographic algorithms to use.

If you want to use a particular WS-Security Policy algorithm suite, you can select it here. The **Signature Method**, **Key Wrap Algorithm**, and **Digest Method** fields will then be automatically populated with the corresponding algorithms for that suite.

Signature Method:
The **Signature Method** field enables you to configure the method used to generate the signature. Various strengths of the HMAC-SHA1 algorithms are available from the dropdown.

Key Wrap Algorithm:
Select the algorithm to use to wrap (encrypt) the symmetric signing key. This option need only be configured when you are using a symmetric key to sign the message.

Digest Algorithm:
Select the digest algorithm to you to produce a cryptographic hash of the signed data.

Options

This tab enables you to configure various advanced options on the generated XML Signature. The following fields can be configured on this tab:

WS-Security Options:
WSSE 1.1 defines a `<SignatureConfirmation>` element that can be used as proof that a particular XML Signature was processed. A recipient and verifier of an XML Signature must generate a `<SignatureConfirmation>` element for each piece of data that was signed (for each `<Reference>` in the XML Signature). A `<SignatureConfirmation>` element contains the hash of the signed data and must be signed by the recipient before returning it in the response to the initiator (the original signatory of the data).

When the initiator receives the `<SignatureConfirmation>` elements in the response, it compares the hash with the hash of the data that it produced initially. If the hashes match, the initiator knows that the recipient has processed the same signature.

Select the **Initiator** option here if the API Gateway is the initiator as outlined in the scenario above. The API Gateway will keep a record of the signed data and will compare it to the contents of the `<SignatureConfirmation>` elements returned from the recipient in the response message.

Alternatively, if the API Gateway is acting as the recipient in this transaction, you can select the **Responder** radio button to instruct the API Gateway to generate the `<SignatureConfirmation>` elements and return them to the inititor. The signature confirmations will be added to the WS-Security header.

Layout Type:
Select the WS-SecurityPolicy layout type that you want the XML Signature and any generated tokens to adhere to. This includes elements such as `<Signature>`, `<BinarySecurityToken>`, and `<EncryptedKey>`, which can all be generated as part of the signing process.

Fail if No Nodes to Sign:
Check this option if you want the filter to fail if it cannot find any nodes to sign as configured on the **What to Sign** tab.

Add Inclusive Namespaces for Exclusive Canonicalization:
You can include information about the namespaces (and their associated prefixes) of signed elements in the signature itself. This ensures that namespaces that are in the same scope as the signed element, but not directly or visibly used by this element, are included in the signature. This ensures that the signature can be validated as a standalone entity outside of the context of the message from which it was extracted.

It is also worth pointing out that the WS-I specification only permits the use of exclusive canonicalization in an XML Signature. The `<InclusiveNamespaces>` element is an attempt to take advantage of some of the behavior of *inclusive* canonicalization, while maintaining the simplicity of *exclusive canonicalization*.

A *PrefixList* attribute is used to list the prefixes of in-scope, but not visibly used elements and attributes. The following example shows how the `PrefixList` attribute is used in practice:

```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope'>
 <soap:Header>
```

```
  <wsse:Security xmlns:wsse='http://docs.oasis-open.org/...'
                 xmlns:wsu='http://docs.oasis-open.org/...'>
   <wsse:BinarySecurityToken wsu:Id='SomeCert'
                             ValueType="http://docs.oasis-open.org/...">
   lui+Jy4WYKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sH
   </wsse:BinarySecurityToken>
   <ds:Signature xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:SignedInfo>
     <ds:CanonicalizationMethod
         Algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
      <c14n:InclusiveNamespaces
          xmlns:c14n='http://www.w3.org/2001/10/xml-exc-c14n#'
          PrefixList='wsse wsu soap' />
     </ds:CanonicalizationMethod>
     <ds:SignatureMethod
          Algorithm='http://www.w3.org/2000/09/xmldsig#rsa-sha1'/>
     <ds:Reference URI=''>
      <ds:Transforms>
       <dsig:XPath xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:m='http://example.org/ws'>
       //soap:Body/m:SomeElement
       </dsig:XPath>
       <ds:Transform Algorithm='http://www.w3.org/2001/10/xml-exc-c14n#'>
        <c14n:InclusiveNamespaces
              xmlns:c14n='http://www.w3.org/2001/10/xml-exc-c14n#'
              PrefixList='soap wsu test' />
       </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm='http://www.w3.org/2000/09/xmldsig#sha1' />
      <ds:DigestValue>VEPKwzfPGOxh2OUpoK0bcl58jtU=</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>+diIuEyDpV7qxVoUOkb5rj61+Zs=</ds:SignatureValue>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI='#SomeCert' />
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>
  </wsse:Security>
 </soap:Header>
 <soap:Body xmlns:wsu='http://docs.oasis-open.org/...'
            xmlns:test='http://www.test.com' wsu:Id='TheBody'>
  <m:SomeElement xmlns:m='http://example.org/ws' attr1='test:fdwfde' />
 </soap:Body>
</soap:Envelope>
```

Indent:
Select this method to ensure that the generated signature is properly indented.

Create Enveloped Signature:
By selecting this option, an enveloped XML Signature is generated. The following skeleton signed SOAP message shows the enveloped signature:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" id="Sample">
 <ds:SignedInfo>
  <ds:Reference URI="">
   <ds:Transforms>
    <ds:Transform
        Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
   </ds:Transforms>
  </ds:Reference>
 </ds:SignedInfo>
```

```
</ds:Signature>
```

This indicates to the application validating the signature that the signature itself should not be included in the signed data. In other words, to validate the signature, the application must first strip out the signature. This is necessary in cases where the entire SOAP envelope has been signed, and the resulting signature has been inserted into the SOAP header. In this case, the signature is over a nodeset which has been altered (the Signature has been inserted), and so the signature will break.

Insert CarriedKeyName for EncryptedKey:
Select this option to include a `<CarriedKeyName<` element in the `<EncryptedKey>` block that is generated when using a symmetric signing key.

XML Signature Verification

Overview

In addition to validating XML Signatures for authentication purposes, the API Gateway can also use XML Signatures to prove message integrity. By signing an XML message, a client can be sure that any changes made to the message do not go unnoticed by the API Gateway. Therefore by validating the XML Signature on a message, the API Gateway can guarantee the *integrity* of the message.

Before configuring the **XML Signature Verification** filter, enter an appropriate name for this filter in the **Name** field.

Signature Verification

The following sections are available on the **Signature Verification** tab:

Signature Location:
Because there may be multiple signatures contained in the message, you must specify which signature the API Gateway uses to verify the integrity of the message. The signature can be extracted from one of the following:

- From the SOAP header
- Using WS-Security Actors
- Using XPath

Select the appropriate option from the drop-down list.

For details on all the configuration options available in this section, see the *Signature Location* topic.

Find Signing Key
The public key used to verify the signature can be taken from the following locations:

- **Via KeyInfo in Message:**
  Typically, a `<KeyInfo>` block is used in an XML Signature to reference the key used to sign the message. For example, it is common for a `<KeyInfo>` block to reference a `<BinarySecurityToken>` that contains the certificate associated with the public key used to verify the signature.
- **Via Selector Expression:**
  The certificate used to verify the signature can be extracted from a selector expression. For example, a previous filter (for example, a **Find Certificate** filter) may have already located a certificate and populated the `certificate` message attribute. If you wish to use this certificate to verify the signature, specify the selector expression in the field provided (for example, `${certificate}`). Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.
- **Via Certificate in LDAP:**
  Clients may not always want to include their public keys in their signatures. In such cases, the public key can be retrieved from a specified LDAP directory. To do this, select the **Via Certificate in LDAP** radio button, and select a previously configured LDAP directory from the drop-down list. You can add LDAP connections under the **External Connections** node in the Policy Studio tree. Right-click the **LDAP Connection** tree node, and select **Add an LDAP Connection**.
- **Via Certificate in Store:**
  Similarly, you can retrieve a certificate from the Certificate Store by selecting this option, and clicking the **Select** button. Select the checkbox next to the certificate that contains the public key that you want to use to verify the signature, and click **OK**.

What Must Be Signed

This section defines the content that must be signed for a SOAP message to pass the filter. This ensures that the client has signed something meaningful (part of the SOAP message), instead of arbitrary data that would pass a blind signature validation. This further strengthens the integrity verification process.

The nodeset that must be signed can be identified by a combination of XPath expressions, node locations, and/or the contents of a message attribute. For more details on how to configure this section, see the *What To Sign* topic.

Note

If all attachments are required to be signed, select the **All attachments** checkbox to enforce this.

Advanced

The following advanced configuration options are available:

Signature Confirmation:
If this filter is configured as part of an Initiator policy, where the API Gateway acts as the client in a Web Services transaction, select the **Initiator** option. This means that the filter keeps a record of the Signature that it has verified, and checks the `<SignatureConfirmation>` returned by the Recipient.

Alternatively, if the API Gateway acts as the Recipient in the transaction, select the **Recipient** option. In this case, the API Gateway returns the `<SignatureConfirmation>` elements in the response to the Initiator.

Default Derived Key Label:
If the API Gateway consumes a `<DerivedKeyToken>`, the default value entered is used to recreate the derived key.

Algorithm Suite:
Select the WS-Security Policy *Algorithm Suite* that must have been used when signing the message. This check ensures that the appropriate algorithms were used to sign the message.

Fail if No Signatures to Verify:
Select this option if you want to configure the filter to fail if no XML Signatures are present in the incoming message.

Verify Signature for Authentication Purposes:
You can use the **XML Signature Verification** filter to authenticate an end user. If the message can be successfully validated, it proves that only the private key associated with the public key used to verify the signature was used to sign the message. Because the private key is only accessible to its owner, a successful verification can be used to effectively authenticate the message signer.

Retrieve DOM using Selector Expression:
You can configure this field to verify the response from a SAML PDP. When the API Gateway receives a response from the SAML PDP, it stores the signature on the response in a message attribute. You can specify this attribute using a selector expression to verify this signature. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Remove enclosing WS-Security element on successful verification:
Select this checkbox if you wish to remove the enclosing WS-Security block when the signature has been successfully verified. This setting is not selected by default.

PGP Sign

Overview

You can use the **PGP Sign** filter to digitally sign messages passing through the API Gateway pipeline. Messages signed on the API Gateway can be verified by the recipient by validating the signature using a public PGP key. Signed messages received at the API Gateway can be verified in the same manner. The **PGP Sign** filter supports the following signing methods:

Compressed	Compresses the message and creates a hash of the contents before signing. Because the message is contained in the signature, the signature can be used in place of the message. The typical use of this method produces a signature in printable ASCII form (ASCII Armor output). You can deselect this option to produce a binary signature.
Clear Signed	Clear signing a message leaves the message intact and adds the signature under the clear message text. This provides for optional verification of the message signature and contents. The output has a content type of `application/pgp-signature`. **Note** It is not possible to clear sign binary objects.
Detached Signature (MIME)	Creates a multipart MIME document where the message remains in clear text and the signature is attached as a MIME part.

For an example use case, see the *PGP Verify* filter.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

PGP Private Key to be retrieved from one of the following locations:
Select how the private PGP key is retrieved to sign the message:

Use the following private key from the PGP Key Pair list	Click the browse button on the right, and select a PGP key pair configured in the Certificate Store. For details on configuring PGP key pairs, see the topic on *Certificates and Keys*.
Look up the private key using the following alias	Enter the alias name of the PGP private key used in the Certificate Store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression that specifies the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).

The following message attribute will contain the private key	Enter a selector expression that specifies the name of the message attribute that contains the private key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.private.key}`).

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Signing Method:
Select the method used to create the digital signature for the message attachment:

Compressed	Creates a compressed signature. Because the message is contained in the signature, this signature is used in place of the message. You can use the **ASCII Armor Output** setting to specify whether to output the binary message data as printable ASCII Armor text. This option is selected by default.
Clear signed	A clear signed message has the message intact and a signature attached under the clear message text. This is useful when the software reading the message does not understand the PGP structure, because it can still display the signed content, but without verifying the signature.
Detached signature (MIME)	Creates a multipart MIME document where the message is in clear text and the signature is attached as a MIME part. Similar to clear signed, this is useful when the software reading the message does not understand the PGP structure.

PGP Verify

Overview

You can use the **PGP Verify** filter to verify Pretty Good Privacy (PGP) signed messages passing through the API Gateway pipeline. PGP signed messages received at the API Gateway can be verified by validating the signature using the public PGP key of the message signer. The **PGP Verify** filter supports the following signing methods:

Compressed	Verifies a compressed signature. Because the message is contained in the signature, this signature is used in place of the message.
Clear Signed	A clear signed message has the message intact and a signature attached under the clear message text. Verifying this message verifies the sender and the message integrity.
Detached Signature (MIME)	Verifies a multipart MIME document where the message is in clear text and the signature is attached as a MIME part.

An example use case for this filter would be when files are sent to the API Gateway over Secure Shell File Transfer Protocol (SFTP) in PGP signed format. The API Gateway can use the **PGP Verify** filter to verify the message, and then use Threat Detection filters to perform virus scanning. The clean files can be PGP signed again using the **PGP Sign** filter before being sent over SFTP to their target destination. For more details, see the *PGP Sign* filter.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

PGP Public Key to be retrieved from one of the following locations:
Select how the sender's public PGP key is retrieved to verify the message:

Use the following public key from the PGP Key Pair list	Click the browse button on the right, and select a PGP key pair configured in the Certificate Store. For details on configuring PGP key pairs, see the topic on *Certificates and Keys*.
Look up the public key using the following alias	Enter the alias name of the PGP public key used in the Certificate Store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression that specifies the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
The following message attribute will contain the public key	Enter a selector expression that specifies the name of the message attribute that contains the public key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.public.key}`).

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Signing Method:
Select the signing method that was used to create the digital signature for the message attachment:

- Compressed
- Clear Signed
- Detached Signature (MIME)

For details on creating PGP digital signatures, see the topic on the *PGP Sign* filter.

Content type:
Enter the `Content-Type` of the verified message data. Defaults to `application/octet-stream`.

SMIME Sign

Overview

You can use the **SMIME Sign** filter to digitally sign a multipart message as it passes through the API Gateway core pipeline. The recipient of the message can then verify the integrity of the SMIME message by validating the Public Key Cryptography Standards (PKCS) #7 signature.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

Sign Using Key:
Select the checkbox next to the certificate that contains the public key associated with the private signing key that you wish to use to sign the message.

Create Detached Signature in Attachment:
Specifies whether to create a detached digital signature in the message attachment. This is selected by default. For example, this is useful when the software reading the message does not understand the PKCS#7 binary structure, because it can still display the signed content, but without verifying the signature.

If this is unselected, the message content is embedded with the PKCS#7 binary signature. This means that user agents that do not understand PKCS#7 can not display the signed content. Intermediate systems between the sender and final recipient may modify the text content slightly (for example, line wrapping, whitespace, or text encoding). This may cause the message to fail signature validation due to changes in the signed text that are not malicious, nor necessarily affecting the meaning of the text.

SMIME Verify

Overview

You can use the **SMIME Verify** filter to check the integrity of a Secure/Multipurpose Internet Mail Extensions (SMIME) message. This filter enables you to verify the Public Key Cryptography Standards (PKCS) #7 signature over the message.

You can select the certificates that contain the public keys that you wish to use to verify the signature. Alternatively, you can specify a message attribute that contains the certificate with the public key that you wish to use.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for the filter.

Certificates from the following list:
Select the certificates that contain the public keys that you wish to use to verify the signature. This is the default option.

Certificate in attribute:
Alternatively, enter the message attribute that specifies the certificate that contains the public key that you wish to use to verify the signature. Defaults to `${certificate}`.

Remove Outer Envelope if Verification is Successful:
Select this option if you want to remove the PKCS#7 signature and all its associated data from the message if it verifies successfully.

Generic Error

Overview

In cases where a transaction fails, the API Gateway can use a *Generic Error* to convey error information to the client based on the message type (for example, SOAP or JSON). By default, the API Gateway returns a very basic error to the client when a message filter fails. You can add the **Generic Error** filter to a policy to return more meaningful error information to the client based on the message type.

When the **Generic Error** filter is configured, the API Gateway examines the incoming message and attempts to infer the type of message to be returned. For example, for an incoming SOAP message, the API Gateway sends an appropriate SOAP response (for example, SOAP 1.1 or 1.2) using the SOAP fault processor. For an incoming JSON message, the API Gateway sends an appropriate JSON response. If the inference process fails, the API Gateway sends a SOAP message by default. For example error messages, see the *JSON Error* and *SOAP Fault* topics.

You can also transform the error message returned by applying an XSLT stylesheet. The API Gateway implicitly transforms the incoming message into XML before applying the stylesheet to the message.

Important

For security reasons, it is good practice to return as little information as possible to the client. However, for diagnostic reasons, it is useful to return as much information to the client as possible. Using the **Generic Error** filter, administrators have the flexibility to configure just how much information to return to clients, depending on their individual requirements.

General Configuration

Configure the following general settings:

Name:
Enter an appropriate name for this filter.

HTTP Response Code Status
Enter the HTTP response code status for this **Generic Error** filter. This ensures that a meaningful response is sent to the client in the case of an error occurring in a configured policy. Defaults to `500` (`Internal Server Error`). For a complete list of status codes, see the HTTP Specification [http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html].

Generic Error Contents

The following configuration options are available in this section:

Show detailed explanation of error:
If this option is selected, a detailed explanation of the Generic Error is returned in the error message. This makes it possible to suppress the reason for the exception in a tightly locked down system (the reason is displayed as `message blocked` in the Generic Error). Defaults to the value of the `${circuit.failure.reason}` message attribute selector.

Show filter execution path
When this option is selected, the API Gateway returns a Generic Error containing the list of filters run on the message before the error occurred. For each filter listed in the Generic Error, the status is given (`pass` or `fail`).

Show stack trace
If this option is selected, the API Gateway returns the Java stack trace for the error to the client. This option should only be enabled under instructions from the Oracle Support Team.

Show current message attributes

By selecting this option, the message attributes present at the time the Generic Error was generated are returned to the client. For example, for an incoming SOAP message, each message attribute forms the content of a `<fault:attribute>` element.

Warning

For security reasons, **Show filter execution path**, **Show stack trace**, and **Show current message attributes** should not be used in a production environment.

Use Stylesheet
Select this option if you wish to transform the error message returned by applying an XSLT stylesheet. Click the browse button on the right of the **Stylesheet** text box, and select a stylesheet in the dialog. To add a stylesheet, right-click the **Stylesheets** node, and select **Add Stylesheet**. Alternatively, you can also add stylesheets under the **Resources** -> **Stylesheets** node in the Policy Studio main menu.

Because XSLT stylesheets accept XML as input, the API Gateway implicitly transforms the incoming message into XML. The API Gateway then retrieves the selected XSLT stylesheet and applies the transformation to the message, and sends the response in the format specified in the XSLT stylesheet.

Using the Set Message Filter
You can also use the **Set Message** filter create customized Generic Errors. The **Set Message** filter can change the contents of the message body to any arbitrary content. When an exception occurs in a policy, you can use this filter to customize the body of the Generic Error. For details on how to use the **Set Message** filter to generate customized faults and return them to the client, see the example in the *SOAP Fault* topic. You can use the same approach to generate customized Generic Errors.

JSON Error

Overview

In cases where a JavaScript Object Notation (JSON) transaction fails, the API Gateway can use a *JSON Error* to convey error information to the client. By default, the API Gateway returns a very basic fault to the client when a message filter has failed. You can add the **JSON Error** filter to a policy to return more meaningful error information to the client. For example, the following message extract shows the format of a JSON Error raised when a JSON Schema Validation filter fails:

```
{
        "reasons": [
            {
                "language": "en",
                "message": "JSON Schema Validation filter failed"
            }
        ],
        "details": {
            "msgId": "Id-f5aab7304f6c754804f70000",
            "exception message": "JSON Schema Validation filter failed",
           ...
        }
}
```

Important

For security reasons, it is good practice to return as little information as possible to the client. However, for diagnostic reasons, it is useful to return as much information to the client as possible. Using the **JSON Error** filter, administrators have the flexibility to configure just how much information to return to clients, depending on their individual requirements.

For more details on JSON schema validation, see the topic on the *JSON Schema Validation* filter. For more details on JSON, see http://www.json.org/index.html.

General Configuration

Configure the following general settings:

Name:
Enter an appropriate name for this filter.

HTTP Response Code Status
Enter the HTTP response code status for this JSON error filter. This ensures that a meaningful response is sent to the client in the case of an error occurring in a configured policy. Defaults to `500` (`Internal Server Error`). For a complete list of status codes, see the HTTP Specification [http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html].

JSON Error Contents

The following configuration options are available in this section:

Show detailed explanation of error:
If this option is selected, a detailed explanation of the JSON Error is returned in the error message. This makes it possible to suppress the reason for the exception in a tightly locked down system. By default, the reason is displayed as `message blocked` in the JSON Error. This option displays the value of the `${circuit.failure.reason}` message attribute selector.

Show filter execution path

When this option is selected, the JSON Error returned by the API Gateway contains the list of filters run on the message before the error occurred. For each filter listed in the JSON Error, the status is output (`Pass` or `Fail`). The following message extract shows a *filter execution path* returned in a JSON Error:

```
"path" : {
      "policy" : "test_policy",
      "filters" : [ {
        "name" : "True Filter",
        "status" : "Pass"
      }, {
        "name" : "JSON Schema Validation",
        "status" : "Fail",
        "filterMessage" : "Filter failed"
      }, {
        "name" : "Generic Error",
        "status" : "Fail",
        "filterMessage" : "Filter failed"
      } ]
    },
```

Show stack trace

If this option is selected, the API Gateway returns the Java stack trace for the error to the client. This option should only be enabled under instructions from the Oracle Support Team.

Show current message attributes

By selecting this option, the message attributes present when the JSON Error is generated are returned to the client. The value of each message attribute is output as shown in the following example:

```
"attributes": [
        {
            "name": "circuit.exception",
            "value": "com.vordel.circuit.CircuitAbortException: JSON Schema Validation
            filter failed"
        },
        {
            "name": "circuit.failure.reason",
            "value": "JSON Schema Validation filter failed"
        },
        {
            "name": "content.body",
            "value": "com.vordel.mime.JSONBody@185afba1"
        },
        {
            "name": "failure.reason",
            "value": "JSON Schema Validation filter failed"
        },
        {
            "name": "http.client",
            "value": "com.vordel.dwe.http.ServerTransaction@7d3e1384"
        },
        {
            "name": "http.headers",
            "value": "com.vordel.mime.HeaderSet@76737f58"
        },
        {
            "name": "http.response.info",
            "value": "ERROR"
        },
        {
            "name": "http.response.status",
            "value": "500"
        },
        {
            "name": "id",
            "value": "Id-f5aab7304f6c754804f70000"
        },
        {
            "name": "json.errors",
            "value": "org.codehaus.jackson.JsonParseException: Unexpected character
            ('\"' (code 34)): was expecting comma to separate OBJECT entries\n at [Source:
            com.vordel.dwe.InputStream@592c34b; line: 3, column: 25]"
        },
        ...
]
```

Warning

For security reasons, **Show filter execution path**, **Show stack trace**, and **Show current message attributes** should not be used in a production environment.

Customized JSON Errors

You can use the following approaches to create customized JSON errors:

Using the Generic Error Filter
Instead of using the **JSON Error** filter, you can use the **Generic Error** filter to transform the JSON error message returned by applying an XSLT stylesheet. The **Generic Error** filter examines the incoming message and infers the type of message to be returned (for example, JSON or SOAP). For more details, see the *Generic Error* topic.

Using the Set Message Filter
You can create customized JSON Errors using the **Set Message** filter with the *JSON Error* filter. The **Set Message** filter can change the contents of the message body to any arbitrary content. When an exception occurs in a policy, you can use this filter to customize the body of the JSON Error. For details on how to use the **Set Message** filter to generate customized faults and return them to the client, see the example in the *SOAP Fault* topic. You can use the same approach to generate customized JSON Errors.

SOAP Fault

Overview

In cases where a typical SOAP transaction fails, a *SOAP Fault* can be used to convey error information to the SOAP client. The following message shows the format of a SOAP Fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>Receiver</env:Value>
    <env:Subcode>
     <env:Value>policy failed</env:Value>
    </env:Subcode>
   </env:Code>
   <env:Detail xmlns:oraclefault="http://www.oracle.com/soapfaults"
               oraclefault:type="exception" type="exception"/>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

By default, the API Gateway returns a very basic SOAP Fault to the client when a message filter has failed. You can add the **SOAP Fault** processor to a policy to return more complicated error information to the client.

For security reasons, it is good practice to return as little information as possible to the client. However, for diagnostic reasons, it is useful to return as much information to the client as possible. Using the **SOAP Fault** processor, administrators have the flexibility to configure just how much information to return to clients, depending on their individual requirements.

SOAP Fault Format

The following configuration options are available in this section:

SOAP Version:
You can send either a SOAP Fault 1.1 or 1.2 response to the client. Select the appropriate version using the radio buttons provided.

Fault Namespace:
Select the default namespace to use in SOAP Faults, or enter a new one if necessary.

Indent SOAP Fault:
If this option is selected, an XSL stylesheet is run over the SOAP Fault to indent nested XML elements. The indented SOAP Fault is returned to the client.

SOAP Fault Contents

The following configuration options are available in this section:

Show Detailed Explanation of Fault:
If this option is selected, a detailed explanation of the SOAP Fault is returned in the fault message. This makes it possible to suppress the reason for the exception in a tightly locked down system (the reason is displayed as `message blocked` in the SOAP Fault).

Show Filter Execution Path
When this option is selected, the API Gateway returns a SOAP Fault containing the list of filters run on the message be-

fore the error occurred. For each filter listed in the SOAP Fault, the status is given (`pass` or `fail`). The following message shows a *filter execution path* returned in a SOAP Fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>Receiver</env:Value>
    <env:Subcode>
     <env:Value>policy failed</env:Value>
    </env:Subcode>
   </env:Code>
   <env:Detail xmlns:oraclefault="http://www.oracle.com/soapfaults"
   oraclefault:type="exception" type="exception">
    <oraclefault:path>
     <oraclefault:visit node="HTTP Parser" status="Pass"></oraclefault:visit>
     <oraclefault:visit node="/services" status="Fail"></oraclefault:visit>
     <oraclefault:visit node="/status" status="Fail"></oraclefault:visit>
    </oraclefault:path>
   </env:Detail>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

Show Stack Trace
If this option is selected, the API Gateway returns the Java stack trace for the error to the client. This option should only be enabled under instructions from the Oracle Support Team.

Show Current Message Attributes
By selecting this option, the message attributes present at the time the SOAP Fault was generated are returned to the client. Each message attribute forms the content of a `<fault:attribute>` element, as shown in the following example:

```
<fault:attributes>
  <fault:attribute name="circuit.failure.reason" value="null">
  <fault:attribute name="circuit.lastProcessor" value="HTTP Digest">
  <fault:attribute name="http.request.clientaddr" value="/127.0.0.1:4147">
  <fault:attribute name="http.response.status" value="401">
  <fault:attribute name="http.request.uri" value="/authn">
  <fault:attribute name="http.request.verb" value="POST">
  <fault:attribute name="http.response.info" value="Authentication Required">
  <fault:attribute name="circuit.name" value="Digest AuthN">
</fault:attributes>
```

Customized SOAP Faults

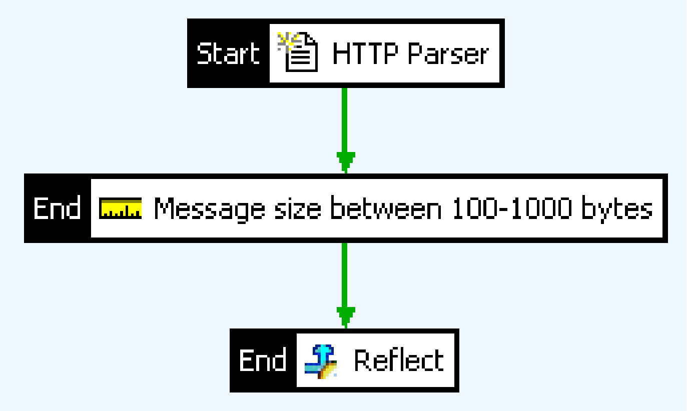
You can use the following approaches to create customized SOAP faults:

Using the Generic Error Filter
Instead of using the **SOAP Fault** filter, you can use the **Generic Error** filter to transform the SOAP fault message returned by applying an XSLT stylesheet. The **Generic Error** filter examines the incoming message and infers the type of message to be returned (for example, JSON or SOAP). For more details, see the *Generic Error* topic.

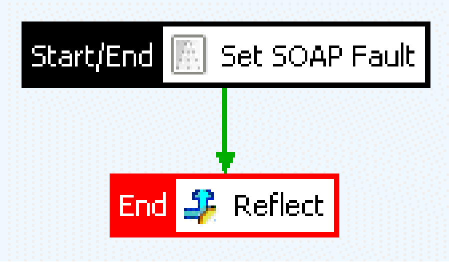
Using the Set Message Filter
You can create customized SOAP Faults using the **Set Message** filter with the **SOAP Fault** filter. The **Set Message** filter can change the contents of the message body to any arbitrary content. When an exception occurs in a policy, you can use this filter to customize the body of the SOAP fault. The following example demonstrates how to generate customized

SOAP faults and return them to the client.

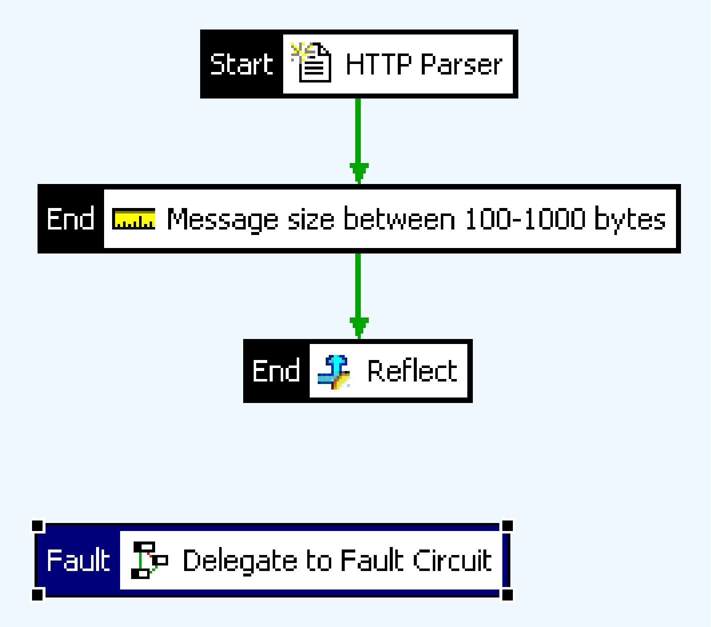
Step 1: Create the Top-level Policy:
This example first creates a very simple policy called **Main Policy**. This policy ensures the size of incoming messages is between 100 and 1000 bytes. Messages in this range are echoed back to the client.



Step 2: Create the Fault Policy
Next, create a second policy called **Fault Circuit**. This policy uses the **Set Message** filter to customize the body of the SOAP fault. When configuring this filter, enter the contents of the customized SOAP fault that you want to return to clients in the text area provided.



Step 3: Create a shortcut to the Fault Policy
Add a **Policy Shortcut** filter to the **Main Policy** and configure it to refer to the **Fault Circuit**. Do ***not*** connect this filter to the policy. Instead, right-click the filter, and select the **Set as Fault Handler** menu option. The **Main Policy** is displayed as follows:


Start
HTTP Parser
End
Message size between 100-1000 bytes
End
Reflect
Fault
Delegate to Fault Circuit

So how does it work? Assume a 2000-byte message is received by the API Gateway and is passed to the **Main Policy** for processing. The message is parsed by the **HTTP Parser** filter, and the size of the message is checked by the **Message Size** filter. Because the message is greater than the size constraints set by this filter, and because there is no failure path configured for this filter, an exception is thrown.

When an exception is thrown in a policy, it is handled by the designated *Fault Handler*, if one is present. In the **Main Policy**, a **Policy Shortcut** filter is set as the fault handler. This filter delegates to the **Fault Circuit**, meaning that when an exception occurs, the **Main Policy** invokes (or delegates to) the **Fault Circuit**.

The **Fault Circuit** consists of two filters, which play the following roles:

1. **Set Message:**
   This filter is used to set the body of the message to the contents of the customized SOAP fault.
2. **Reflect:**
   When the SOAP fault has been set to the message body, it is returned to the client using the **Reflect** filter.

System Alerting

Overview

This tutorial shows the API Gateway's enhanced logging capabilities. System alerts are usually sent when a filter fails, but they can also be used for notification purposes. The API Gateway can send system alerts to several alert destinations, including a Windows Event Log, UNIX/Linux sylsog, SNMP Network Management System, Check Point Firewall-1, email recipient, or Twitter.

There are two main steps involved in configuring the API Gateway to send system alerts:

1. Configuring an alert destination
2. Configuring an **Alert** filter

Configuring an Alert Destination

The first step in configuring the API Gateway to send alerts is to configure an *alert destination*. The API Gateway can send alerts to the following destinations:

- Syslog (Local or Remote)
- Windows Event Log
- Check Point FireWall-1 (OPSEC)
- SNMP Network Management System
- Email Recipient
- Twitter

You can configure these alert destinations under the **Libraries** -> **Alerts** node in the Policy Studio tree.

Syslog (Local or Remote)
Many types of UNIX and Linux provide a general purpose logging utility called `syslog`. Both local and remote processes can send logging messages to a centralized system logging daemon, known as `syslog`, which in turn writes the messages to the appropriate log files. You can configure the level of detail at which `syslog` logs information. This enables administrators to centrally manage how log files are handled, rather than separately configuring logging for each process.

Each type of process logs to a different syslog *facility*. There are facilities for the kernel, user processes, authorization processes, daemons, and a number of place-holders that can be used by site-specific processes. For example, the API Gateway enables you to log to facilities such as `auth`, `daemon`, `ftp`, `local0-7`, and `syslog` itself.

Remote Syslog
To configure a remote syslog alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node, and click **Add** -> **Syslog Remote** at the bottom of the screen on the right.
2. The **Syslog Server** dialog enables you to specify details about the machine on which the syslog daemon is running. The API Gateway connects to this daemon and logs to the specified facility when the alert event is triggered. Complete the following fields on the **Syslog Server** dialog:
   - **Name:**
     Enter a name for this alert destination.
   - **Host:**
     Enter the host name or IP address of the machine where the syslog daemon is running.
   - **Facility:**
     Select the facility that the API Gateway sends alerts to from the drop-down list.
3. Click **OK**.

Local Syslog (UNIX only)

To configure a local syslog alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node, and click **Add** -> **Syslog Local (UNIX only)** at the bottom of the screen on the right.
2. The **Syslog Server** dialog enables you to specify where the alert is sent when the alert event is triggered. Complete the following fields on the **Syslog Server** dialog:
   - **Name:**
     Enter a name for this alert destination.
   - **Facility:**
     Select the facility that the API Gateway sends alerts to from the drop-down list.
3. Click **OK**.

Windows Event Log

This alert destination enables alert messages to be written to the local or a remote Windows Event Log. To add a Windows Event Log alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node in the Policy Studio tree, and click **Add** -> **Windows Event Log** at the bottom of the screen on the right.
2. The **Windows Event Log Alerting** dialog enables you to specify the machine of the Event Log the API Gateway sends alerts to. Complete the following fields on this dialog:
   - **Name:**
     Enter a name for this alert destination.
   - **UNC Server name:**
     Enter the UNC (Universal Naming Code) of the machine where the event log resides. For example, to send alerts to the event log running on a machine called `\\NT_SERVER`, enter `\\NT_SERVER` as the UNC name for this host.
3. Click **OK**.

Check Point FireWall-1 (OPSEC)

The API Gateway complies with Open Platform for Security (OPSEC). OPSEC compliance is awarded by Check Point Software Technologies to products that have been successfully integrated with at least one of their products. In this case, the API Gateway has been integrated with the Check Point FireWall-1 product.

FireWall-1 is the industry leading firewall that provides network security based on a security policy created by an administrator. Although OPSEC is not an open standard, the platform is recognized worldwide as the standard for interoperability of network security, and the alliance contains over 300 different companies. OPSEC integration is achieved through a number of published APIs, which enable third-party vendors to interoperate with Check Point products.

To configure a FireWall-1 alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node in the Policy Studio tree, and click **Add** -> **OPSEC** at the bottom of the screen on the right.
2. The **OPSEC Alerting** dialog enables you to specify details about the machine on which FireWall-1 is installed, the port it is listening on, and how to authenticate to the firewall. The API Gateway connects to the specified firewall when the alert event is triggered and prevents further requests for the particular client that triggered the alert. The following configuration settings must be set:
   - **sam_server auth_port:**
     The port number used to establish SIC (Secure Internal Communications) based connections with the firewall.
   - **sam_server auth_type:**
     The authentication method used to connect to the firewall.
   - **sam_server ip:**
     The host name or IP address of the machine that hosts the Check Point Firewall.

- **sam_server opsec_entity_sic_name:**
  The firewall's SIC name.
- **opsec_sic_name:**
  The OPSEC application's SIC Name, which is the application's full DName as defined by the VPN-1 SmartCenter Server.
- **opsec_sslca_file:**
  The name of the file containing the OPSEC application's digital certificate.

3. Click **OK**.

You can store configuration information in a file and then load it using the **Browse** button. Alternatively, you can use the **Template** button to load the required settings into the text area, and add the configuration values manually.

For the API Gateway to establish the SSL connection to the firewall, the `opsec_sslca_file` specified must be uploaded to the API Gateway machine. You can do this by clicking the **Add** button at the bottom of the screen.

For more information on OPSEC settings, see the documentation for your OPSEC application.

SNMP Network Management System
This alert destination enables the API Gateway to send Simple Network Management Protocol (SNMP) traps to a Network Management System (NMS).

To configure an SNMP alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node in the Policy Studio tree, and click **Add** -> **SNMP** at the bottom of the screen on the right.
2. The **SNMP Alerting** dialog enables you to specify details about the NMS that the API Gateway should send an alert to. Complete the following fields:
   - **Host:**
     The host name or IP address of the machine on which the NMS system resides.
   - **Port:**
     The port on which the NMS system is listening.
   - **Timeout:**
     The timeout in seconds for connections from the API Gateway to the NMS system.
   - **Retries:**
     The number of retries that should be attempted whenever a connection failure occurs.
   - **SNMP Version:**
     Select the version of SNMP that you wish to use for this alert.
3. Click **OK**.

Email Recipient
This alert destination enables alert messages to be sent by email. To add a Windows Event Log alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node in the Policy Studio tree, and click **Add** -> **Email** at the bottom of the screen on the right.
2. The **Email Alerting** dialog enables you to configure how the email alert is to be sent. Complete the following fields:
   - **Name:**
     Enter a name for this alert destination.
   - **Email Recipient (To):**
     Enter the recipient of the alert mail in this field. Use a semicolon-separated list of email addresses to send alerts to multiple recipients.
   - **Email Sender (From):**
     Email alerts appear *from* the sender email address specified here.

Important

Some mail servers do not allow relaying mail when the sender in the **From** field is not recognized by the server.

- **Email Subject:**
  Email alerts use the subject specifed in this field.

3. In the **SMTP Server Settings**, specify the following fields:
   - **Outgoing Mail Server (SMTP):**
     Specify the SMTP server that the API Gateway uses to relay the alert email.
   - **Port:**
     Specify the SMTP server port to connect to. Defaults to port 25.
   - **Connection Security:**
     Select the connection security used to send the alert email (`SSL`, `TLS`, or `NONE`). Defaults to `NONE`.
4. If you are required to authenticate to the SMTP server, specify the following fields in **Log on Using**:
   - **User Name:**
     Enter the user name for authentication.
   - **Password:**
     Enter the password for the user name specified.
5. Finally, you can select the **Email Debugging** setting to find out more information about errors encountered by the API Gateway when attempting to send email alerts. All trace files are written to the `/trace` directory of your the API Gateway installation. This setting is not selected by default.
6. Click **OK**.

Twitter
This alert destination enables the API Gateway to send tweet alerts to Twitter. Twitter uses the OAuth open authentication standard. To enable the API Gateway to send tweet alerts using the Twitter API, you first need to do the following:

- Create a Twitter account to represent you as the user
- Register a custom application for your API Gateway instance, which posts alerts on the user's behalf

Twitter requires that API calls are made for both the user and the application. The Twitter API requires the following credentials:

- Consumer Key of registered applications
- Consumer Secret Key of registered application
- Access Token allowing application to post on behalf of a user
- Access Token Secret to verify the Access Token

Twitter uses this information to determine which application is calling the API, and verifies that the Twitter user you are attempting to make API requests on behalf of has authorized access to their account using the specified application. Twitter identifies and authenticates all requests as coming from both the user performing the request and the registered API Gateway application working on the user's behalf.

Registering a client application
To use the Twitter API, you must create a Twitter account, and register a client application for the API Gateway. If you have not already created a Twitter account, register a new account using the instructions on `http://www.twitter.com`. When you have created the account, register a client application for the API Gateway as follows:

1. Go to `http://dev.twitter.com/`.

2. On the **Twitter** toolbar, select **Your apps**.
3. Click the **Register a new app** button.
4. Enter the details for your custom application. Some details are arbitrary, but you must specify the following values:
   - **Application Type**:
     Select the **Client** radio button.
   - **Default Access Type**:
     Select the **Read & Write** radio button.

Note

The **Application Name** may already be registered to another user, so you may need to specify a different unique name.

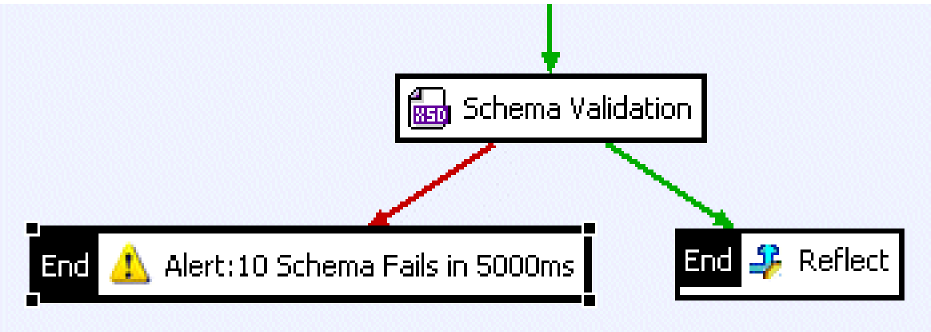
5. Click **Register Application**. Each client application you register is provisioned a consumer key and consumer secret. These are used, in conjunction with the OAuth library, to sign every request you make to the API. Using this signing process, Twitter trusts that the traffic identifying itself as you is indeed you.
6. Select your registered application, and select **My Access Token**. This provides you with an access token and an access token secret. You must store these safely.

Configuring a Twitter alert destination
To configure a Twitter alert destination, perform the following steps:

1. Right-click the **Libraries** -> **Alerts** node in the Policy Studio tree.
2. Click **Add** -> **Twitter** at the bottom of the screen on the right.
3. The **Twitter Alerting** dialog enables you to specify credentials for the Twitter user that the API Gateway uses to send an alert to. Complete the following fields on this dialog:
   - **Consumer Key**:
     The Consumer Key of your registered application.
   - **Consumer Secret**:
     The Consumer Secret of your registered application.
   - **Access Token**:
     The Access Token that represents you.
   - **Access Token Secret**:
     The Access Token Secret that represents you.

Configuring an Alert Filter

Typically, an **Alert** filter is placed on the failure path of another filter in the policy. For example, you could configure an alert if a schema validation fails 10 times within a 5000 millisecond period for a specified Web Service. In this case, you would place the **Alert filter** on the failure path from the **Schema Validation** filter, as shown in the following policy screen shot:



When editing policies, you can drag and drop the **Alert** filter from the **Monitoring** filter group. To configure an alert filter, specify the following settings on the **Alert filter** screen:

Name:
Enter a descriptive name for the filter.

Message:
Configure the following settings on the **Message** tab:

Alert Type	Select the severity level of the alert. This option is only relevant for alert destinations that support severity levels, such as the Windows Event Log.
Alert Message	Enter the text that appears in the alert. You can enter message attributes using selectors, which are looked up and expanded to values at runtime. For example, instead of sending a generic alert stating `Authentication Failed`, you can use a message attribute to give the ID of the user that was not authenticated. To do this, use the following syntax: `${name_of_attribute}` The following examples show how to use message attributes in alert messages: • `Authentication failure for user: ${authentication.subject.id}.` • `{alert.number.failures} authentication failures have occurred in ${alert.time.period} seconds.` • `${alert.number.failures} exceptions have occurred in policy ${circuit.name}.` • `The last exception was ${circuit.exception} with path ${circuit.path}.` **Note** An alert message is not required for alerts sent to an OPSEC firewall.

Tracking:
Configure the following settings on the **Tracking** tab:

Accumulated number of messages	Enter the number of times this filter can be invoked before the alert is sent.
In time period (secs)	Enter the time period in which the accumulated number of messages can occur before and alert is triggered.
Track per client	Select this option if you want to record the accumulated number of messages in the specified time period for each client.

Destination:
All pre-configured alert destinations are displayed in the **Alert Name** table. Select the destinations that the API Gateway sends alerts to if the criteria specified for this filter are met. You can click **Add** to configure an alert destination. For more details, see the section called “Configuring an Alert Destination”.

Audit Log Settings

Overview

One of the most important features of a server-based product is its ability to maintain highly detailed and configurable logging. It is crucial that a record of each and every transaction is kept, and that these records can easily be queried by an administrator to carry out detailed transaction analysis. In recognition of this requirement, the API Gateway provides detailed logging to a number of possible locations.

You can configure the API Gateway so that it logs information about all requests. Such information includes the request itself, the time of the request, where the request was routed to, and the response that was returned to the client. The logging information can be written to the console, log file, local/remote syslog, and/or a database, depending on what is configured in the logging settings.

The API Gateway can also digitally sign the logging information it sends to the log files and the database. This means that the logging information can not be altered after it has been signed, thus enabling an irreversible audit trail to be created.

Configuring Log Output

To edit the default logging settings that ship with the API Gateway, in the Policy Studio main menu, select **Tasks** -> **Manage Settings** -> **Audit Log**. Alternatively, in the Policy Studio tree, select the **Settings** node, and click the **Audit Log** tab at the bottom of the screen. You can configure the API Gateway to log to the following locations.

Log to Text File

To configure the API Gateway to log in text format to a file, click the **Text File** tab, and select the **Enable logging to file** checkbox. You can configure the following fields:

- **File Name:**
  Enter the name of the text-based file that the API Gateway logs to. By default, the log file is called `transactionLog`.
- **File Extension:**
  Enter the file extension of the log file in this field. By default, this has a `.log` extension.
- **Directory:**
  Enter the directory of the log file in this field. By default, all log files are stored in the `/logs/audit` directory of your API Gateway installation.
- **File Size:**
  Enter the maximum size that the log file grows to. When the file reaches the specified limit, a new log file is created. By default, the maximum file size is 1000 kilobytes.
- **Roll Log Daily:**
  Specify whether to roll over the log file at the start of each day. This is enabled by default.
- **Number of Files:**
  Specify the number of log files that are stored. The default number is 20.
- **Format:**
  You can specify the format of the logging output using the values entered here. You can use selectors to output logging information that is specific to the request. The default logging format is as follows:

  ```
  ${level} ${timestamp} ${id} ${text} ${filterType} ${filterName}
  ```

  The available logging properties are described as follows:
  - **`level`:**
    The log level (`fatal`, `fail`, `success`).
  - **`timestamp`:**

The time that the message was processed in user-readable form.

- **`id`:**
  The unique transaction ID assigned to the message.
- **`text`:**
  The text of the log message that was configured in the filter itself. In the case of the **Log Message Payload** filter, the `${payload}` selector contains the message that was sent by the client.
- **`filterName`:**
  The name of the filter that generated the log message.
- **`filterType`:**
  The type of the filter that logged the message.
- **`ip`:**
  The IP address of the client that sent the request.

- **Signing Key:**
  To sign the log file, select a **Signing Key** from the Certificates Store that is used in the signing process. By signing the log files, you can verify their integrity at a later stage.

Log to XML File

To configure the API Gateway to log to an XML file, click the **XML File** tab, and select the **Enable logging to XML file** checkbox.

The log entries are written as the values of XML elements in this file. You can view historical XML log files (not the current file) as HTML for convenience by opening the XML file in your default browser. The `/logs/xsl/MessageLog.xsl` stylesheet is used to render the XML log entries in a more user-friendly HTML format.

You can configure the following fields on the **XML File** tab:

- **File Name:**
  Enter the name of the text-based file that the API Gateway logs to. By default, the log file is called `oracle`.
- **File Extension:**
  Enter the file extension of the log file in this field. By default, the log file is given the `.log` extension.
- **Directory:**
  Enter the directory of the log file in this field. By default, all log files are stored in the `/logs/audit` directory of your API Gateway installation.
- **File Size:**
  Enter the maximum size that the log file grows to. When the file reaches the specified limit, a new log file is created. By default, the maximum file size is 1000 kilobytes.
- **Roll Log Daily:**
  Specify whether to roll over the log file at the start of each day. This is enabled by default.
- **Number of Files:**
  Specify the number of log files that are persisted. The default number is 20.
- **Signing Key:**
  To sign the log file, select a **Signing Key** from the Certificates Store that will be used in the signing process. By signing the log files, you can verify their integrity at a later stage.

Log to Database

Using this option, you can configure the API Gateway to log messages to an Oracle, SQL Server, or MySQL relational database. Before configuring the API Gateway to log to a database, you must first create the tables that the API Gateway writes to. The SQL commands required to set up these tables for each of these databases can be found under the `INSTALL_DIR/system/conf/sql`. This folder contains a directory for each of the Oracle, SQL Server, and MySQL databases, each of which contains the appropriate SQL scripts.

The SQL commands to generate the database table can be found in the `audit_trail.sql` file. Select this file from the appropriate directory (depending on your database), and use the tool of your choice to run the SQL commands contained in the file. For example, to create the logging tables in a MySQL database, you can simply copy and paste the SQL commands into the MySQL command prompt.

When you have set up the logging database tables, you can configure the API Gateway to log to the database. To do this, click the **Database** tab, and select the **Enable logging to database** checkbox. You can configure the following fields on the **Database** tab:

- **Connection:**
  Select an existing database from the **Connection** drop-down list. To add a database connection, click the **External Connections** button on the left, right-click the **Database Connections** tree node, and select **Add a Database Connection**. For more details, see the *Database Connection* topic.
- **Signing Key:**
  You can sign log messages stored in the database to ensure that they are not tampered with. Click the **Signing Key** button to open the list of certificates in the Certificate Store. You can then select the key to use to sign log messages.

Log to Local Syslog

To configure the API Gateway to send logging information to the local UNIX syslog, click the **Local Syslog** tab, and select the **Enable logging to local UNIX Syslog** checkbox. You can configure the following fields:

- **Select Syslog server:**
  Select the local syslog facility that the API Gateway should log to. The default is `LOCAL0`.
- **Format:**
  You can specify the format of the log message using the values (including selectors) entered in this field. For details on the properties that are available, see the section called “Log to Text File”.

Log to Remote Syslog

To configure the API Gateway to send logging information to a remote syslog, click the **Remote Syslog** tab, and select the **Enable logging to Remote Syslog** checkbox. You can configure the following fields:

- **Syslog Server**
  Select a previously configured **Syslog Server** from the drop-down list.
- **Format:**
  You can specify the format of the log message using the values (including properties) entered in this field. For details on the properties that are available, see the section called “Log to Text File”.

Log to System Console

To configure the API Gateway to send logging information to the system console, click the **System Console** tab, and select the **Enable logging to system console** checkbox. For details on how to use the **Format** field to configure the format of the log message, see the section called “Log to Text File”.

Access Log Settings

Overview

The Access Log records a summary of the request and response messages that pass through the API Gateway. By default, the API Gateway records this in the `access.log` file in the `log` directory. This file rolls over at the start of each day so that the name of the log file includes the date on which it was created (for example, `access_30May2012.log`).

The Access Log file format is based on that used by Apache HTTP Server. This means that the log file can be consumed by third-party Web analytics tools such as Webtrends to generate charts and statistics.

Log Format

The syntax used to specify the Access Log file is based on the syntax of available patterns used by the Access Log files in Apache HTTP Server. For example:

```
%h %l %u %t "%r" %s %b
```

The items in this example are explained as follows:

%h	Remote hostname or IP address.
%l	Remote logical username.
%u	Remote user that was authenticated (for example, Distinguished Name of a certificate).
%t	Date and time of the request in Common Log Format.
%r	First line of the request that originated at the client.
%s	HTTP status code returned to the client in the response.
%b	Bytes sent, excluding HTTP headers.

The following extract from the `access.log` file illustrates the resulting log format:

```
s1.oracle.com - lisa [09/05/2012:18:24:48 00] "POST / HTTP/1.0" 200 429
s2.oracle.com - dave [09/05/2012:18:25:26 00] "POST / HTTP/1.0" 200 727
s3.oracle.com - fred [09/05/2012:18:27:12 00] "POST / HTTP/1.0" 200 596
................
................
................
```

For more details on Apache HTTP Server Access Log formats, see the following:

- http://httpd.apache.org/docs/current/logs.html
- http://httpd.apache.org/docs/current/mod/mod_log_config.html#formats

Configuring the Access Log

To configure the Access Log, select the **Settings** node in the Policy Studio tree, and click the **Access Log** tab at the bottom of the screen. Configure the following fields to enable the server to write an Access Log to file:

Enable Apache Access File Logger:
Select whether to configure the Process to start writing event data to the Access Log. This setting is disabled by default.

Pattern:
Enter the Access Log file pattern. This is based on the syntax used in Apache HTTP Server Access Log files, for example:

```
%h %l %u %t "%r" %s %b
```

For more details, see the section called “Log Format”.

Base Log File Name:
Enter the name of the Access Log file in this field. When the file rolls over (because the maximum file size has been reached, or because the date has changed), a suitable increment is appended to the file name. Defaults to `access`.

Directory Name:
Enter the directory for the Access Log file. Defaults to the `logs/access` directory of your product installation.

Log File Extension:
Enter the file extension for the log file. Defaults to `.log`.

Max Files:
Specify the number of log files that are stored. Defaults to `20`.

Max Log File Size:
Specify the maximum size that the log file is allowed reach before it rolls over to a new file. Defaults to `1000` kilobytes.

Roll Over Daily:
Select whether to roll over the log file at the start of each day. This is enabled by default.

Note

These settings configure the Access Log at the API Gateway level. You can configure the Access Log at the service level on a Relative Path. For more details, see the section called “Relative Paths”.

Log Level and Message

Overview

By default, logging is configured for a service with logging level of failure. You can also configure each filter in a policy to log its own message depending on whether it succeeds, fails, and/or throws an exception. Log messages can be stored in several locations, including a database, a file, or the system console. For more details on configuring logging destinations, see the topic on *Audit Log Settings*.

Logging levels apply to the following cases:

- A filter succeeds if it returns a true result after carrying out its processing. For example, if an LDAP directory returns an `authorized` result to an authorization filter, the filter succeeds.
- A filter fails if it returns a false result after performing its processing. For example, an authorization filter returns false if an LDAP directory returns a `not authorized` result to the filter.
- A filter aborts when it can not make the decision it is configured to make. For example, if an LDAP-based authorization filter can not connect to the LDAP directory, it aborts because it can neither authorize nor refuse access. This is regarded as a *fatal* error.

Configuration

You can access the **Log Level and Message** configuration screen by clicking the **Next** button on the main screen of all filters. This screen includes the following fields:

Logging Level:
Configure one of the following options:

Use Service Level Settings	This option is selected by default. Logging is configured for the Web Service with logging level of **Failure**.
Override Logging Level for this Filter	Alternatively, select this option to configure log messages for this filter when it succeeds, fails, and/or aborts. Select **Success**, **Failure**, and/or **Fatal** to configure this filter to log at the respective levels.

Log Messages:
Default log message values are provided at each level for all filters. When you select the checkbox for a particular level, the default log message for that level is used. You can specify an alternative log message by entering the message in the text field provided.

All filters *require* and *generate* message attributes, while some *consume* attributes. In some cases, it may be useful to log the value of these attributes. For example, instead of an authentication filter logging a generic `Authentication Failed` message, you can use the value of the `authentication.subject.id` attribute to log the ID of the user that could not be authenticated.

Use the following format to enter a message attribute selector in a log message:

```
${name_of_attribute}
```

At runtime, the API Gateway expands these selectors to the value of the message attribute. For example, to make sure the ID of a non-authenticated user is logged in the message, enter something like the following in the text field for the **Failure** case:

```
The user '${authentication.subject.id}' could not be authenticated.
```

Then if a user with ID `oracle` can not be authenticated by the API Gateway (a failure case), the following message is logged:

```
The user 'oracle' could not be authenticated.
```

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Audit Logging Behavior:
This setting is relevant only in cases where you have configured the API Gateway to log audit trail messages to a database. For more details, see the instructions in the section called "Log to Database".

You can select the **Abort policy processing on database log error** checkbox if you have configured the API Gateway to write log messages to a database, but that database is not available at runtime. If you have selected this checkbox, and the database is not available, the filter aborts, which in turn causes the policy to abort. In this case, the Fault Handler for the policy is invoked.

Filter Category:
The category selected here identifies the category of filters to which this filter belongs. The default selection should be appropriate in most cases.

Log Message Payload

Overview

The **Log Message Payload** filter is used to log the message payload at any point in the policy. The message payload includes the HTTP headers and MIME/DIME attachments.

By placing the **Log Message Payload** filter at various key locations in the policy, a complete audit trail of the message can be achieved. For example, by placing the filter after each filter in the policy, the complete history of the message can be logged. This is especially useful in cases where the message has been altered by the API Gateway (for example, by signing or encrypting the message, inserting security tokens, or by converting the message to another grammar using XSLT).

Log messages can be stored in several locations, including a database, a file, or the system console. For more details on configuring logging destinations, see the topic on *Audit Log Settings*.

Configuration

Enter an appropriate name for the **Log Message Payload** filter in the **Name** field. It is good practice to use descriptive names for these filters. For example, **Log message before signing message** and **Log message after signing** would be useful names to give to two **Log Message Payload** filters that are placed before and after a **Sign Message** filter.

By default, the **Log Message Payload** filter writes entries to the log file in the following format:

```
${timestamp} ${id} ${filterName} ${payload}
```

However, you can alter the format of the logging output using the values entered in the **Format** field. You can use selectors to output logging information that is specific to the request. You can specify the following properties:

- **`level`:**
  The log level (i.e. fatal, fail, success).
- **`id`:**
  The unique transaction ID assigned to the message.
- **`ip`:**
  The IP address of the client that sent the request.
- **`timestamp`:**
  The time that the message was processed in user-readable form.
- **`filterName`:**
  The name of the filter that generated the log message.
- **`filterType`:**
  The type of the filter that logged the message.
- **`text`:**
  The text of the log message that was configured in the filter itself.
- **`payload`:**
  The complete contents of the HTTP request, including HTTP headers, body, and attachments.

Log Access Filter

Overview

The **Log Access** filter is used by the API Gateway to log records of all messages that pass through the filter. The API Gateway writes the access log to an `access.log` file in the `logs/access` directory. This file rolls over at the start of each day so that the name of the log file includes the date the date on which it was created (for example, `access_30Apr2010.log`).

Important

The **Log Access** filter is deprecated in version 7.x. Instead you should use the **Access Log** available in the **Settings** in the Policy Studio. For more details, see the *Access Log Settings* topic.

Log Format

The format of the log entries is Common Log Format, which has the following format:

```
host ident authuser date request status bytes
```

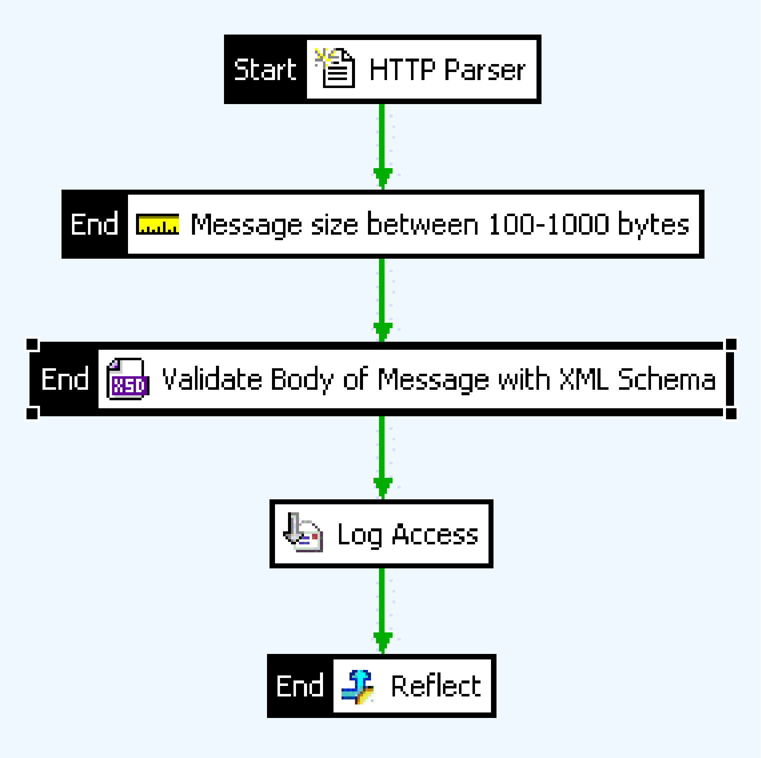
The following list explains each item:

- **host:** The remote hostname.
- **ident:** The remote logname of the user.
- **authuser:** The username by which the user has authenticated himself (for example, the Distinguished Name of a certificate).
- **date:** The date and time of the request.
- **request:** The request line exactly as it originated at the client.
- **status:** The HTTP status code returned to the client.
- **bytes:** The content-length of the document returned to the client.

The following extract from the `access.log` file illustrates the format:

```
m1.oracle.com - Good [30/Mar/2009:22:09:05 00] "http://services/qotd" 200 587
m3.oracle.com - Good [30/Mar/2009:22:10:34 00] "http://services/qotd" 200 671
m1.oracle.com - Good [30/Mar/2009:22:10:53 00] "http://services/qotd" 200 571
................
................
................
```

Because the **Log Access** filter reports the number of bytes returned to the client (the `bytes` parameter explained above), it should be positioned towards the end of a policy. A typical policy involving a **Log Access** filter might appear as follows:


Start
HTTP Parser
End
Message size between 100-1000 bytes
End
Validate Body of Message with XML Schema
Log Access
End
Reflect

Configuration

The **Log Access** filter requires only a **Name** field to be configured.

Service Level Agreement (SLA) Filter

Overview

A Service Level Agreement (SLA) is an agreement put in place between a Web Services host and a client of that Web Service in order to guarantee a certain minimum quality of service. It is common to see SLAs in place to ensure that a minimum number of messages result in a communications failure and that responses are received within an acceptable timeframe. In cases where the conditions of the SLA are breached, it is crucial that an alert can be sent to the appropriate party.

The API Gateway satisfies these requirements by allowing SLAs to be configured at the policy level. It is possible to configure SLAs to monitor the following types of problems:

- Response times
- HTTP status codes returned from the Web Service
- Communication failures

The SLA monitoring performed by the API Gateway is *statistical*. Because of this, a single message (or even a small number of messages) is not considered a sufficient sample to cause an alert to be triggered. The monitoring engine actually uses an *exponential decay* algorithm to determine whether an SLA is failing or not. This algorithm is best explained with an example.

Assume the *poll rate* is set to 3 seconds (i.e. 3000ms), the *data age* is set to 6 seconds (i.e. 6000ms), and you have a Web Service with an average processing time of 100ms. A single client sending a stream of requests through the API Gateway will be able to generate about 10 requests per second, given the Web Services's 100ms response time.

At every 3 seconds poll period you will have data from a previous 30 samples to consider the average response times of. However, rather than simply using the response time of the *last* 3 seconds worth of data, historical data is "smoothed" into the current estimate of the failing percentage. The new data is combined with the existing data such that it will take approximately the data age time for a sample to disappear from the average.

Therefore the closer the data age is to the sampling rate, the less significant historical data becomes, and the more significant the "last" sample becomes.

In order to generate an alert, you must also have enough significant samples at each poll period to consider the date to be statistically valid. For example, if a single request arrives over a period of 1 hour it may not be fair to say that "less than 20%" of all received requests have failed the response time requirements. For this reason, statistical analysis provides a more realistic SLA monitoring mechanism than a solution based purely on absolute metrics.

Response Time Requirements

You can monitor the response times of Web Services protected by the **SLA Filter**. This filter provides different ways of measuring response times:

Response Time Measurement	***Description***
receive-request-start	The time that the API Gateway receives the first byte of the request from the client.
receive-request-end	The time that the API Gateway receives the last byte of the request from the client.
send-request-start	The time that the API Gateway sends the first byte of the request to the Web Service.
send-request-end	The time that the API Gateway sends the last byte of the

Response Time Measurement	***Description***
	request to the Web Service
receive-response-start	The time that the API Gateway receives the first byte of the response from the Web Service
receive-response-end	The time that the API Gateway receives the last byte of the response from the Web Service.
send-response-start	The time that the API Gateway sends the first byte of the response to the client.
send-response-end	The time that the API Gateway sends the last byte of the response to the client.

The API Gateway will measure each of the 8 time values. They will available for processing after the policy has completed for a single request. These 8 options are available for the following reasons:

- The API Gateway may start to send the first byte to the Web Service before the last byte is received from the client, i.e. send-request-start <receive-request-end. This will occur if the invoked policy does not require the full message to be read into memory.
- The API Gateway may start to send the response to the client before the complete response has been received from the Web Service, i.e. send-response-start < receive-response-end. This will occur when invoked policy does not require the full message to be read into memory.
- It is possible that the Web Service may start to send the response before it has received the complete request. However, the API Gateway will not start to read the response until it has sent the complete request. This means that the following is always true:- send-request-end < receive-response-start.
- The time value for send-response-end will depend upon the client application. This value will be larger if the client is slow to read the response.

To add a Response Time Requirement for an SLA, click the **Add** button.

To configure the start time and end time for the response time measurement, click the **Add** button. On the **Settings** tab, specify the percentage of response times that must be below a specified time interval (in milliseconds) in the fields provided. The purpose of these options is to allow for situations where a very small number of unusually slow requests may cause an SLA to trigger unnecessarily. By using percentages, such requests will not distort the statistics collected by the API Gateway.

Click the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, i.e. when the breached conditions are no longer in breach of the SLA.

Finally, click the **Advanced** tab to configure timing information. Select a **Start Timing Point** from the 8 times listed in the table above. The API Gateway will *start* measuring the response time from this time. Then select an **End Timing Point** from the 8 times listed in the table above. The API Gateway will *stop* measuring the response time from this time.

HTTP Status Requirements

HTTP status codes may be received from a Web Service. The API Gateway can be configured to monitor these and generate alerts based on the number of occurrences of certain types of status code response. HTTP status codes are three digit codes that may be grouped into standard status "classes", with the first digit indicating the status class. The status classes are as follows:

HTTP Status Code Class	***Description***
1xx	These status codes indicate a provisional response.
2xx	These status codes indicate that the client's request was successfully received, understood, and accepted.
3xx	These status codes indicate that further action needs to be taken by the user agent in order to fulfill the request.
4xx	These status codes are intended for cases in which the client seems to have erred. For example 401, means that authentication has failed.
5xx	These status codes are intended for cases where the server has encountered an unexpected condition that prevented it from fulfilling the request. For example, 500 is used to transmit SOAP faults.

The API Gateway may monitor a class (i.e. range) of status codes, or they may monitor specific status codes. For example, it is possible to configure the following HTTP status code requirements:

- At least 97% of the requests must yield HTTP status codes between 200 and 299
- At most 2% of requests may yield HTTP status codes between 400 and 499
- At most 0% of requests may yield HTTP status code 500

Click the **Add** button in the **HTTP Status Code Requirements** section.

Select an existing status code or class of status codes from the **HTTP Status Code** dropdown. To add a new code or range of codes, click the **Add** button.

Enter a name for the new code or range of codes in the **Name** field of the **Configure HTTP Status Code** dialog. Enter the *first* HTTP status code in the range of status codes that you want to monitor in the **Start Status** field. Then enter the *last* HTTP status code in the range of status codes that you want to monitor in the **End Status** field.

If you just want to monitor one specific status code, enter the same code in the **Start Status** and **End Status** fields.

Click **OK** when you are satisfied with the selected range of status codes to return to the previous dialog. The remaining 2 fields allow the administrator to specify the minimum or maximum percentage of received HTTP status codes that fall into the configured range before an alert is triggered.

Again, the use of percentages here is to allow for situations where a very small number of requests return the status codes within the "forbidden" range. By using percentages, such requests will not distort the statistics collected by the API Gateway.

Click the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, (when the breached conditions are no longer in breach of the SLA).

Communications Failure Requirements

The API Gateway is deemed to have experienced a *communications failure* when it fails to connect to the Web Service, fails to send the request, or fails to receive the response.

The requirements for communications failures may be expressed as follows:

- No more than 4% of requests may result in communications failures.

Enter the percentage of allowable communications failures in the field provided. An alert will be configured if the percentage of communicates failures rises above this level.

Click the **Message Text** tab to configure the messages that will appear in the alert message when the SLA is breached and also when the SLA is cleared, i.e. when the breached conditions are no longer in breach of the SLA.

Select Alerting System

If an alert is triggered, it must be sent to an alerting destination. The API Gateway can send alerts to the following destinations:

- Windows Event Log
- Email Recipient
- SNMP Network Management System
- Local Syslog
- Remote Syslog
- CheckPoint FireWall-1 (OPSEC)
- Twitter

The **Select Alerting System** table at the bottom of the screen displays all available alerting destinations that have been configured. You can click **Add** to configure an alert destination. For more details, see the topic on *System Alerting*.

Select one or more alerting systems in the table. An alert will be sent to each selected system in the event of a violation of the performance requirements. Alert clearances will be generated when the violation no longer exists.

Set Service Context

Overview

The **Set Service Context** filter configures service-level monitoring details. For example, you can use the fields on this filter screen to configure whether the API Gateway stores service usage and service usage per client details. You can also set the name of the service displayed in the web-based API Gateway Manager monitoring tools and Oracle API Gateway Analytics reporting tools.

When you use the API Service Manager tool to virtualize an API service, a **Set Service Context** filter is automatically generated for the service. For example, you can view generated policies in the Policy Studio tree under the **Polices** -> **Generated Polices** -> **API Service Manager** -> **Services**.

Configuration

Name:
Enter an appropriate name for the filter to be displayed in a policy.

Service Name:
Enter an appropriate name for this service to be displayed in the Web-based API Gateway Manager tools, and in the Oracle API Gateway Analytics interface when generating reports for this service.

Monitoring Options:
The fields in this group enable you to configure whether this service stores usage metrics data to a database. For example, this information can be used by Oracle API Gateway Analytics to produce reports showing how and who is calling this service. The following fields are available:

- **Monitor service usage:**
  Select this option if you want to store message metrics for this service.
- **Monitor service usage per client:**
  Select this option if you want to generate reports monitoring which authenticated clients are calling which services.
- **Monitor client usage:**
  If you want to generate reports on authenticated clients, but are not interested in which services they are calling, select this option and deselect **Monitoring service usage per client**.
- **Which attribute is used to identify the client?:**
  Enter the message attribute to use to identify authenticated clients. The default is `authentication.subject.id`, which stores the identifier of the authenticated user (for example, the username or user's X.509 Distinguished Name).
- **Composite Context:**
  This setting enables you to select a service context as a composite context in which multiple service contexts are monitored during the processing of a message. This setting is not selected by default.

  For example, the API Gateway receives a message, and sends it to `serviceA` first, and then to `serviceB`. Monitoring is performed separately for each service by default. However, you can set a composite service context before `serviceA` and `serviceB` that includes both services. This composite service passes if both services complete successfully, and monitoring is also performed on the composite service context.

API Gateway OAuth 2.0 Introduction

Overview

The Oracle API Gateway can be used as an OAuth Authorization Server and as an OAuth Resource Server. The Authorization Server issues tokens to clients on behalf of a Resource Owner for use in authenticating subsequent API calls to the Resource Server.

You should be familiar with the terms and concepts of *The OAuth 2.0 Authorization Framework* specification: http://tools.ietf.org/html/draft-ietf-oauth-v2-27

API Gateway OAuth Components
The API Gateway ships with the following features to support OAuth 2.0:

- Client application registration
- Generation of access tokens and authorization codes
- Support for the following OAuth flows:
  - Authorization Code
  - Implicit Grant
  - Resource Owner Password Credentials
  - Client Credentials
  - JWT
  - Refresh Token
  - Revoke Token
  - Token Information Service
- Sample clients for all of the above scenarios

OAuth 2.0 Definitions

The API Gateway uses the following definitions of basic OAuth 2.0 terms:

Resource Owner: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user.

Resource Server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Client: An application making protected requests on behalf of the resource owner and with its authorization.

Authorization Server: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

OAuth 2.0 Authentication Flows

The API Gateway supports the following authentication flows:

OAuth 2.0 Authorization Code Grant (Web Server): The Web server authentication flow is used by applications that are hosted on a secure server. A critical aspect of the Web server flow is that the server must be able to protect the issued client application's secret.

OAuth 2.0 Implicit Grant (User-Agent): The user-agent authentication flow is used by client applications residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript or Flash. These clients cannot keep the client secret confidential.

OAuth 2.0 Resource Owner Password Credentials: This username-password authentication flow can be used when the client application already has the resource owner's credentials.

OAuth 2.0 Client Credentials: This username-password flow is used when the client application needs to directly access its own resources on the resource server. Only the client application's credentials are used in this flow. The resource owner's credentials are not required.

OAuth 2.0 JWT: This flow is similar to OAuth 2.0 Client Credentials. A JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.

OAuth 2.0 Refresh Token: After the consumer has been authorized for access, it can use a refresh token to get a new access token. This is only done after the consumer already has received an access token using the Authorization Code Grant or Resource Owner Password Credentials Grant flow.

OAuth 2.0 Revoke Token: A revoke token request causes the removal of the client permissions associated with the particular token to access the end-user's protected resources.

OAuth 2.0 Token Information Service: The OAuth Token Info service responds to requests for information on a specified OAuth 2.0 access token.

Further Information

For more details on the API Gateway OAuth 2.0 support, see the following topics:

- *Configuring and Managing OAuth 2.0*
- *API Gateway OAuth 2.0 Authentication Flows*

For more details on OAuth 2.0, see *The OAuth 2.0 Authorization Framework*:
http://tools.ietf.org/html/draft-ietf-oauth-v2-27

Configuring and Managing OAuth 2.0

Overview

Client applications that send OAuth requests to the API Gateway’s Authorization Server must be registered with the Authorization Server. This topic describes the OAuth 2.0 endpoints used to manage client applications and the pre-registered examples provided with the API Gateway. It describes the registry used to store these client applications, and provides details how to manage them using a REST API-based HTML interface. This topic also includes details on the database schema, and SSL commands used for the example client applications.

Enabling OAuth 2.0 Management

The API Gateway provides the following endpoints used to manage OAuth 2.0 client applications:

Description	URL
Authorization Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/authorize`
Token Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/token`
Token Info Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/tokeninfo`
Revoke Endpoint (REST API)	`https://`*`GATEWAY`*`:8089/api/oauth/revoke`
Oracle API Manager (HTML Interface)	`https://`*`GATEWAY`*`:8089`
Oracle API Manager (REST API)	`https://`*`GATEWAY`*`:8089/api/kps/ClientApplicationRegistry`

where *`GATEWAY`* is the machine on which the API Gateway is installed.

Important

You must first enable the OAuth listener port in the API Gateway before these endpoints are available.

Enabling OAuth Endpoints
To enable the OAuth management endpoints on your API Gateway, perform the following steps:

1. In the Policy Studio tree, select **Listeners** -> **API Gateway** -> **OAuth 2.0 Services** -> **Ports**.
2. Right-click the **OAuth 2.0 Interface** in the panel on the right, and select **Edit**.
3. Select **Enable Interface** in the dialog.
4. Click the **Deploy** button in the toolbar.
5. Enter a description and click **Finish**.

Note

On Linux-based systems, such as Oracle Enterprise Linux, you must open the firewall to allow external access to port `8089`.

If you need to change the port number, set the value of the `env.PORT.OAUTH2.SERVICES` environment variable. For more details, see *Deploying the API Gateway in Multiple Environments*.

Pre-registered Client Applications

The API Gateway ships with a number of pre-registered example client applications.

Note

These client applications are for demonstration purposes only and should be removed before moving the Authorization Server into production.

The default example client applications include the following:

Client ID	Client Secret
`SampleConfidentialApp`	`6808d4b6-ef09-4b0d-8f28-3b05da9c48ec`
`SamplePublicApp`	`3b001542-e348-443b-9ca2-2f38bd3f3e84`

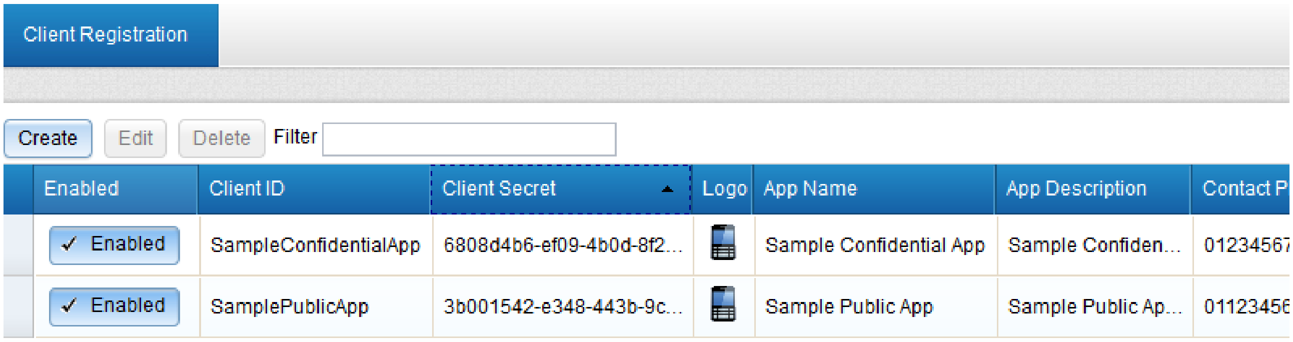
Managing Registered Clients

Every client application that sends OAuth requests to the API Gateway's OAuth Authorization Server must be registered with the Oracle API Manager. The API Gateway provides the API Manager Web-based HTML interface for managing registered client applications. It also provides the API Manager REST API that enables you to manage registered clients on the command line.

Accessing the API Manager Web Interface
You can access the API Manager Web interface at the following URL:

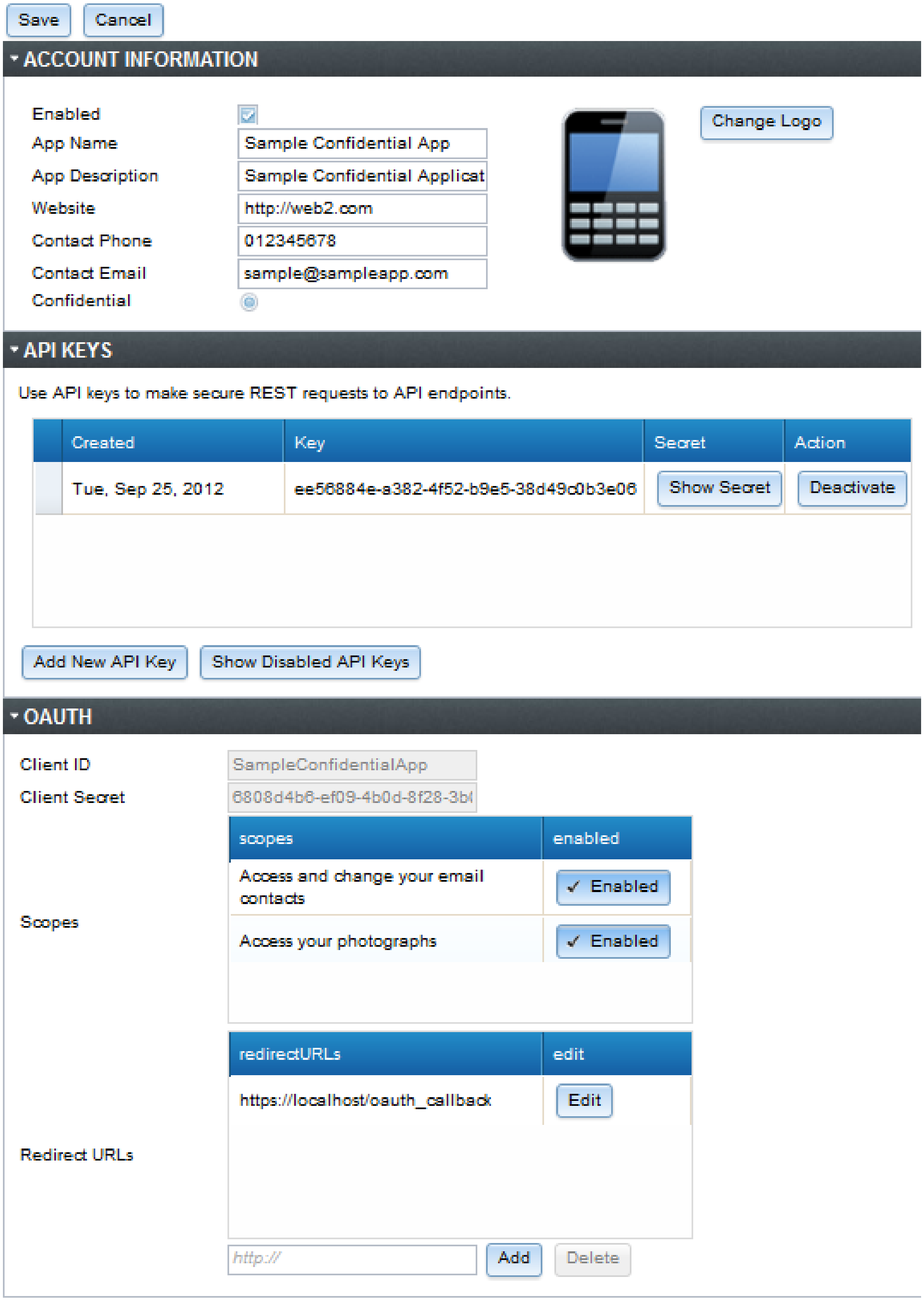
```
https://localhost:8089
```

This interface is displayed as follows:



Enabled	Client ID	Client Secret ▲	Logo	App Name	App Description	Contact P
✓ Enabled	SampleConfidentialApp	6808d4b6-ef09-4b0d-8f2...		Sample Confidential App	Sample Confiden...	01234567
✓ Enabled	SamplePublicApp	3b001542-e348-443b-9c...		Sample Public App	Sample Public Ap...	01123456

You can select a registration entry and click **Edit** to update its details. For example, you can can select the scopes that an application requires access to, and configure API keys or certificates. Remember to click **Save** when finished.



ACCOUNT INFORMATION

Enabled	☑
App Name	Sample Confidential App
App Description	Sample Confidential Applicat
Website	http://web2.com
Contact Phone	012345678
Contact Email	sample@sampleapp.com
Confidential	●

API KEYS

Use API keys to make secure REST requests to API endpoints.

Created	Key	Secret	Action
Tue, Sep 25, 2012	ee56884e-a382-4f52-b9e5-38d49c0b3e06	Show Secret	Deactivate

OAUTH

Client ID	SampleConfidentialApp
Client Secret	6808d4b6-ef09-4b0d-8f28-3b(

Scopes

scopes	enabled
Access and change your email contacts	✓ Enabled
Access your photographs	✓ Enabled

Redirect URLs

redirectURLs	edit
https://localhost/oauth_callback	Edit

http://

By default, the API Manager back-end is file-based and uses the API Gateway's Key Property Store (KPS). For more details, see *Key Property Stores*. You can also store this data in a database. For more details, see the section called "Token Management".

Accessing the API Manager REST API
For details on using the API Manager REST interface on the command line, see the section called "API Manager REST API".

Sample Clients

The API Gateway includes sample Jython client applications for all supported OAuth flows in the following directory your API Gateway installation:

```
INSTALL_DIR/samples/scripts/oauth
```

To run a sample script, open a UNIX shell or DOS command prompt in the following directory:

```
INSTALL_DIR/samples/scripts
```

Windows
For example, run the following command:

```
> run.bat oauth\implicit_grant.py
```

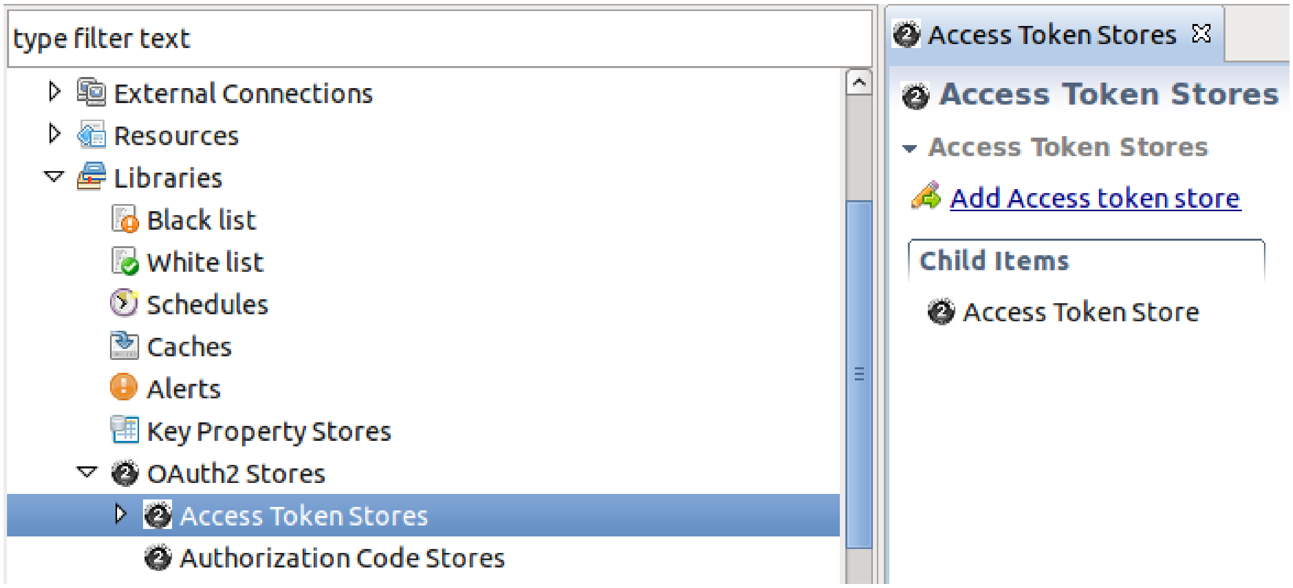
Linux/Solaris
For example, run the following command:

```
> sh run.sh oauth/implicit_grant.py
```

Token Management

The API Gateway can store generated authorization codes and access tokens in its caches or in a central database. The Authorization Server issues tokens to clients on behalf of a Resource Owner for use in authenticating subsequent API calls to the Resource Server. These issued tokens need to be persisted so that subsequent client requests to the Authorization Server can be validated.

The following screen shows the OAuth stores in the Policy Studio:



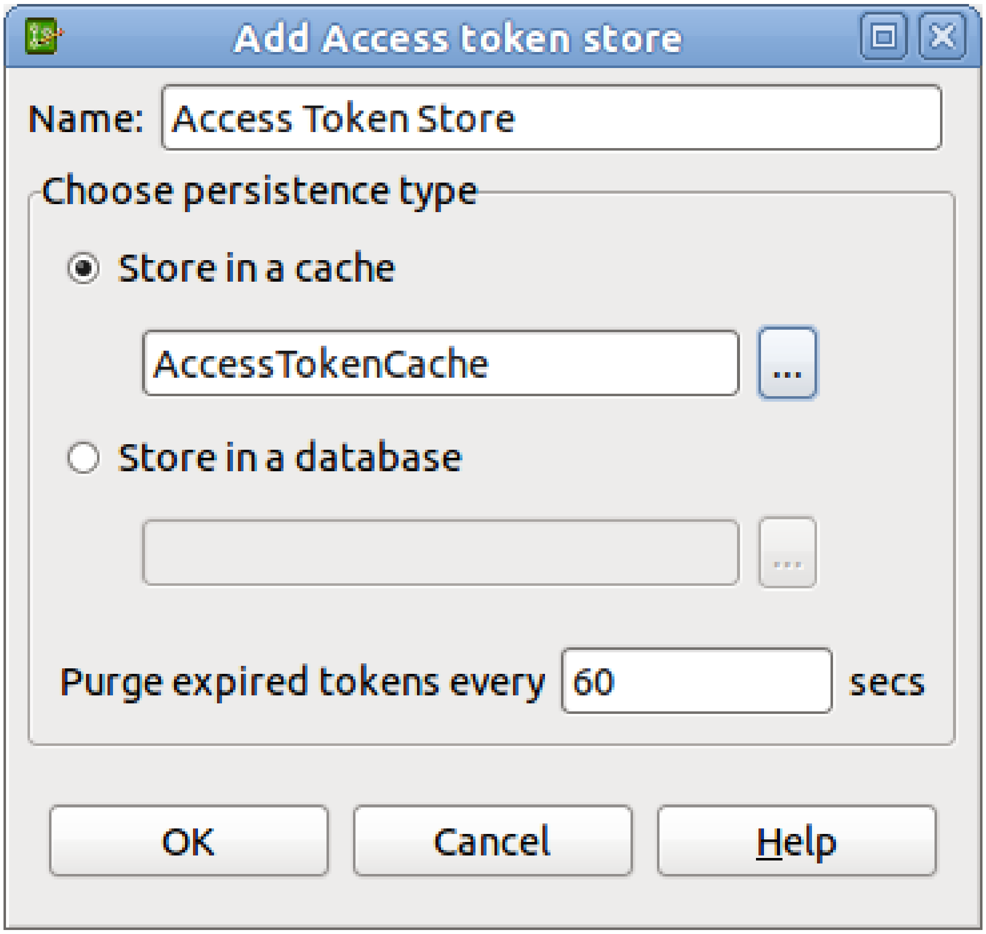
Note

The Authorization Server can cache authorization codes and access tokens depending on the OAuth flow. The steps for adding an authorization code cache are similar to adding an access token cache.

The Authorization Server offers the following token persistent storage options:

- Database
- Cache

The following screen shows these options in the Policy Studio:



Add Access token store

Name: Access Token Store

Choose persistence type

● Store in a cache

AccessTokenCache

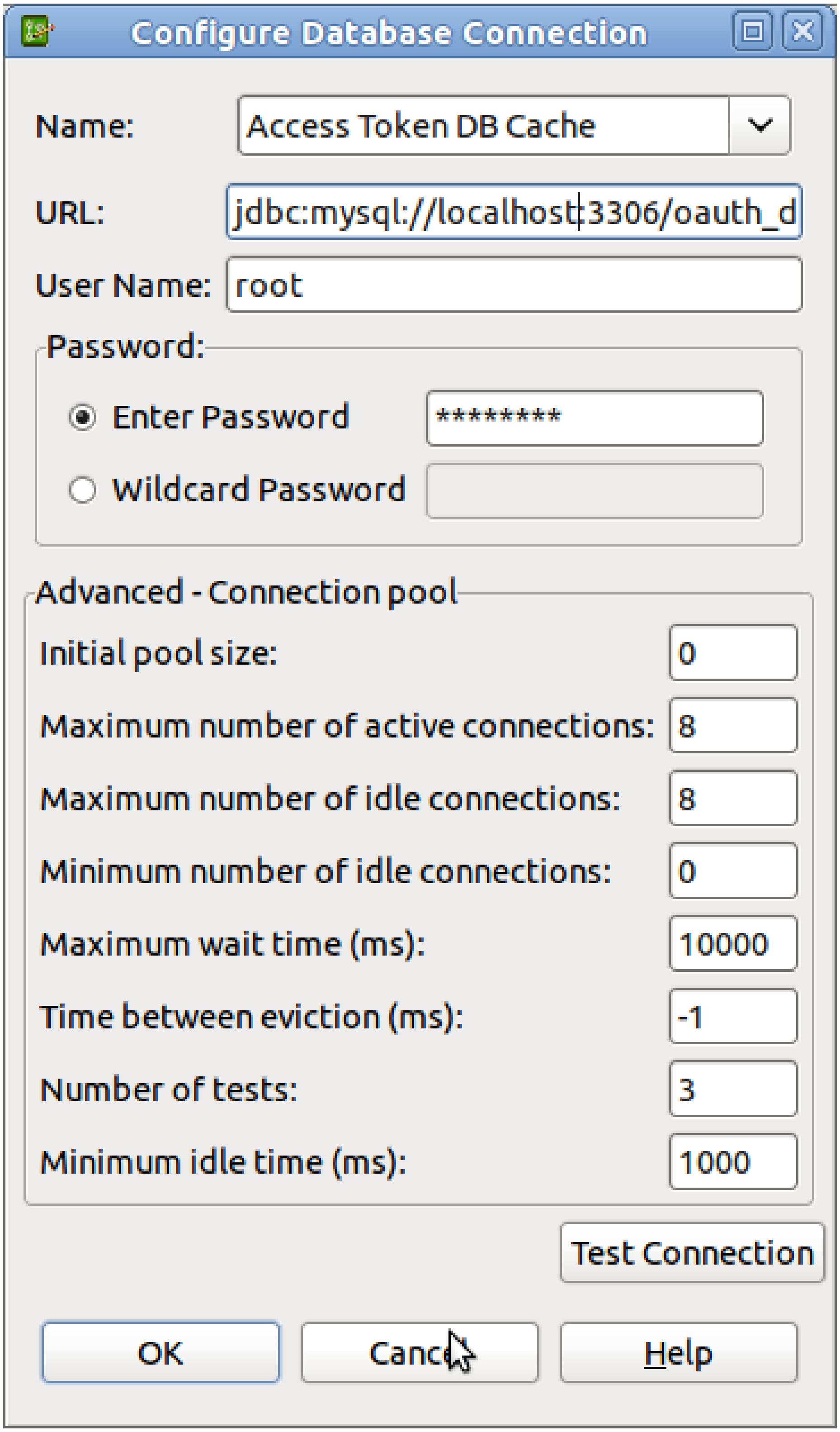
○ Store in a database

Purge expired tokens every 60 secs

The **Purge expired tokens every** 60 secs setting enables you to configure the time in seconds that a background process will polls the database or cache looking for expired access/refresh tokens or authorization codes.

Caching Tokens using a Database
Perform the following steps:

1. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.
2. This displays the dialog that enables you to choose the persistence type. Select **Store in a database**, and select the browse button to display a database configuration dialog.
3. Complete the database configuration details. The following example uses a MySQL database, and a database instance named `oauth_db`. For more details, see *Database Connection*.



Configure Database Connection

Name: Access Token DB Cache

URL: jdbc:mysql://localhost:3306/oauth_d

User Name: root

Password:

● Enter Password ********

○ Wildcard Password

Advanced - Connection pool

Initial pool size:	0
Maximum number of active connections:	8
Maximum number of idle connections:	8
Minimum number of idle connections:	0
Maximum wait time (ms):	10000
Time between eviction (ms):	-1
Number of tests:	3
Minimum idle time (ms):	1000

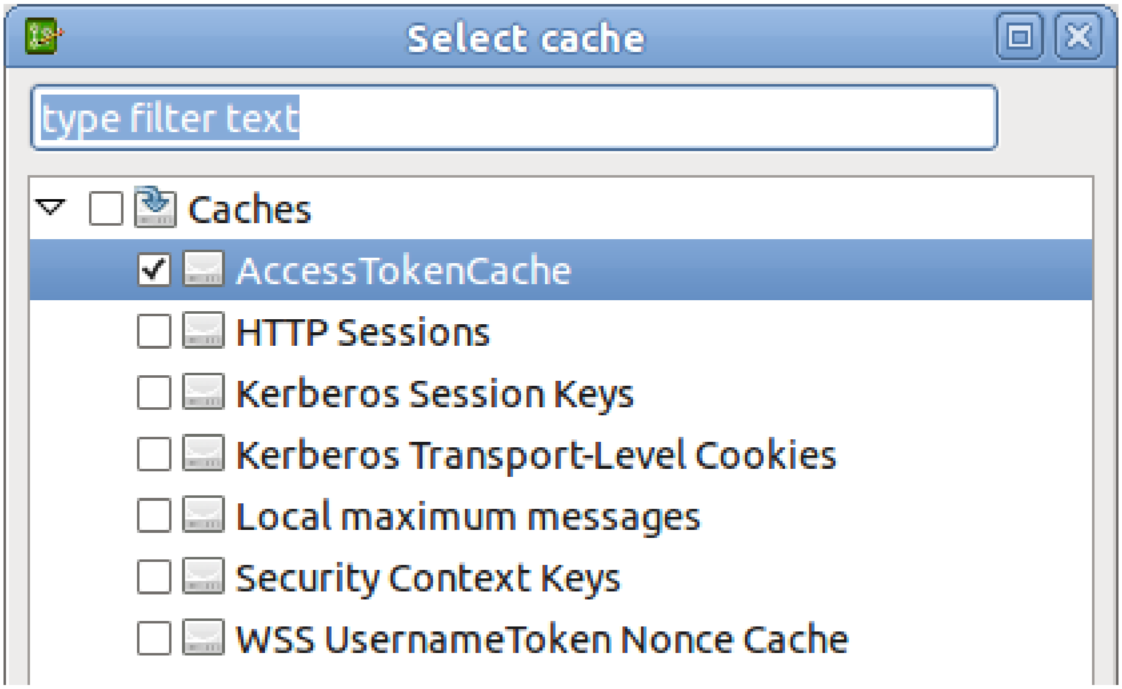
Note

On first use of the database for caching access tokens, the following tables are created automatically: oauth_access_token and oauth_refresh_token. A table named oauth_authz_code is created for caching authorization codes.

For more details, see the section called “Database-Backed API Manager”.

Caching Tokens using EHCache

1. Right-click **Access Token Stores** in the Policy Studio tree, and select **Add Access Token Store**.
2. This displays a dialog that enables you to choose the persistence type. Select **Store in a cache**, and select the browse button to display the cache configuration dialog.
3. Add a new cache (for example, AccessTokenCache). For more details, see *Global Caches*.



Select cache

type filter text

- □ Caches
  - ☑ AccessTokenCache
  - □ HTTP Sessions
  - □ Kerberos Session Keys
  - □ Kerberos Transport-Level Cookies
  - □ Local maximum messages
  - □ Security Context Keys
  - □ WSS UsernameToken Nonce Cache

API Manager REST API

This section desribes a set of commands that enables you to use the API Manager REST API to manage client applications.

Register a Client Application
You can register client applications using a command line utility such as curl. For example, to register a new client application, use the following command:

```
curl -v --header "Content-Type:application/json" -X POST -d @newobj.json
  --user admin:changeme
```

where newobj.json is something like the following:

```
{
      "storeId": "ClientApplicationRegistry",
      "typeId": "AppDetails",
      "objects": [
          {
             "enabled": true,
             "logo": "http://localhost:8090/logos/logo.gif",
             "scopes": [
                 "https://localhost:8090/auth/userinfo.email",
                 "https://localhost:8090/auth/userinfo.profile"
               ],
             "contactPhone": "012345678",
             "contactEmail": "sample@sampleapp.com",
             "description": "Oracle Sample Confidential Application",
             "name": "Oracle Sample Confidential App",
             "clientID": "SampleConfidentialApp",
             "clientSecret": "NjgwOGQ0YjYtZWYwOS00YjBkLThmMjgtM2IwNWRhOWM0OGVj",
             "redirectURLs": [
                 "http://localhost:8080/auth/redirect.html"
               ],
             "base64EncodedCert":
                     "-----BEGIN CERTIFICATE-----
                 MIIDpTCCAw6gAwIBAgIJAKqGffYjCdSiMA0GCSqGSIb3DQEBBQUAMIGUMQswCQYD
                         .....
                         -----END CERTIFICATE-----\n",
             "clientType": "confidential"
           }
       ]
   }
```

Retrieve all Registered Clients
Use the following command:

```
curl --user admin:changeme   | python -mjson.tool
```

Viewing a Registered Client
This is similar to retrieving all registered clients except you are passing in a registered `client_id` (in this example, `95574d66-f115-4c9b-b93e-526d383b4b6a`). Use the following command:

```
curl --user admin:changeme
  https://localhost:8089/api/kps/ClientApplicationRegistry/
  95574d66-f115-4c9b-b93e-526d383b4b6a | python -mjson.tool
```

To access one of the preregistered clients, you can call the following:

```
curl --user admin:changeme
  https://localhost:8089/api/kps/ClientApplicationRegistry/
  SampleConfidentialApp | python -mjson.tool
```

Removing a Registered Client
To delete a client, pass in the client ID at the end of the request. For example:

```
curl -v -X DELETE --user admin:changeme
  https://localhost:8089/api/kps/ClientApplicationRegistry/
  95574d66-f115-4c9b-b93e-526d383b4b6a
```

Verify Client Deletion is Successful

This is the same request as viewing a register client application, but this time the API Manager KPS returns a JSON error. For exmaple:

```
curl --user admin:changeme
  https://localhost:8089/api/kps/ClientApplicationRegistry/
  95574d66-f115-4c9b-b93e-526d383b4b6a | python -mjson.tool
```

The following JSON error is returned:

```
{
    "errors": [
        {
            "code": 102,
            "message": "Unexpected exception:
              com.vordel.kps.ObjectNotFound : id:
              95574d66-f115-4c9b-b93e-526d383b4b6a"
         }
    ]
}
```

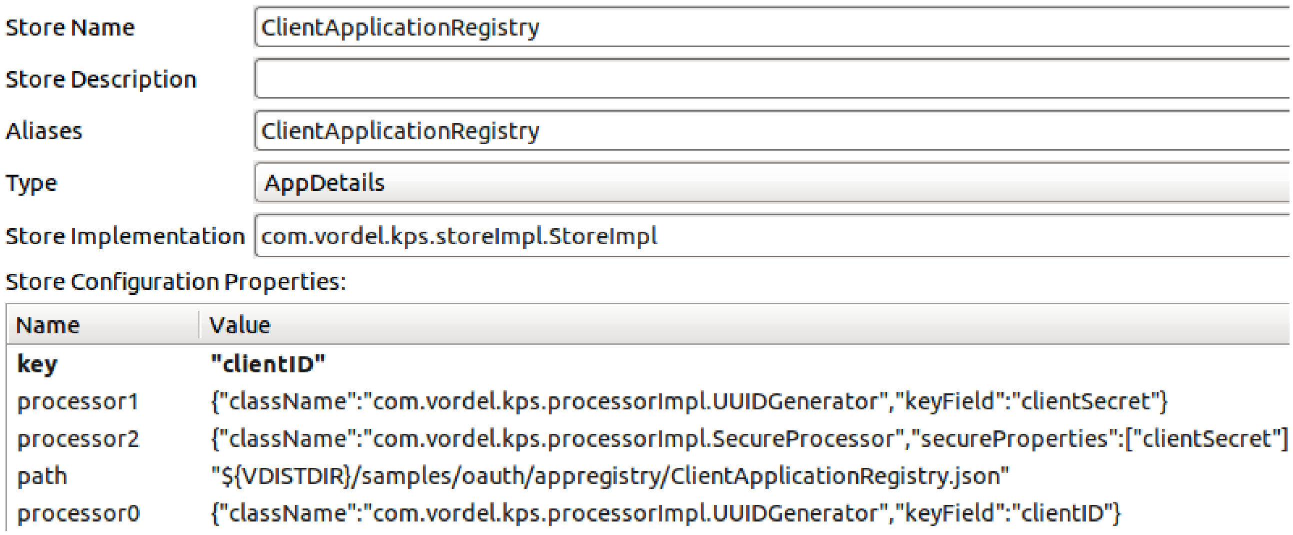
Disabling a Client Application

```
curl -v --header "Content-Type:application/json" -X PUT -d '{"enabled" : false}'
  --user admin:changeme https://localhost:8089/api/kps/ClientApplicationRegistry/
    c5659b84-8524-4592-aed8-c7fa77c8eb26
```

Alternatively, you can upload a full update by passing in a file using `-d @updatedObject.json`.

Database-Backed API Manager

The Oracle API Manager KPS can also be backed by a database. For more details, see *Key Property Stores*. The default file-based KPS that ships in the `samples` directory of the API Gateway is displayed as follows in the Policy Studio:



Store Name	ClientApplicationRegistry
Store Description	
Aliases	ClientApplicationRegistry
Type	AppDetails
Store Implementation	com.vordel.kps.storeImpl.StoreImpl

Store Configuration Properties:

Name	Value
key	**"clientID"**
processor1	{"className":"com.vordel.kps.processorImpl.UUIDGenerator","keyField":"clientSecret"}
processor2	{"className":"com.vordel.kps.processorImpl.SecureProcessor","secureProperties":["clientSecret"]
path	"${VDISTDIR}/samples/oauth/appregistry/ClientApplicationRegistry.json"
processor0	{"className":"com.vordel.kps.processorImpl.UUIDGenerator","keyField":"clientID"}

To change this definition to use a database for storage, add the following property *`name:value`* pairs:

- Database username: `username: root`
- Database password: `password: password`
- Database URL: `url: jdbc:mysql://localhost:3306/oauth_db`

- Database driver: `driver: com.mysql.jdbc.Driver`

Note

Alternatively, you can set the `dbConnectionName` property. This enables you to use a database connection that you have already defined in **External Connections** -> **Database Connections** in the Policy Studio. A setting of `dbConnectionName` means that the `username`, `password`, `url`, and `driver` properties are not required. The `schemaOption` parameter, if set to create, creates the database tables required by the KPS. This variable is typically used at development and test time.

For details on adding KPS properties in the Policy Studio, see *Key Property Stores*.

OAuth Database Schemas

This section shows the OAuth database schemas displayed by example `mysql` commands.

oauth_access_token schema
The following shows the result from the `show columns from auth_access_token;` command:

```
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| id            | varchar(2    | NO   | PRI | NULL    |       |
| auth_request  | blob         | NO   |     | NULL    |       |
| client_id     | varchar(255  | NO   |     | NULL    |       |
| expiry_ti     | datetime     | NO   |     | NULL    |       |
| token         | blob         | NO   |     | NULL    |       |
| refresh_token | varchar(2    | YES  |     | NULL    |       |
| user_auth     | varchar(255  | NO   |     | NULL    |       |
| user_name     | varchar(255  | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

oauth_refresh_token schema
The following shows the result from the `show columns from oauth_refresh_token;` command:

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| token_id     | varchar(255) | NO   | PRI | NULL    |       |
| auth_request | blob         | NO   |     | NULL    |       |
| expiry_time  | datetime     | NO   |     | NUL     |       |
| token        | blob         | NO   |     | NULL    |       |
| user_name    | varchar(255) | NO   |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

oauth_refresh_token schema
The following shows the result from the `show columns from oauth_refresh_token;` command:

```
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| id            | varchar(255) | NO   | PRI | NULL    |       |
| authorization | blob         | NO   |     | NULL    |       |
| expiry_time   | datetime     | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

OpenSSL Commands

The following example `openssl` command shows generating a client application certificate and private key:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mykey.pem
-out mycert.pem
Generating a 1024 bit RSA private key
...................................................................
...........++++++
.....++++++
writing new private key to 'mykey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value.
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:MA
Locality Name (eg, city) []:Newton
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Oracle
Organizational Unit Name (eg, section) []:API Gateway
Common Name (eg, YOUR name) []:SampleConfidentialApp
Email Address []:support@widgits.com
```

OAuth 2.0 Message Attributes

Most of the OAuth 2.0 policy filters in the API Gateway generate message attributes that can be queried further using API Gateway selector syntax. The message attributes generated by the OAuth filters are as follows:

- `accesstoken`
- `accesstoken.authn`
- `authzcode`
- `authentication.subject.id`
- `oauth.client.details`

accesstoken methods

The following methods are available to call on the `accesstoken` message attribute:

```
${accesstoken.getValue()}
${accesstoken.getExpiration()}
${accesstoken.getExpiresIn()}
${accesstoken.isExpired()}
${accesstoken.getTokenType()}
${accesstoken.getRefreshToken()}
${accesstoken.getOAuth2RefreshToken().getValue()}
${accesstoken.getOAuth2RefreshToken().getExpiration()}
${accesstoken.getOAuth2RefreshToken().getExpiresIn()}
${accesstoken.getOAuth2RefreshToken().hasExpired()}
${accesstoken.hasRefresh()}
${accesstoken.getScope()}
${accesstoken.getAdditionalInformation()}
```

The following example shows output from querying each of the `accesstoken` methods:

```
so0HlJYASrnXqn2fL2VWgiunaLfSBhWv6W7JMbmOa131HoQzZB1rNJ
```

```
Fri Oct 05 17:16:54 IST 2012
3599
false
Bearer
xif9oNHi83N4ETQLQxmSGoqfu9dKcRcFmBkxTkbc6yHDfK
xif9oNHi83N4ETQLQxmSGoqfu9dKcRcFmBkxTkbc6yHDfK
Sat Oct 06 04:16:54 IST 2012
43199
false
true
https://localhost:8090/auth/userinfo.email
{department=engineering}
```

accesstoken.authn methods
The following methods are available to call on the `accesstoken.authn` message attribute:

```
${accesstoken.authn.getUserAuthentication()}
${accesstoken.authn.getAuthorizationRequest().getScope()}
${accesstoken.authn.getAuthorizationRequest().getClientId()}
${accesstoken.authn.getAuthorizationRequest().getState()}
${accesstoken.authn.getAuthorizationRequest().getRedirectUri()}
${accesstoken.authn.getAuthorizationRequest().getParameters()}
```

The following example shows output from querying each of the `accesstoken.authn` methods:

```
admin
[https://localhost:8090/auth/userinfo.email]
SampleConfidentialApp
343dqak32ksla
https://localhost/oauth_callback
{client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
  scope=https://localhost:8090/auth/userinfo.email, grant_type=authorization_code,
  redirect_uri=https://localhost/oauth_callback, state=null,
  code=FOT4nudbglQouujRl8oH3EOMzaOlQP, client_id=SampleConfidentialApp}
```

authzcode methods
The following methods are available to call on the `authzcode` message attribute:

```
${authzcode.getCode()}
${authzcode.getState()}
${authzcode.getApplicationName()}
${authzcode.getExpiration()}
${authzcode.getExpiresIn()}
${authzcode.getRedirectURI()}
${authzcode.getScopes()}
${authzcode.getUserIdentity()}
```

The following example shows output from querying each of the `authzcode` methods:

```
F8aHby7zctNRknmWlp3voe61H20Md1
sds12dsd3343ddsd
SampleConfidentialApp
Fri Oct 05 15:47:39 IST 2012
599 (expiry in secs)
https://localhost/oauth_callback
[https://localhost:8090/auth/userinfo.email]
admin
```

oauth.client.details methods

The following methods are available to call on the `oauth.client.details` message attribute:

```
${authzcode.getCode()}
${authzcode.getState()}
${authzcode.getApplicationName()}
${authzcode.getExpiration()}
${authzcode.getExpiresIn()}
${authzcode.getRedirectURI()}
${authzcode.getScopes()}
${authzcode.getUserIdentity()}
```

The following example shows output from querying each of the `oauth.client.details` methods:

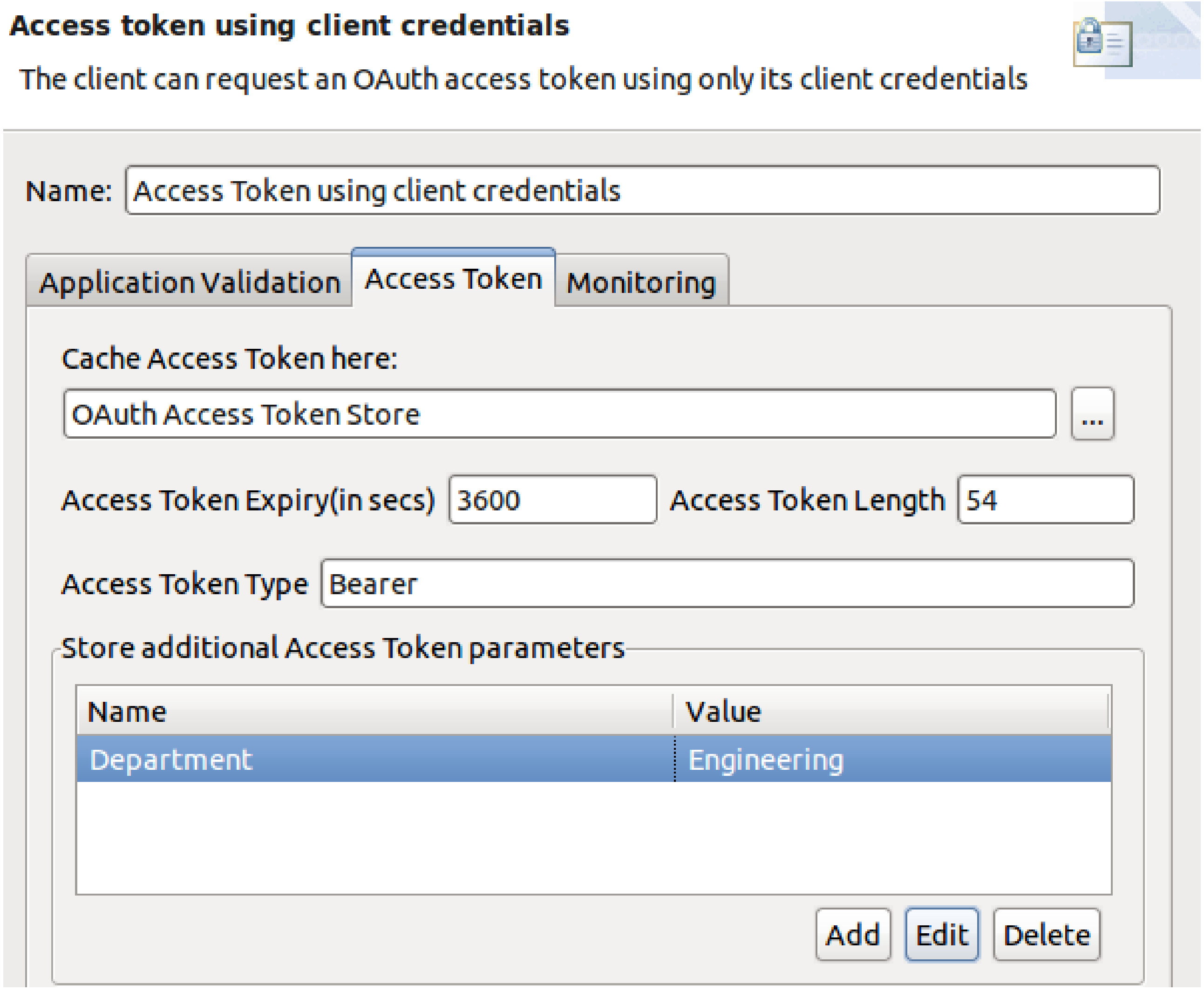
```
F8aHby7zctNRknmWlp3voe61H20Md1
sds12dsd3343ddsd
SampleConfidentialApp
Fri Oct 05 15:47:39 IST 2012
599 (expiry in secs)
https://localhost/oauth_callback
[https://localhost:8090/auth/userinfo.email]
admin
```

Example Use

If you add additional access token parameters to the OAuth 2.0 **Access Token Info** filter, you can return a lot of additional information about the token. For example:

```
{
   "audience" : "SampleConfidentialApp",
   "user_id" : "admin",
   "scope" : "https://localhost:8090/auth/userinfo.email",
   "expires_in" : 3567,
   "Access Token Expiry Date" : "Wed Aug 15 11:19:19 IST 2012",
   "Authentication parameters" : "{username=admin,
    client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
    scope=https://localhost:8090/auth/userinfo.email, grant_type=password,
    redirect_uri=null, state=null, client_id=SampleConfidentialApp,
    password=changeme}",
   "Access Token Type:" : "Bearer"
```

You also have the added flexibility to add extra name/value pair settings to access tokens upon generation.The OAuth 2.0 access token generation filters provide an option to store additional parameters for an access token. For example, if you add the name/value pair `Department/Engineering` to the **Client Credentials** filter:



Access token using client credentials

The client can request an OAuth access token using only its client credentials

Name: Access Token using client credentials

Cache Access Token here: OAuth Access Token Store

Access Token Expiry(in secs) 3600 Access Token Length 54

Access Token Type Bearer

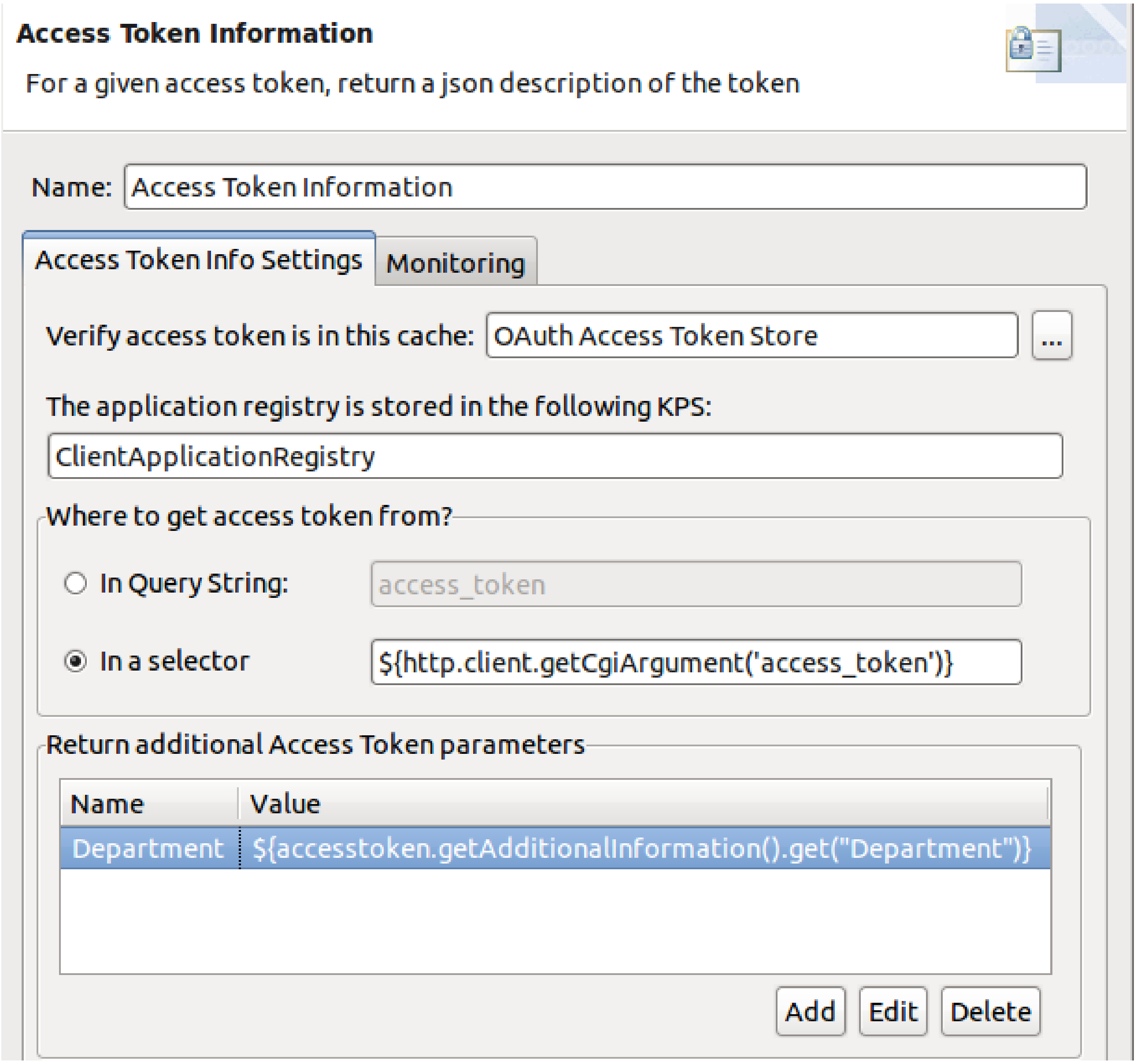
Store additional Access Token parameters

Name	Value
Department	Engineering

You can then update the **Access Token Info** filter to add a name/value pair using a selector to get the following value:

```
Department/${accesstoken.getAdditionalInformation().get("Department")}
```

For example:



Access Token Information

For a given access token, return a json description of the token

Name: Access Token Information

Verify access token is in this cache: OAuth Access Token Store

The application registry is stored in the following KPS:

ClientApplicationRegistry

Where to get access token from?

○ In Query String: access_token

● In a selector ${http.client.getCgiArgument('access_token')}

Return additional Access Token parameters

Name	Value
Department	${accesstoken.getAdditionalInformation().get("Department")}

Then the JSON response is as follows:

```
{
  "audience" : "SampleConfidentialApp",
  "user_id" : "SampleConfidentialApp",
  "scope" : "https://localhost:8090/auth/userinfo.email",
  "expires_in" : 3583,
  "Access Token Type:" : "Bearer",
  "Authentication parameters" :
  "{client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec,
    scope=https://localhost:8090/auth/userinfo.email, grant_type=client_credentials,
    redirect_uri=null, state=null, client_id=SampleConfidentialApp}",
  "Department" : "Engineering",
  "Access Token Expiry Date" : "Wed Aug 15 12:10:57 IST 2012"
```

You can also use API Gateway selector syntax when storing additional information with the token. For more details on selectors, see *Selecting Configuration Values at Runtime*.

API Gateway OAuth 2.0 Authentication Flows

Overview

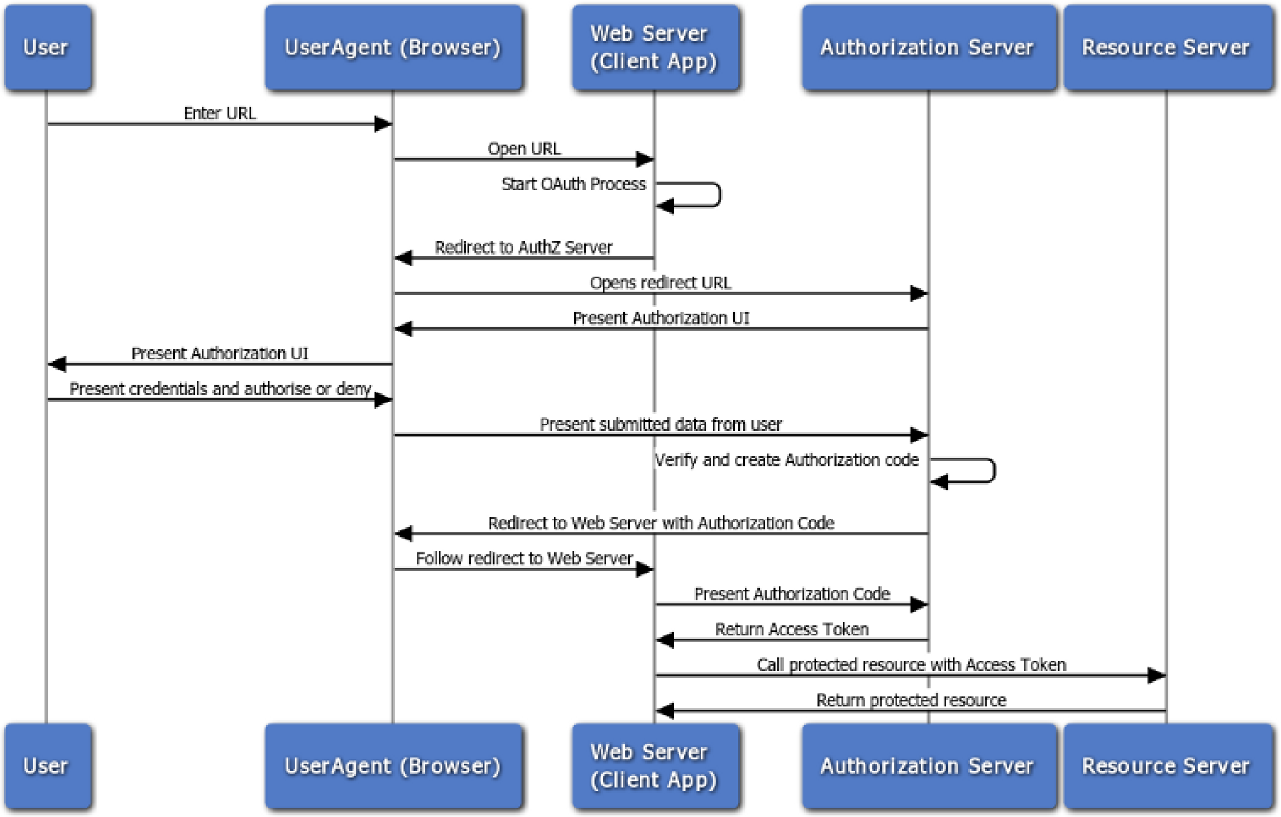
The API Gateway can use the OAuth 2.0 protocol for authentication and authorization. The API Gateway can act as an OAuth 2.0 Authorization Server and supports several OAuth 2.0 flows that cover common Web server, JavaScript, device, installed application, and server-to-server scenarios. This topic describes each of the supported OAuth 2.0 flows in detail, and shows how to run example client applications.

Authorization Code (or Web Server) Flow

The Authorization Code or Web server flow is suitable for clients that can interact with the end-user’s user-agent (typically a Web browser), and that can receive incoming requests from the authorization server (can act as an HTTP server).

The Authorization Code flow is as follows:

1. The Web server redirects the user to the API Gateway acting as an Authorization Server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the Web server receives a callback with an authorization code.
3. After obtaining the authorization code, the Web server passes back the authorization code to obtain an access token response.
4. After validating the authorization code, the API Gateway passes back a token response to the Web server.
5. After the token is granted, the Web server accesses their data.



Obtaining an Access Token
The detailed steps for obtaining an access token are as follows:

1. Redirect the user to the authorization endpoint with the following parameters:

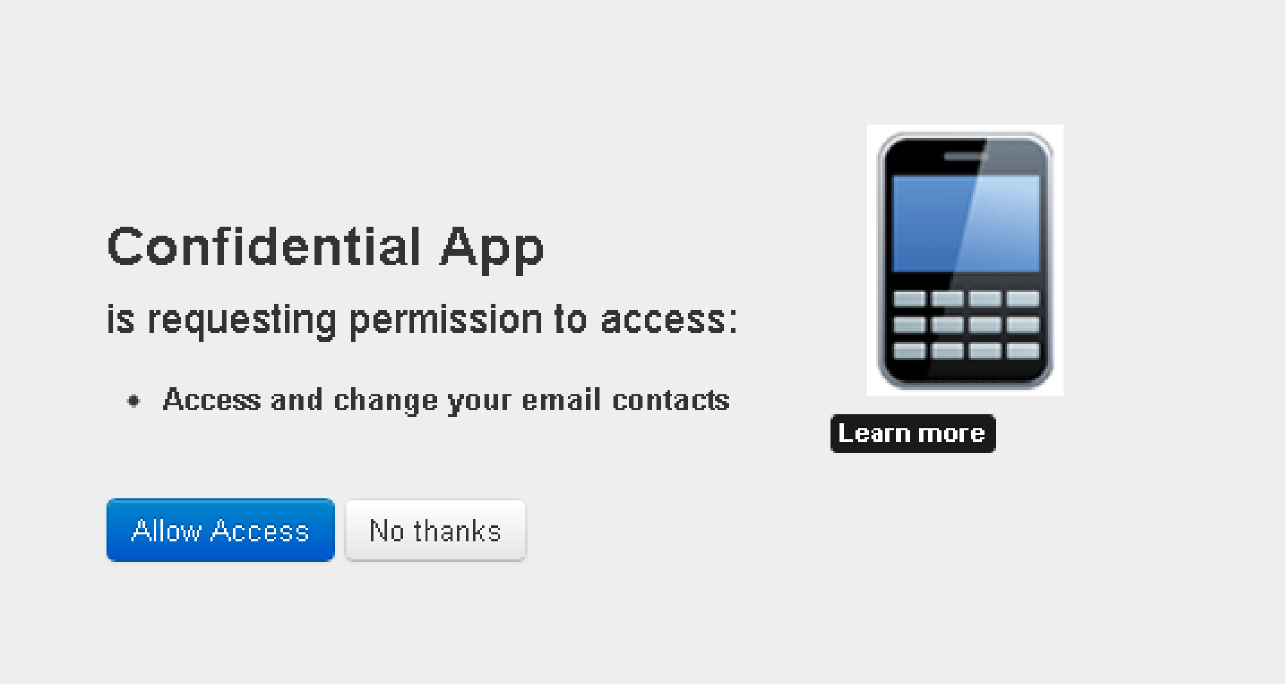
Parameter	Description
`response_type`	Required. Must be set to `code`.
`client_id`	Required. The Client ID generated when the application was registered in the Oracle API Manager.
`redirect_uri`	Optional. Where the authorization code will be sent. This value must match one of the values provided in the Oracle API Manager.
`scope`	Optional. A space delimited list of scopes, which indicate the access to the Resource Owner's data being requested by the application.
`state`	Optional. Any state the consumer wants reflected back to it after approval during the callback.

The following is an example URL:

```
https://apigateway/oauth/authorize?client_id=SampleConfidentialApp&
response_type=code&&redirect_uri=http%3A%2F%2Flocalhost%3A8090%2Fauth%2Fredirect.html&
scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email
```

Note

During this step the Resource Owner user must approve access for the application Web server to access their protected resources, as shown in the following example screen.



Confidential App

is requesting permission to access:

- Access and change your email contacts

2. The response to the above request is sent to the `redirect_uri`. If the user approves the access request, the response contains an authorization code and the `state` parameter (if included in the request). If the user does not approve the request, the response contains an error message. All responses are returned to the Web server on the query string. For example:

```
https://localhost/oauth_callback&code=9srN6sqmjrvG5bWvNB42PCGju0TFVV
```

3. After the Web server receives the authorization code, it may exchange the authorization code for an access token and a refresh token. This request is an HTTPS `POST`, and includes the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `authorization_code`.
`code`	Required. The authorization code received in the redirect above.

redirect_uri	Required. The redirect URL registered for the application during application registration.
client_id*	Optional. The client_id obtained during application registration.
client_secret*	Optional. The client_secret obtained during application registration.
format	Optional. Expected return format. The default is json. Possible values are: • urlencoded • json • xml

* If the client_id and client_secret are not provided as parameters in the HTTP POST, they must be provided in the HTTP Basic Authentication header (Authorization base64Encoded(client_id:client_secret)).

The following example HTTPS POST shows some parameters:

```
POST /api/oauth/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded

client_id=SampleConfidentialApp&client_secret=6808d4b6-ef09-4b0d-8f28-3b05da9c48ec
 &code=9srN6sqmjrvG5bWvNB42PCGju0TFVV&redirect_uri=http%3A%2F%2Flocalhost%3A809
 0%2Fauth%2Fredirect.html&grant_type=authorization_code&format=query
```

4. After the request is verified, the API Gateway sends a response to the client. The following parameters are in the response body:

Parameter	**Description**
access_token	The token that can be sent to the Resource Server to access the protected resources of the Resource Owner (user).
refresh_token	A token that may be used to obtain a new access token.
expires	The remaining lifetime on the access token.
type	Indicates the type of token returned. At this time, this field always has a value of Bearer.

The following is an example response:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache{
    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
    "token_type": "Bearer",
    "expires_in": "3600",
}
```

5. After the Web server has obtained an access token, it can gain access to protected resources on the Resource Server by placing it in an `Authorization: Bearer` HTTP header:

```
GET /oauth/protected HTTP/1.1
Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9
Host: apigateway.com
```

For example, the `curl` command to call a protected resource with an access token is as follows:

```
curl -H "Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9"
  https://apigateway.com/oauth/protected
```

Sample Client
The following Jython sample client creates and sends an authorization request for the authorization grant flow to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/authorization_code.py
```

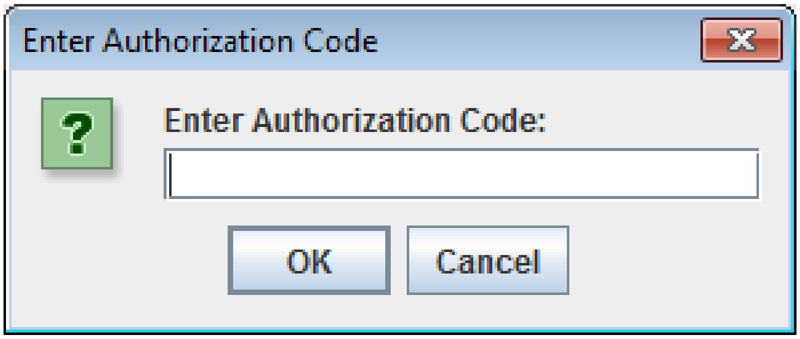
To run the sample, perform the folllowing steps:

1. Open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

   ```
   > run oauth/authorization_code.py
   ```

   The script outputs the following:

   ```
   > Go to the URL here:
   http://127.0.0.1:8080/api/oauth/authorize?client_id=SampleConfidentialApp&response_type=
     code&scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email&redirect_uri=
     https%3A%2F%2Flocalhost%2Foauth_callback
   Enter Authorization code in dialog
   ```

2. Copy the URL output to the command prompt into a browser, and perform the following steps as prompted:

   a. Provide login credentials to the authorization server. The default values are:
      - Username: `admin`
      - Password: `changeme`
   b. When prompted, grant access to the client application to access the protected resource.
3. After the Resource Owner has authorized and approved access to the application, the Authorization Server redirects a fragment containing the access code to the redirection URI. For example:

```
https://localhost/oauth_callback&code=AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
```

In this example, the access token is:

```
AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
```

Enter this value into the **Enter Authorization Code** dialog, and the script attempts to access the protected resource using the access token. For example:

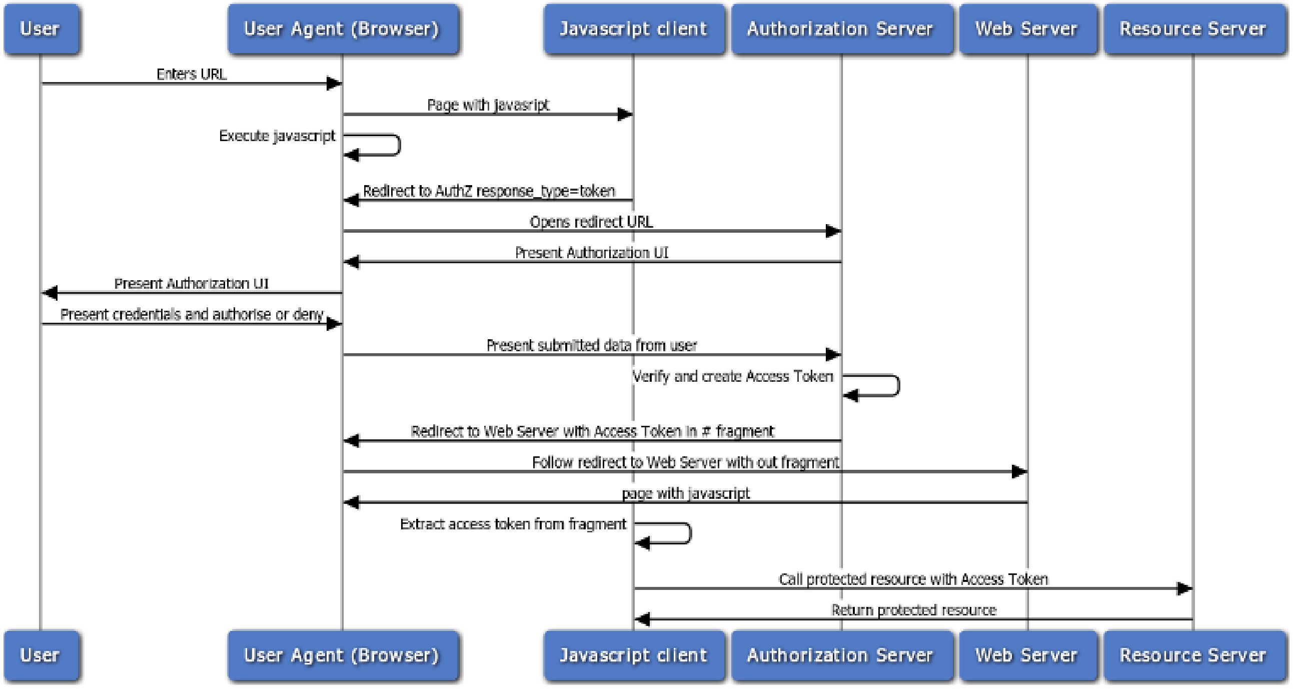
```
Enter Authorization code in dialog
AuthZ code: AaI5Or3RYB2uOgiyqVsLs1ATIY0ll0
Exchange authZ code for access token
Sending up access token request using grant_type set to authorization_code
Response from access token request: 200
Parsing the json response
**********************ACCESS TOKEN RESPONSE***********************************
Access token received from authorization server icPgKP2uVUD2thvAZ5ENhsQb66ffnZEC
 XHyRQEz5zP8aGzcobLV3AR
Access token type received from authorization server Bearer
Access token expiry time: 3599
Refresh token:  NpNbzIVVvj8MhMmcWx2zsawxxJ3YADfc0XIxlZvw0tIhh8
******************************************************************************
Now we can try access the protected resource using the access token
Executing get request on the protected url
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

Implicit Grant (or User Agent) Flow

The Implicit Grant (User-Agent) authentication flow is used by client applications (consumers) residing in the user's device. This could be implemented in a browser using a scripting language such as JavaScript, or from a mobile device or a desktop application. These consumers cannot keep the client secret confidential (application password or private key).

The User Agent flow is as follows:

1. The Web server redirects the user to the API Gateway acting as an Authorization Server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the Web server receives a callback with an access token in the fragment of the redirect URL.
3. After the token is granted, the application can access the protected data with the access token.


User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server
Enters URL
Page with javasript
Execute javascript
Redirect to AuthZ response_type=token
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Access Token
Redirect to Web Server with Access Token in # fragment
Follow redirect to Web Server with out fragment
page with javascript
Extract access token from fragment
Call protected resource with Access Token
Return protected resource
User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server

Obtaining an Access Token

The detailed steps for obtaining an access token are as follows:

1. Redirect the user to the authorization endpoint with the following parameters:

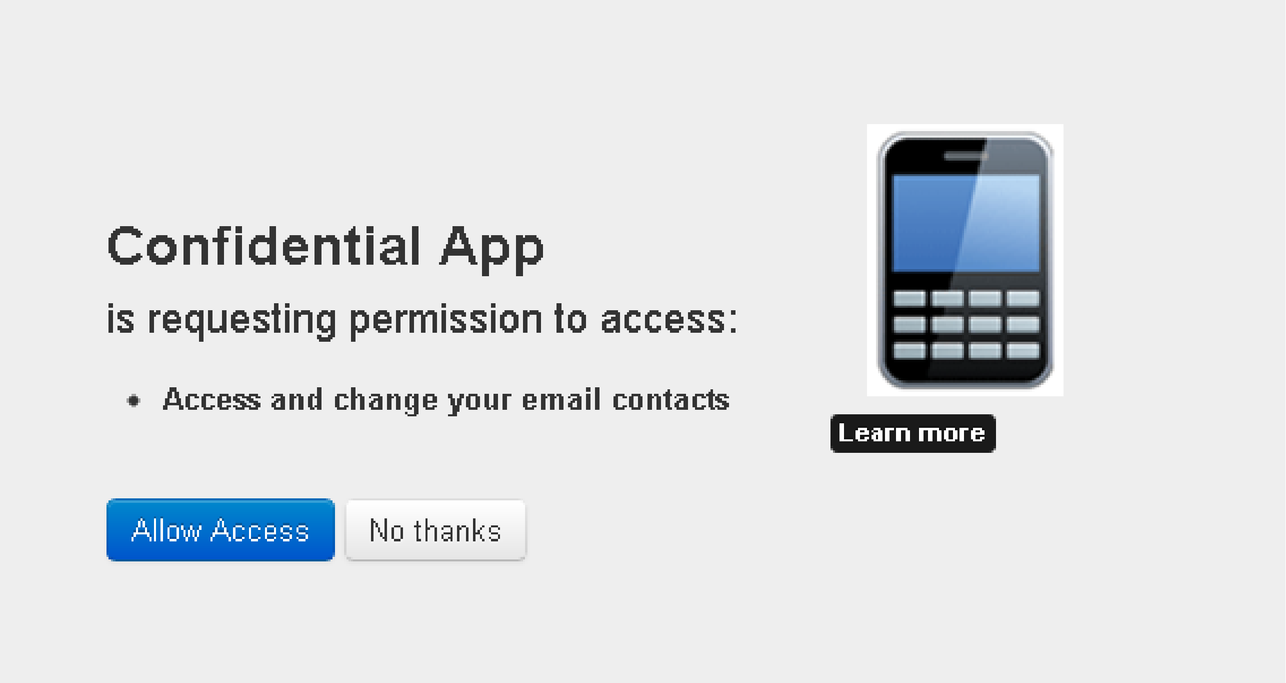
Parameter	Description
`response_type`	Required. Must be set to token.
`client_id`	Required. The Client ID generated when the application was registered in the Oracle API Manager.
`redirect_uri`	Optional. Where the access token will be sent. This value must match one of the values provided in the Oracle API Manager.
`scope`	Optional. A space delimited list of scopes, which indicates the access to the Resource Owner's data requested by the application.
`state`	Optional. Any state the consumer wants reflected back to it after approval during the callback.

The following is an example URL:

```
https://apigateway/oauth/authorize?client_id=SampleConfidentialApp&response_type=
  token&&redirect_uri=http%3A%2F%2Flocalhost%3A8090%2Fauth%2Fredirect.html&scope=
  https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email
```

Note

During this step the Resource Owner user must approve access for the application (Web server) to access their protected resources, as shown in the following example screen.



Confidential App

is requesting permission to access:

- Access and change your email contacts

2. The response to the above request is sent to the `redirect_uri`. If the user approves the access request, the response contains an access token and the state parameter (if included in the request). For example:

```
https://localhost/oauth_callback#access_token=19437jhj2781FQd44AzqT3Zg
 &token_type=Bearer&expires_in=3600
```

If the user does not approve the request, the response contains an error message.

3. After the request is verified, the API Gateway sends a response to the client. The following parameters are contained in the fragment of the redirect:

Parameter	**Description**
`access_token`	The token that can be sent to the Resource Server to access the protected resources of the Resource Owner (user).
`expires`	The remaining lifetime on the access token.
`type`	Indicates the type of token returned. At this time, this field

	will always have a value of Bearer.
state	Optional. If the client application sent a value for state in the original authorization request, the state parameter is populated with this value.

4. After the application has obtained an access token, it can gain access to protected resources on the Resource Server by placing it in an `Authorization: Bearer` HTTP header:

```
GET /oauth/protected HTTP/1.1
Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9
Host: apigateway.com
```

For example, the `curl` command to call a protected resource with an access token is as follows:

```
curl -H "Authorization: Bearer O91G451HZ0V83opz6udiSEjchPynd2Ss9"
https://apigateway.com/oauth/protected
```

Sample Client

The following Jython sample client creates and sends an authorization request for the implicit grant flow to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/implicit_grant.py
```

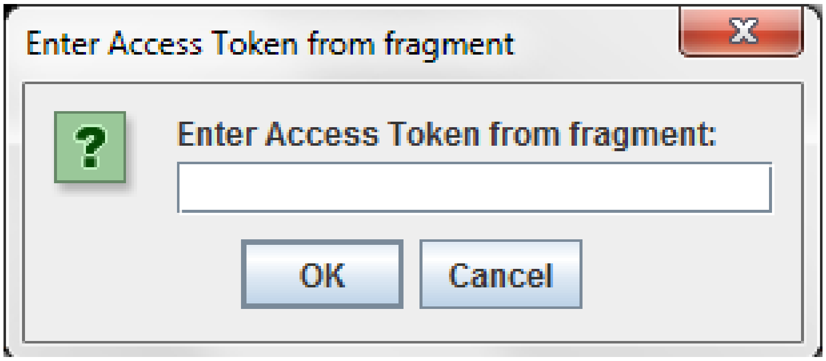
To run the sample, perform the following steps:

1. Open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

   ```
   > run oauth/implicit_grant.py
   ```

   The script outputs the following:

   ```
   > Go to the URL here:
    http://127.0.0.1:8080/api/oauth/authorize?client_id=SampleConfidentialApp&
     response_type=token&scope=https%3A%2F%2Flocalhost%3A8090%2Fauth%2Fuserinfo.email&
     redirect_uri=https%3A%2F%2Flocalhost%2Foauth_callback&state=1956901292
    Enter Access Token code in dialog
   ```



Enter Access Token from fragment

Enter Access Token from fragment:

2. After the Resource Owner has authorized and approved access to the application, the Authorization Server redirects to the redirection URI a fragment containing the access code. For example:

   ```
   https://localhost/oauth_callback#access_token=
    4owzGyokzLLQB5FH4tOMk7Eqf1wqYfENEDXZ1mGvN7u7a2Xexy2OU9&expires_in=
    3599&state=1956901292&token_type=Bearer
   ```

   In this example, the access token is:

   ```
   4owzGyokzLLQB5FH4tOMk7Eqf1wqYfENEDXZ1mGvN7u7a2Xexy2OU9
   ```

   Enter this value into the **Enter Access Token from fragment** dialog, and the script attempts to access the protected resource using the access token. For example:

   ```
   **********************ACCESS TOKEN RESPONSE******************************
   Access token received from authorization server 4owzGyokzLLQB5FH4tOMk7Eqf1wqYfEN
   EDXZ1mGvN7u7a2Xexy2OU9
   ******************************************************************************
   Now we can try access the protected resource using the access token
   Executing get request on the protected url
   Response from protected resource request is: 200
   <html>Congrats! You've hit an OAuth protected resource</html>
   ```

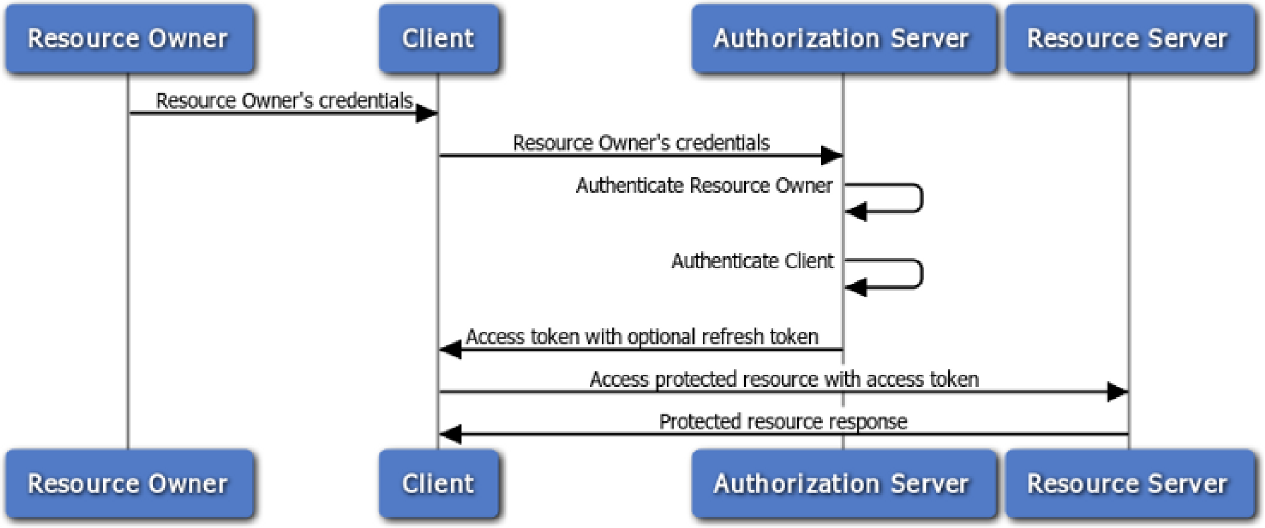
Resource Owner Password Credentials Flow

The Resource Owner password credentials flow is also known as the username-password authentication flow. This flow can be used as a replacement for an existing login when the consumer already has the user’s credentials.

The Resource Owner password credentials grant type is suitable in cases where the Resource Owner has a trust relationship with the client (for example, the device operating system or a highly privileged application). The Authorization Server should take special care when enabling this grant type, and only allow it when other flows are not viable.

This grant type is suitable for clients capable of obtaining the Resource Owner's credentials (username and password, typically using an interactive form). It is also used to migrate existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token.

Resource Owner Password Credentials flow


Resource Owner
Client
Authorization Server
Resource Server
Resource Owner's credentials
Resource Owner's credentials
Authenticate Resource Owner
Authenticate Client
Access token with optional refresh token
Access protected resource with access token
Protected resource response
Resource Owner
Client
Authorization Server
Resource Server

Requesting an Access Token

The client token request should be sent in an HTTP POST to the token endpoint with the following parameters:

Parameter	Description
grant_type	Required. Must be set to password
username	Required. The Resource Owner's user name.
password	Required. The Resource Owner's password.
scope	Optional. The scope of the authorization.
format	Optional. Expected return format. The default is json. Possible values are: • urlencoded • json • xml

The following is an example HTTP POST request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JWgrant_type=password&username=
 johndoe&password=A3ddj3w
```

Handling the Response

The API Gateway will validate the resource owner’s credentials and authenticate the client against the Oracle API Manager. An access token, and optional refresh token, is sent back to the client on success. For example, a valid response is as follows:

```
HTTP/1.1 200 OK
 Cache-Control: no-store
 Content-Type: application/json
 Pragma: no-cache
 {
     "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
     "token_type": "Bearer",
     "expires_in": "3600",
     “refresh_token”: “8722gffy2229220002iuueee7GP..........”
 }
```

Sample Client

The following Jython sample client sends a request to the Authorization Server using the Resource Owner password credentials flow:

```
INSTALL_DIR/samples/scripts/oauth/resourceowner_password_credentials.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

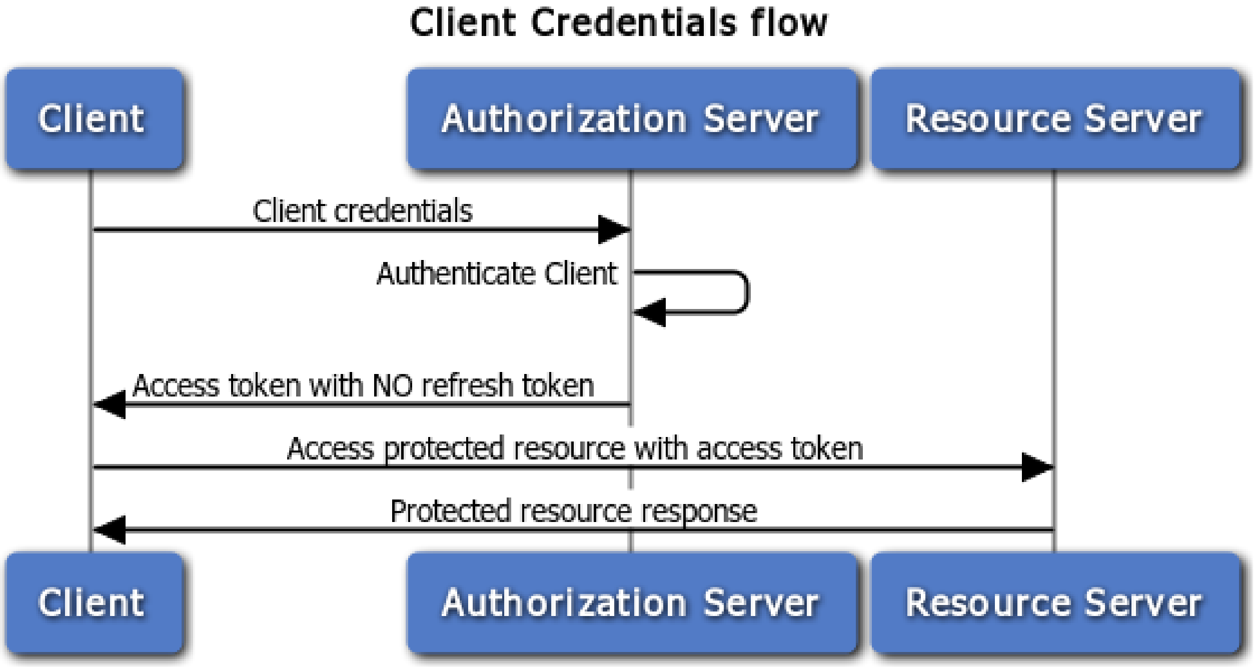
```
> run oauth/resourceowner_password_credentials.py
```

The script outputs the following:

```
Sending up access token request using grant_type set to password
Response from access token request: 200
Parsing the json response
*********************ACCESS TOKEN RESPONSE**********************************
Access token received from authorization server lrGHhFhFwSmycXStIza1jjvXlSaac9
 JNIgviF7oPiV8OnxlSIsrxVA
Access token type received from authorization server Bearer
Access token expiry time: 3600
****************************************************************************
Now we can try access the protected resource using the access token
Excuting get request on the protected url
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

Client Credentials Grant Flow

The client credentials grant type must only be used by confidential clients. The client can request an access token using only its client credentials (or other supported means of authentication) when the client is requesting access to the protected resources under its control. The client can also request access to those of another Resource Owner that has been previously arranged with the Authorization Server (the method of which is beyond the scope of the specification).



Requesting an Access Token
The client token request should be sent in an HTTP `POST` to the token endpoint with the following parameters:

Parameter	Description

`grant_type`	Required. Must be set to `client_credentials`.
`scope`	Optional. The scope of the authorization.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded` • `json` • `xml`

The following is an example POST request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
grant_type=client_credentials
```

Handling the Response

The API Gateway authenticates the client against the Oracle API Manager. An access token is sent back to the client on success. A refresh token is not included in this flow. An example valid response is as follows:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache
{    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
     "token_type": "Bearer",
     "expires_in": "3600"
}
```

Sample Client

The following Jython sample client sends a request to the Authorization Server using the client credentials flow:

```
INSTALL_DIR/samples/scripts/oauth/client_credentials.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/client_credentials.py
```

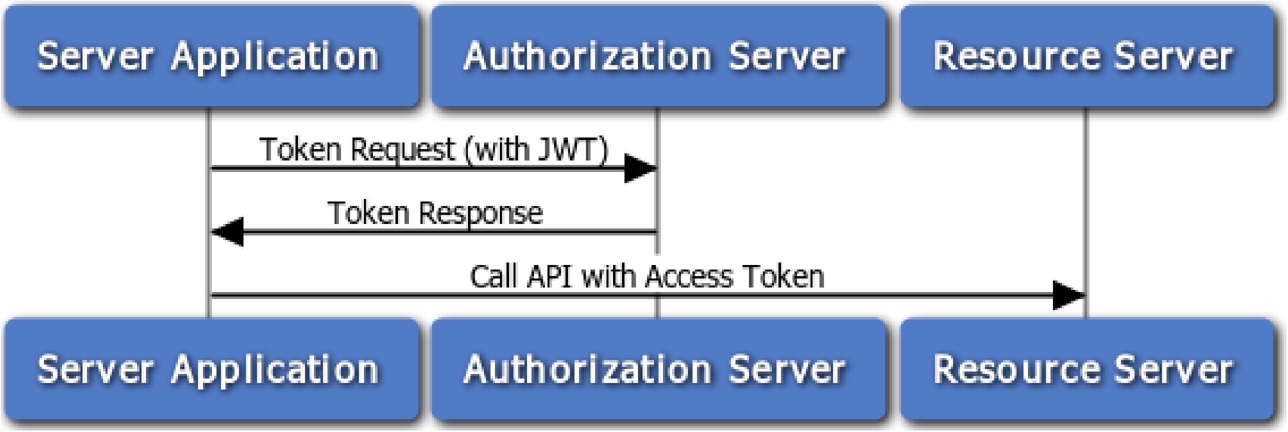
The outputs the following:

```
Sending up access token request using grant_type set to client_credentials
Response from access token request: 200
Parsing the json response
**********************ACCESS TOKEN RESPONSE***********************************
Access token received from authorization server
OjtVvNusLg2ujy3a6IXHhavqdEPtK7qSmIj9fLl8qywPyX8bKEsjqF
Access token type received from authorization server Bearer
Access token expiry time: 3599
```

```
****************************************************************************
Now we can try access the protected resource using the access token
Response from protected resource request is: 200
<html>Congrats! You've hit an OAuth protected resource</html>
```

OAuth 2.0 JWT Flow

A JSON Web Token (JWT) is a JSON-based security token encoding that enables identity and security information to be shared across security domains.



In the OAuth 2.0 JWT flow, the client application is assumed to be a confidential client that can store the client application's private key. The X.509 certificate that matches the client's private key must be registered in the Oracle API Manager. The API Gateway uses this certificate to verify the signature of the JWT claim. For information on creating a private key and certificate, see the section called "OpenSSL Commands".

For more details on the OAuth 2.0 JWT flow, see
http://self-issued.info/docs/draft-ietf-oauth-jwt-bearer-00.html

Creating a JWT Bearer Token
To create a JWT bearer token, perform the following steps:

1. Construct a JWT header in the following format:

   ```
   {"alg":"RS256"}
   ```

2. Base64url encode the JWT Header as defined here, which results in the following:

   ```
   eyJhbGciOiJSUzI1NiJ9
   ```

3. Create a JWT Claims Set, which conforms to the following rules:
   - The issuer (`iss`) must be the OAuth `client_id` or the remote access application for which the developer registered their certificate.
   - The audience (`aud`) must match the value configured in the JWT filter. By default, this value is as follows:

     ```
     http://apigateway/api/oauth/token
     ```

   - The validity (`exp`) must be the expiration time of the assertion, within five minutes, expressed as the number of seconds from `1970-01-01T0:0:0Z` measured in UTC.
   - The time the assertion was issued (`iat`) measured in seconds after `00:00:00` UTC, January 1, 1970.
   - The JWT must be signed (using RSA SHA256).

- The JWT must conform with the general format rules specified here: http://tools.ietf.org/html/draft-jones-json-web-toke.

For example:

```
{
"iss": "SampleConfidentialApp",
    "aud": "http://apigateway/api/oauth/token",
    "exp": "1340452126",
    "iat": "1340451826"
}
```

4. Base64url encode the JWT Claims Set, resulting in:

```
eyJpc3MiOiJTYW1wbGVDb25maWRlbnRpYWxBcHAiLCJhdWQiOiJodHRwOi8vYXBpc2VydmV
  yL2FwaS9vYXV0aC90b2tlbiIsImV4cCI6IjEzNDA0NTIxMjYiLCJpYXQiOiIxMzQwNDUxODI2In0=
```

5. Create a new string from the encoded JWT header from step 2, and the encoded JWT Claims Set from step 4, and append them as follows:

```
Base64URLEncode(JWT Header) + . + Base64URLEncode(JWT Claims Set)
```

This results in a string as follows:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHRw
 Oi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiAi
 MTM0MTM1NDMwNSJ9
```

6. Sign the resulting string in step 5 using SHA256 with RSA. The signature must then be Base64url encoded. The signature is then concatenated with a `.` character to the end of the Base64url representation of the input string. The result is the following JWT (line breaks added for clarity):

```
{Base64url encoded header}.
{Base64url encoded claim set}.
```

This results in a string as follows:

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHR
wOi8vYXBpc2VydmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiA
iMTM0MTM1NDMwNSJ9.ilWR8O8OlbQtT5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp4dx3
KdEDj4YcsyCEAPhfopUlZO3LE-iNPlbxB5dsmizbFIc2oGZr7Zo4IlDf92OJHq9DGqwQosJ-s9GcIRQk
-IUPF4lVy1Q7PidPWKR9ohm3c2gt8
```

Requesting an Access Token

The JWT bearer token should be sent in an HTTP `POST` to the Token Endpoint with the following parameters:

Parameter	Description
`grant_type`	Required. Must be set to `urn:ietf:params:oauth:grant-type:jwt-bearer`.
`assertion`	Required. Must be set to the JWT bearer token, base64url-encoded.
`format`	Optional. Expected return format. The default is `json`. Possible values are: • `urlencoded`

	• json • xml

The following is an example POST request:

```
POST /api/oauth/token HTTP/1.1
Content-Length: 424
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=eyJhbGciOiJS
 UzI1NiJ9.eyJpc3MiOiAiU2FtcGxlQ29uZmlkZW50aWFsQXBwIiwgImF1ZCI6ICJodHRwOi8vYXBpc2Vy
 dmVyL2FwaS9vYXV0aC90b2tlbiIsICJleHAiOiAiMTM0MTM1NDYwNSIsICJpYXQiOiAiMTM0MTM1NDMwN
 SJ9.ilWR8O8OlbQtT5zBaGIQjveOZFIWGTkdVC6LofJ8dN0akvvD0m7IvUZtPp4dx3KdEDj4YcsyCEAPh
 fopUlZO3LE-iNPlbxB5dsmizbFIc2oGZr7Zo4IlDf92OJHq9DGqwQosJ-s9GcIRQk-IUPF4lVy1Q7PidP
 WKR9ohm3c2gt8
```

Handling the Response
The API Gateway returns an access token if the JWT claim and access token request are properly formed, and the JWT has been signed by the private key matching the registered certificate for the client application in the Oracle API Manager.

For example, a valid response is as follows:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
Pragma: no-cache
{
    "access_token": “O91G451HZ0V83opz6udiSEjchPynd2Ss9......",
    "token_type": "Bearer",
    "expires_in": "3600",
}
```

Sample Client
The following Jython sample creates and sends a JWT Bearer token to the Authorization Server:

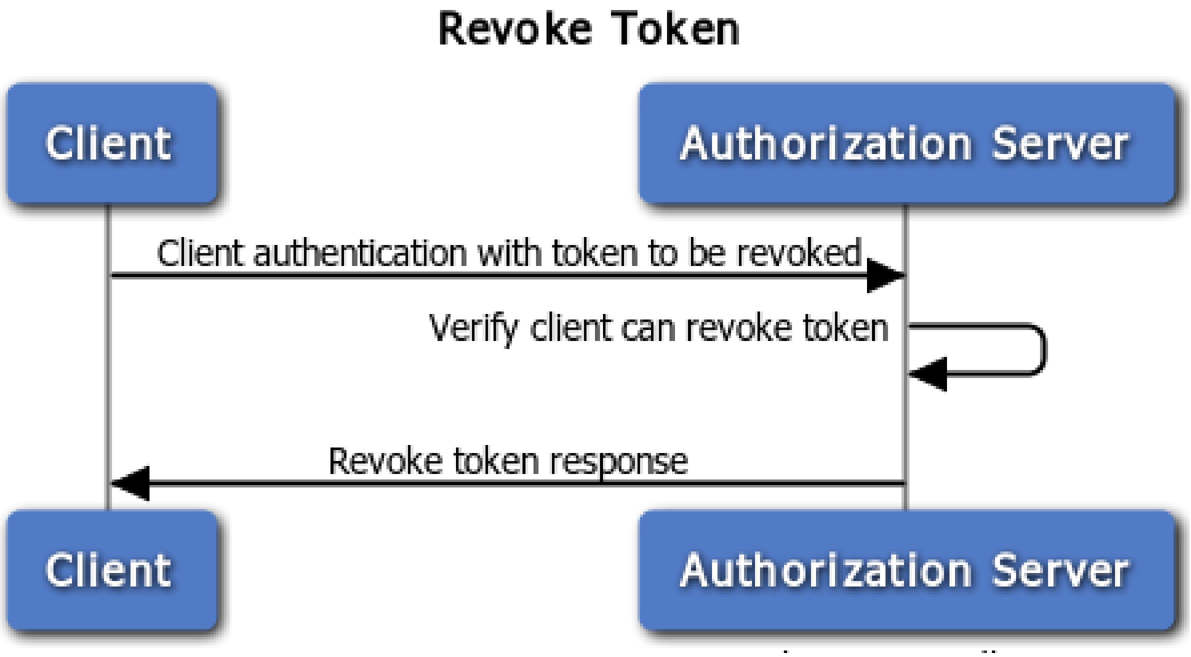
```
INSTALL_DIR/samples/scripts/oauth/jwt.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/jwt.py
```

Revoke Token

In some cases a user may wish to revoke access given to an application. An access token can be revoked by calling the API Gateway revoke service and providing the access token to be revoked. A revoke token request causes the removal of the client permissions associated with the particular token to access the end-user's protected resources.



The endpoint for revoke token requests is as follows:

```
https://<API Gateway>:8089/api/oauth/revoke
```

The token to be revoked should be sent to the revoke token endpoint in an HTTP POST with the following parameter:

Parameter	Description
token	Required. A token to be revoked (for example, 4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4).

The following is an example POST request:

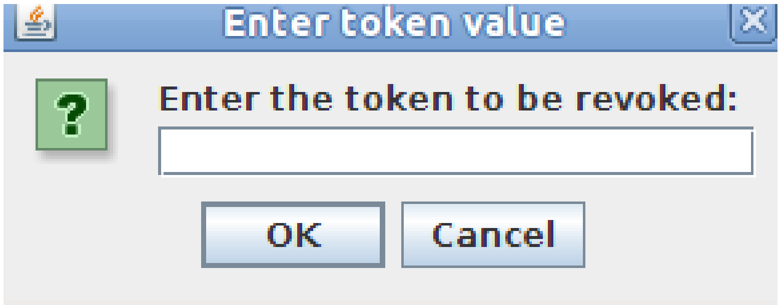
```
POST /api/oauth/revoke HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: 192.168.0.48:8080
Authorization: Basic U2FtcGxlQ29uZmlkZW50aWFsQXBwOjY4MDhkNGI2LWVmMDktNGIwZC04ZjI4LT
NiMDVkYTljNDhlYw==token=4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4
```

Sample Client
The following Jython sample client creates a token revoke request to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/revoke_token.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/revoke_token.py
```



Enter the token to be revoked:

When the Authorization Server receives the token revocation request, it first validates the client credentials and verifies whether the client is authorized to revoke the particular token based on the client identity.

Note

Only the client that was issued the token can revoke it.

The Authorization Server decides whether the token is an access token or a refresh token:

- If it is an access token, this token is revoked.
- If it is a refresh token, all access tokens issued for the refresh token are invalidated, and the refresh token is revoked.

Response Codes
The following HTTP status response codes are returned:

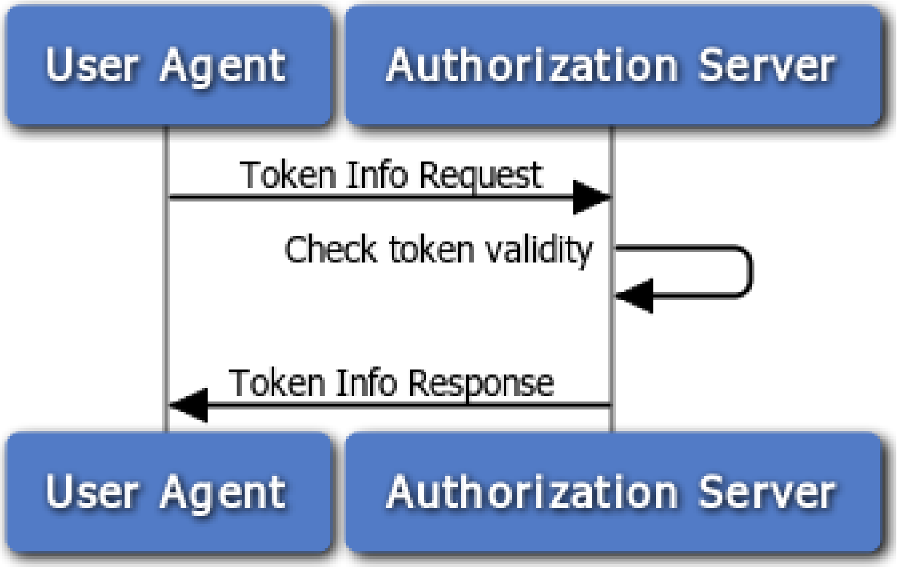
- HTTP 200 if processing is successful.
- HTTP 401 if client authentication failed.
- HTTP 403 if the client is not authorized to revoke the token.

The following is an example response:

```
Token to be revoked: 3eXnUZzkODNGb9D94Qk5XhiV4W4gu9muZ56VAYoZiot4WNhIZ72D3
Revoking token...............
Response from revoke token request is: 200
Successfully revoked token
```

Token Info Service

You can use the **Token Info Service** to validate that an access token issued by the API Gateway. A request to the `tokenInfo` service is an HTTP `GET` request for information in a specified OAuth 2.0 access token.



The endpoint for the token information service is as follows:

```
https://<apigateway>:8089/api/oauth/tokeninfo
```

Getting information about a token from the Authorization Server only requires a `GET` request to the tokeninfo endpoint. For example:

```
GET /api/oauth/tokeninfo HTTP/1.1
Host: 192.168.0.48:8080
access_token=4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4
```

This request includes the following parameter:

Parameter	Description
`access_token`	Required. A token that you want information about (for example: `4eclEUX1N6oVIOoZBbaDTI977SV3T9KqJ3ayOvs4gqhGA4`)

The following example uses this parameter:

```
https://apigateway/api/oauth/tokeninfo?access_token=4eclEUX1N6oVIOoZBba
 DTI977SV3T9KqJ3ayOvs4gqhGA4
```

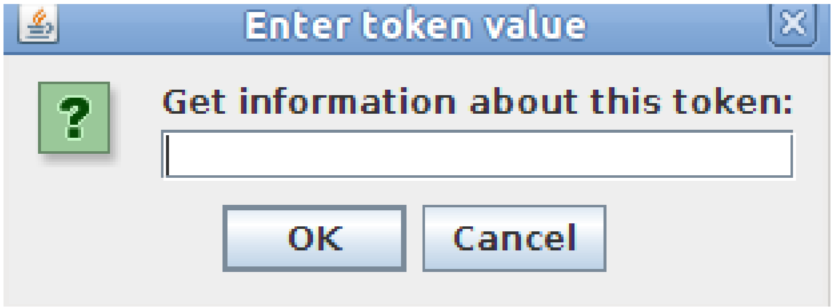
Sample Client
The following Jython sample client creates a token revoke request to the Authorization Server:

```
INSTALL_DIR/samples/scripts/oauth/token_info.py
```

To run the sample, open a shell prompt at `INSTALL_DIR/samples/scripts`, and execute the following command:

```
> run oauth/token_info.py
```

This displays the following dialog:



When the Authorization Server receives the Token Info request, it first ensures the token is in its cache (EhCache or Database), and ensures the token is valid and has not expired.

The following is an example response:

```
Get token info for this token: BcYGjPOQSCrtbEc1F0ag8zf6OT9rCaMLiI1dYjFLT5zhxz3x5ScrdN
Response from token info request is: 200
**********************TOKEN INFO RESPONSE***********************************
Token audience received from authorization server: SampleConfidentialApp
Scopes user consented to: https://localhost:8090/auth/userinfo.email
Token expiry time: 3566
User id : admin
******************************************************************************
```

Response Codes
The following HTTP Status codes are returned:

- 200 if processing is successful
- 400 on failure

The response is sent back as a JSON message. For example:

```
{
 "audience" : "SampleConfidentialApp",
   "user_id" : "admin",
   "scope" : "https://localhost:8090/auth/userinfo.email",
   "expires_in" : 2518
 }
```

You can get additional information about the access token using message attributes. For more details, see the section called “OAuth 2.0 Message Attributes”.

OAuth Access Token Information

Overview

The OAuth 2.0 **Access Token Information** filter is used to return a JSON description of the specified OAuth 2.0 access token. OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions.

An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes. For details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

Access Token Info Settings

Configure the following fields on this tab:

Verify access token is in this cache:
Click the browse button to select where to verify that the access token is present (for example, in the default **OAuth Access Token Store**). To add a store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time in seconds that the database or cache is polled for expired tokens. Defaults to `60` seconds.

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Where to get access token from?:
Select one of the following:

- **In Query String**:
  This is the default setting. Defaults to the `access_token` parameter.
- **In a selector**:
  Defaults to the `${http.client.getCgiArgument('access_token')}` selector. For more details on API Gateway selectors, see *Selecting Configuration Values at Runtime*.

Return additional Access Token parameters:
Click **Add** to return additional access token parameters, and enter the **Name** and **Value** in the dialog. For example, you could enter `Department` in **Name**, and the following selector in **Value**:

```
${accesstoken.getAdditionalInformation().get("Department")
```

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Access Token using Authorization Code

Overview

The OAuth 2.0 **Access Token using Authorization Code** filter is used to get a new access token using the authorization code. This supports the OAuth 2.0 Authorization Code Grant or Web server authentication flow, which is used by applications that are hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application Validation

Configure the following fields on this tab:

Use this store to validate the Authorization Code:
Click the browse button to select the store in which to validate the authorization code (for example, in the default **Authz Code Store**). To add a store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired authorization codes. Defaults to every `60` seconds.

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Find client application information from message:
Select one of the following:

- **In Authorization Header**
  This is the default setting.
- **In Query String**:
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Access Token

Configure the following fields on the this tab:

Cache Access Token here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Access Token using Client Credentials

Overview

The OAuth 2.0 **Access Token using Client Credentials** filter enables an OAuth client to request an access token using only its client credentials. This supports the OAuth 2.0 Client Credentials flow, which is used when the client application needs to directly access its own resources on the Resource Server. Only the client application's credentials or public/private key pair are used in the this flow, the Resource Owner's credentials are not required. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application Validation

Configure the following fields on this tab:

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Find client application information from message:
Select one of the following:

- **In Authorization Header:**
  This is the default setting.
- **In Query String:**
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Access Token

Configure the following fields on the this tab:

Cache Access Token Here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*

- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Access Token using JWT

Overview

The OAuth 2.0 **Access Token using JWT** filter enables an OAuth client to request an access token using only a JSON Web Token (JWT). This supports the OAuth 2.0 JWT flow, which is used when the client application needs to directly access its own resources on the Resource Server. Only the client JWT token is used in this flow, the Resource Owner's credentials are not required. A JWT token is a JSON-based security token encoding that enables identity and security information to be shared across security domains. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application Validation

Configure the following fields on this tab:

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Audience (aud) must contain the following URI:
Enter the JWT `aud` (intended audience). The JWT must contain an `aud` URI that identifies the Authorization Server, or service provider domain, as an intended audience. The Authorization Server must also verify that it is an intended audience for the JWT. Defaults to `http://apiserver/api/oauth/token`.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Access Token

Configure the following fields on the this tab:

Cache Access Token Here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is unselected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Authorization Code Flow

Overview

The OAuth 2.0 **Authorization Code Flow** filter is used to consume OAuth authorization requests. This supports the OAuth 2.0 Authorization Code Grant or Web server authentication flow, which is used by applications hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. The Web server flow is suitable for clients capable of interacting with the end-user's user-agent (typically a Web browser), and capable of receiving incoming requests from the Authorization Server (acting as an HTTP server).

The OAuth 2.0 Authorization Code Grant flow is as follows:

1. The Web server redirects the user to the API Gateway acting as an Authorization Server to authenticate and authorize the server to access data on their behalf.
2. After the user approves access, the Web server receives a callback with an authorization code.
3. After obtaining the authorization code, the Web server passes back the authorization code to obtain an access token response.
4. After validating the authorization code, the API Gateway passes back a token response to the Web server.
5. After the token is granted, the Web server accesses their data.

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Validation/Templates

Configure the following fields on this tab:

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Authorize Resource Owner:
Select one of the following:

- **Use internal flow**
  Uses the internal API Gateway flow to authorize the Resource Owner. This is the default setting.
- **Call this policy**

Click the browse button to select a policy to authorize the Resource Owner. You can use the **Policy will store subject in selector** text box to specify where the policy is stored. Defaults to the `${authentication.subject.id}` message attribute. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Authz Code Details

Configure the following fields on the this tab:

Cache Authorization Code here:
Click the browse button to select where to cache the access token (for example, in the default **Authz Code Store**). To add an access token store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Location of Access Code Redirect Page:
Enter the full path to the HTML page used for the access code HTTP redirect. Defaults to the following:

```
${environment.VDISTDIR}/samples/oauth/templates/showAccessCode.html
```

Authz Code Length:
Enter the number of characters in the authorization code. Defaults to `30`.

Authz Code Expiry (in secs):
Enter the number of seconds before the authorization code expires. Defaults to `600` (ten minutes).

Access Token Details

Configure the following fields on the this tab:

Cache Access Token here:
Click the browse button to select where to cache the access token (for example, in the default `OAuth Access Token Store`). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Additional parameters to store for this Access Token:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Record Outbound Transactions
Select whether to record outbound message traffic. You can use this setting to override the **Record Outbound Transactions** setting on the **System Settings** -> **Traffic Monitor** screen. This setting is selected by default.

Authorize Transaction

Overview

The OAuth 2.0 **Authorize Transaction** filter is used to authorize the Resource Owner and grant (allow/deny) client access to the resources. This supports the OAuth 2.0 Authorization Code Grant or Web server authentication flow, which is used by applications hosted on a secure server. A critical aspect of this flow is that the server must be able to protect the issued client application's secret. The Web server flow is suitable for clients capable of interacting with the end-user's user-agent (typically a Web browser), and capable of receiving incoming requests from the Authorization Server (acting as an HTTP server).

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

Validation/Templates

Configure the following fields on this tab:

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

HTML Templates:
Specify the following templates for HTML forms:

- **Login Form:**
  Enter the full path to the HTML form that the Resource Owner can use to log in. Defaults to the following:

  ```
  ${environment.VDISTDIR}/samples/oauth/templates/login.html
  ```

- **Authorization Form:**
  Enter the full path to the HTML form that the Resource Owner can use to grant (allow/deny) client access to the resources. Defaults to the following:

  ```
  ${environment.VDISTDIR}/samples/oauth/templates/requestAccess.html
  ```

Authz Code Details

Configure the following fields on the this tab:

Cache Authorization Code here:
Click the browse button to select where to cache the access token (for example, in the default `Authz Code Store`). To add an access token store, right-click **Authorization Code Stores**, and select **Add Authorization Code Store**. You can

select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Location of Access Code Redirect Page:
Enter the full path to the HTML page used for the access code HTTP redirect. Defaults to the following:

```
${environment.VDISTDIR}/samples/oauth/templates/showAccessCode.html
```

Authz Code Length:
Enter the number of characters in the authorization code. Defaults to `30`.

Authz Code Expiry (in secs):
Enter the number of seconds before the authorization code expires. Defaults to `600` (ten minutes).

Access Token Details

Configure the following fields on the this tab:

Cache Access Token here:
Click the browse button to select where to cache the access token (for example, in the default `OAuth Access Token Store`). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Additional parameters to store for this Access Token:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department`, `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Refresh Access Token

Overview

The OAuth 2.0 **Refresh Access Token** filter enables an OAuth client to get a new access token using a refresh token. This filter supports the OAuth 2.0 Refresh Token flow. After the client consumer has been authorized for access, they can use a refresh token to get a new access token (session ID). This is only done after the consumer already has received an access token using either the Web Server or User-Agent flow. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

Application Validation

Configure the following fields on this tab:

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Find client application information from message:
Select one of the following:

- **In Authorization Header:**
  This is the default setting.
- **In Query String:**
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Access Token

Configure the following fields on the this tab:

Cache Access Token Here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):

Enter the number of seconds before the access token expires. Defaults to 3600 (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to 54.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to Bearer.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to 43200 (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to 46.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, Department and Engineering).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used. For details on the fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Resource Owner Credentials

Overview

The OAuth 2.0 **Resource Owner Credentials** filter is used to directly obtain an access token and an optional refresh token. This supports the OAuth 2.0 Resource Owner Password Credentials flow, which can be used as a replacement for an existing login when the consumer client already has the user's credentials. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. An OAuth access token can be sent to the Resource Server to access the protected resources of the Resource Owner (user). This token is a string that denotes a specific scope, lifetime, and other access attributes.

Application Validation

Configure the following fields on this tab:

Authenticate Resource Owner
Select one of the following:

- **Authenticate credentials using this repository:**
  Select one of the following from the list:
  - `Simple Active Directory Repository`
  - `Local User Store`
- **Call this policy:**
  Click the browse button to select a policy to authenticate the Resource Owner. You can use the **Policy will store subject in selector** text box to specify where the policy is stored. Defaults to the `${authentication.subject.id}` message attribute. For more details on selectors, see *Selecting Configuration Values at Runtime*.

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.
>

Find client application information from message:
Select one of the following:

- **In Authorization Header:**
  This is the default setting.
- **In Query String:**
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Validate Scopes:
Select whether to validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. For example, select **Libraries** -> **OAuth Scopes** in the Policy Studio to view the default scopes:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Access Token

Configure the following fields on the this tab:

Cache Access Token here:
Click the browse button to select where to cache the access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Access Token Expiry (in secs):
Enter the number of seconds before the access token expires. Defaults to `3600` (one hour).

Access Token Length:
Enter the number of characters in the access token. Defaults to `54`.

Access Token Type:
Enter the access token type. This provides the client with information required to use the access token to make a protected resource request. The client cannot use an access token if it does not understand the token type. Defaults to `Bearer`.

Include Refresh Token:
Select whether to include a refresh token. This is a token issued by the Authorization Server to the client that can be used to obtain a new access token. This setting is selected by default.

Refresh Token Expiry (in secs):
When **Include Refresh Token** is selected, enter the number of seconds before the refresh token expires. Defaults to `43200` (twelve hours).

Refresh Token Length:
When **Include Refresh Token** is selected, enter the number of characters in the refresh token. Defaults to `46`.

Store additional Access Token parameters:
Click **Add** to store additional access token parameters, and enter the **Name** and **Value** in the dialog (for example, `Department` and `Engineering`).

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Record Outbound Transactions
Select whether to record outbound message traffic. You can use this setting to override the **Record Outbound Transactions** setting on the **System Settings** -> **Traffic Monitor** screen. This setting is selected by default.

Revoke a Token

Overview

The OAuth 2.0 **Revoke a Token** filter is used to revoke a specified OAuth 2.0 access or refresh token. A revoke token request causes the removal of the client permissions associated with the specified token used to access the user's protected resources. For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions. OAuth refresh tokens are tokens issued by the Authorization Server to the client that can be used to obtain a new access token.

Revoke Token Settings

Configure the following fields on this tab:

Revoke token from this cache:
Click the browse button to select the cache to revoke the token from (for example, the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

The application registry is stored in the following KPS:
Enter the Key Property Store (KPS) in which the application registry is stored. The application registry contains the applications registered with the Authorization Server that are permitted access to specific scopes and resources. Defaults to the example `ClientApplicationRegistry`, which is available at the following URL:

```
http://localhost:8089/appregistry/
```

For more details, see the topic on *Key Property Stores*.

Find client application information from message:
Select one of the following:

- **In Authorization Header:**
  This is the default setting.
- **In Query String:**
  The **Client Id** defaults to `client_id`, and **Client Secret** defaults to `client_secret`.

Monitoring

The settings on this tab configure service-level monitoring options such as whether the service stores usage metrics data to a database. This information can be used by the web-based API Gateway Manager tool to display service use, and by the Oracle API Gateway Analytics tool to produce reports on how the service is used.

Monitoring Options
For details on the **Monitoring Options** fields on this tab, see the **Monitoring Options** in *Set Service Context*.

Validate Access Token

Overview

The OAuth 2.0 **Validate Access Token** filter is used to validate a specified access token contained in persistent storage. OAuth access tokens are used to grant access to specific resources in an HTTP service for a specific period of time (for example, photos on a photo sharing website). This enables users to grant third-party applications access to their resources without sharing all of their data and access permissions.

For more details on supported OAuth flows, see *API Gateway OAuth 2.0 Authentication Flows*.

Configuration

Configure the following fields on this tab:

Name:
Enter a suitable name for this filter.

Verify access token is in this cache:
Click the browse button to select the cache in which to verify access token (for example, in the default **OAuth Access Token Store**). To add an access token store, right-click **Access Token Stores**, and select **Add Access Token Store**. You can select to **Store in a cache** or **Store in a database**. For more details, see the following topics:

- *Global Caches*
- *Database Connection*

The **Purge expired tokens every** setting specifies the time interval in seconds that the database or cache is polled for expired tokens. Defaults to every `60` seconds.

Location of access token:
Select one of the following:

- **In Authorization Header with prefix:**
  The access token is in the Authorization header with the selected prefix. Defaults to `Bearer`. This is the default option.
- **In Query String with name:**
  The access token is in the HTTP query string with the name specified in the text box.
- **In Form POST with name:**
  The access token is in the HTTP form `POST` request with the name specified in the text box.
- **In Attribute:**
  The access token is in the API Gateway message attribute specified in the text box.

Only accept access tokens that have been issued with the following scopes:
Select the OAuth scope(s) from the list registered in the API Gateway. For example, the default scopes are as follows:

```
https://localhost:8090/auth/user.photos
https://localhost:8090/auth/userinfo.email
```

Oracle Access Manager Authorization

Overview

This filter enables you to authorize an authenticated user for a particular resource against Oracle Access Manager (OAM). The user must first have been authenticated to OAM using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter. After successful authentication, OAM issues a Single Sign On (SSO) token, which can then be used instead of the user name and password.

Configuration

Configure the following fields to authorize a user for a particular resource against Oracle Access Manager:

Name:
Enter a descriptive name for this filter.

Attribute Containing SSO Token:
Enter the name of the message attribute that contains the user's SSO token. This attribute will have been populated when authenticating to Oracle Access Manager using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter. By default, the SSO token is stored in the `oracle.sso.token` message attribute.

Resource Type:
Enter the type of the resource for which you are requesting access. For example, when seeking access to a Web-based URL, specify `http`.

Resource Name:
Enter the name of the resource for which the user is requesting access. The default is `//`*`hostname`*`${http.request.uri}`, which contains the original path requested by the client.

Operation:
In most access management products, it is common to authorize users for a limited set of actions on the requested resource. For example, users with management roles may be able to write (HTTP POST) to a certain Web Service, but users with more junior roles might only have read access (HTTP GET) to the same service.

You can use this field to specify the operation that you want to grant the user access to on the specified resource. By default, this field is set to the `http.request.verb` message attribute, which contains the HTTP verb used by the client to sent the message to the API Gateway (for example, POST).

Oracle Access Manager Log in with Certificate

Overview

This filter enables authentication to Oracle Access Manager (OAM) using an X.509 certificate presented by the client. After successful authentication, OAM issues a Single Sign On (SSO) token, which can then be used by the client for subsequent calls to the virtualized service.

General Configuration

Configure the following general settings:

Name:
Enter an appropriate name for this filter.

Attribute containing X509 certificate:
Enter the name of the message attribute that contains the user's X.509 certificate. By default, this is stored in the `certificate` message attribute.

Attribute to contain SSO token id:
Enter the name of the message attribute to contain the user's SSO token. By default, the SSO token is stored in the `oracle.sso.token` message attribute.

Resource Configuration

Configure the following resource settings:

Resource Type:
Enter the type of the resource for which you are requesting access. For example, when seeking access to a Web-based URL, enter `http`.

Resource Name:
Enter the name of the resource for which the user is requesting access. By default, this field is set to `//hostname${http.request.uri}`, which contains the original path requested by the client.

Operation:
In most access management products, it is common to authorize users for a limited set of actions on the requested resource. For example, users with management roles may be able to write (HTTP POST) to a certain Web Service, but users with more junior roles might only have read access (HTTP GET) to the same service.

You can use this field to specify the operation that you want to grant the user access to on the specified resource. By default, this field is set to the `http.request.verb` message attribute, which contains the HTTP verb used by the client to sent the message to the API Gateway (for example, POST).

Include query string:
Select whether the query string parameters are used by the OAM server to determine the policy that protects this resource. This setting is optional if the policies configured do not rely on the query string parameters.

Session Configuration

Configure the following session settings:

Location:
If the client location must be passed to OAM for it to make its decision, you can enter a valid DNS name or IP address to specify this location.

Parameters:
You can add optional additional parameters to be used in the authentication decision. The available optional parameters include the following:

`ip`	IP address, in dotted decimal notation, of the client accessing the resource.
`operation`	Operation attempted on the resource (for HTTP resources, one of `GET`, `POST`, `PUT`, `HEAD`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, or `OTHER`).
`resource`	The requested resource identifier (for HTTP resources, the full URL).
`targethost`	The host (`host:port`) to which resource request is sent.

Note

One or more of these optional parameters may be required by certain authentication schemes, modules, or plugins configured in the OAM server. To determine which parameters to add, see your OAM server configuration and documentation.

OAM Access Server SDK Configuration

Configure the following setting:

OAM Access Server SDK Directory:
Enter the path to your OAM Access Server SDK directory. For more details on the OAM Access Server SDK, see your Oracle Access Manager documentation.

Logout from Oracle Access Manager SSO Session

Overview

This filter enables you to log out a session from Oracle Access Manager by invalidating the SSO token that is associated with this session.

Configuration

Configure the following fields to explicitly log out (invalidate) an SSO token from Oracle Access Manager:

Name:
Enter a descriptive name for this filter.

Attribute Containing SSO Token ID:
Enter the name of the message attribute that contains the SSO token that you want to validate. This attribute will have been populated when authenticating to Oracle Access Manager using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter. By default, the SSO token is stored in the `oracle.sso.token` message attribute.

OAM Access Server SDK Directory:
Enter the path to your OAM Access Server SDK directory. For more details on the OAM Access Server SDK, see your Oracle Access Manager documentation.

Oracle Access Manager SSO Token Validation

Overview

This filter enables you to check an Oracle Access Manager Single Sign On (SSO) token to ensure that it is still valid. The SSO token is issued by Oracle Access Manager (OAM) after the API Gateway authenticates to it on behalf of an end-user using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter. After successfully authenticating to OAM, the SSO token is stored in the `oracle.sso.token` message attribute.

Oracle Access Manager SSO enables a client to send up its user name and password once, and then receive an SSO token (for example, in a cookie or in the XML payload). The client can then send up the SSO token instead of the user name and password.

Configuration

Configure the following fields to validate an SSO token issued by Oracle Access Manager:

Name:
Enter a descriptive name for the filter.

Attribute Containing SSO Token ID:
Enter the name of the message attribute that contains the SSO token that you want to validate. This attribute will have been populated when authenticating to Oracle Access Manager using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filters. By default, the SSO token is stored in the `oracle.sso.token` message attribute.

OAM Access Server SDK Directory:
Enter the path to your OAM Access Server SDK directory. For more details on the OAM Access Server SDK, see your Oracle Access Manager documentation.

Oracle Entitlements Server 10g Authorization

Overview

This filter enables you to authorize an authenticated user for a particular resource against Oracle Entitlements Server (OES) 10g. The user must first have been authenticated to OES 10g (for example, using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter).

This filter enables you to configure the API Gateway to delegate authorization to OES 10g. You can configure the API Gateway to authorize an authenticated user for a particular resource against OES 10g. Credentials used for authentication can be extracted from the HTTP Basic header, WS-Security username token, or the message payload. After successful authentication, the API Gateway can authorize the user to access a resource using OES 10g.

General

Configure the following general field:

Name:
Enter an appropriate descriptive name for this filter.

Settings

Configure the following fields on the **Settings** tab:

Resource:
Enter the URL for the target resource (for example, Web Service). Alternatively, if this policy is reused for multiple services, enter a URL using message attribute selectors, which are expanded at runtime to the value of the specified attribute. For example:

```
${http.destination.protocol}://${http.destination.host}:${http.destination.port}
${http.request.uri}
```

Resource Naming Authority:
Enter `apigatewayResource` to match the Naming Authority Definition loaded in the OES 10g settings. For more details, see *Oracle Security Service Module Settings (10g)*.

Action:
Enter the HTTP verb (for example, `POST`, `GET`, `DELETE`, and so on). Alternatively, if this policy is reused for multiple services, enter a message attribute selector, which is expanded at runtime to the value of the specified attribute (for example, `${http.request.verb}`) For more details on selectors, see *Selecting Configuration Values at Runtime*.

Action Naming Authority:
Enter `apigatewayAction` to match the Naming Authority Definition loaded in the OES 10g settings. For more details, see *Oracle Security Service Module Settings (10g)*.

How access request is processed:
Select one of the following options:

`ONCE`	Specifies that the authorization query is only asked once for a resource and action.
`POST`	Specifies that the authorization query is asked after a resource is acquired, but before it has been processed or presented.
`PRIOR`	Specifies that the authorization query is asked before a resource is acquired.

Application Context

Configure the following field on the **Application Context** tab:

Application's Current Context:
Click **Add** to specify optional Application Contexts as name-value pairs. Enter a **Name** and **Value** in the **Properties** dialog. Repeat to specify multiple properties.

Get Roles from Oracle Entitlements Server 10g

Overview

This filter enables you to get the set of roles that are assigned to an identity for a specific resource (for example, Web Service) and a specific action (for example, HTTP POST) from Oracle Entitlements Server (OES) 10g.

General

Configure the following general field:

Name:
Enter an appropriate descriptive name for this filter.

Settings

Configure the following fields on the **Settings** tab:

Resource:
Enter the URL of the target resource (for example, Web Service). Alternatively, if this policy is reused for multiple services, enter a URL using message attribute selectors, which are expanded at runtime to the value of the specified attribute. For example:

```
${http.destination.protocol}://${http.destination.host}:${http.destination.port}
${http.request.uri}
```

Resource Naming Authority:
Enter `apigatewayResource` to match the Naming Authority Definition loaded in the OES 10g settings. For more details, see *Oracle Security Service Module Settings (10g)*.

Action:
Enter the HTTP verb (for example, `POST`, `GET`, `DELETE`, and so on). Alternatively, if this policy is reused for multiple services, enter a message attribute selector, which is expanded at runtime to the value of the specified attribute (for example, `${http.request.verb}`). For more details on selectors, see *Selecting Configuration Values at Runtime*.

Action Naming Authority:
Enter `apigatewayAction` to match the Naming Authority Definition loaded in the OES 10g settings. For more details, see *Oracle Security Service Module Settings (10g)*.

Application Context

Configure the following field on the **Application Context** tab:

Application's Current Context:
Click **Add** to specify optional Application Contexts as name-value pairs. Enter a **Name** and **Value** in the **Properties** dialog. Repeat to specify multiple properties.

Oracle Entitlements Server 11g Authorization

Overview

This filter enables you to authorize an authenticated user for a particular resource against Oracle Entitlements Server (OES) 11g. The user must first have been authenticated to OES 11g (for example, using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter).

This filter enables you to configure the API Gateway to delegate authorization to OES 11g. You can configure the API Gateway to authorize an authenticated user for a particular resource against OES 11g. Credentials used for authentication can be extracted from the HTTP Basic header, WS-Security username token, or the message payload. After successful authentication, the API Gateway can authorize the user to access a resource using OES 11g.

Configuration

Configure the following fields on the filter screen:

Name:
Enter an appropriate descriptive name for this filter.

Resource:
Enter the URL for the target resource to be authorized (for example, Web Service). Alternatively, if this policy is reused for multiple services, enter a URL using selectors, which are expanded at runtime to the value of the specified attributes. For example:

```
${http.destination.protocol}://${http.destination.host}:${http.destination.port}
${http.request.uri}
```

Action:
Enter the HTTP verb (for example, `POST`, `GET`, `DELETE`, and so on). Alternatively, if this policy is reused for multiple services, enter a selector, which is expanded at runtime to the value of the specified attribute (for example, `${http.request.verb}`). For more details on selectors, see *Selecting Configuration Values at Runtime*.

Environment/Context attributes:
Click **Add** to specify optional Application Contexts as name-value pairs. Enter a **Name** and **Value** in the **Properties** dialog. Repeat to specify multiple properties.

Relative Path Resolver

Overview

The **Relative Path** filter enables you to identify an incoming XML message based on the relative path on which the message is received.

The following example shows how to find the relative path of an incoming message. Consider the following SOAP message:

```
POST /services/helloService HTTP/1.1
Host: localhost:8095
Content-Length: 196
SOAPAction: HelloService
Accept-Language: en-US
UserAgent: API Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <getHello xmlns="http://www.oracle.com/"/>
  </soap:Body>
</soap:Envelope>
```

The relative path for this message is as follows:

```
/services/helloService
```

Configuration

To configure the **Relative Path** filter, complete the following:

1. Enter an appropriate name for the filter in the **Name** field.
2. Enter a regular expression to match the value of the relative path on which messages are received in the **Relative Path** field. For example, enter `^/services/helloService$` to exactly match a path with a value of `/services/helloService`. Incoming messages received on a matching relative path value are passed on to the next filter on the success path in the policy.

SOAP Action Resolver

Overview

The **SOAP Action Resolver** filter enables you to identify an incoming XML message based on the `SOAPAction` HTTP header in the message. The **SOAP Action Resolver** filter applies to SOAP 1.1 and SOAP 1.2.

The following example illustrates how to locate the `SOAPAction` header in an incoming message. Consider the following SOAP message:

```
POST /services/helloService HTTP/1.1
Host: localhost:8095
Content-Length: 196
SOAPAction: HelloService
Accept-Language: en-US
UserAgent: API Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <getHello xmlns="http://www.oracle.com/"/>
  </soap:Body>
</soap:Envelope>
```

The SOAP Action for this message is `HelloService`.

Configuration

To configure the **SOAP Action Resolver** filter, complete the following:

1. Enter an appropriate name for the filter in the **Name** field.
2. Enter a regular expression to match the value of the `SOAPAction` HTTP header in the **SOAP Action** field. For example, enter `^getQuote$` to exactly match a `SOAPAction` header with a value of `getQuote`. Incoming messages with a matching `SOAPAction` value are passed on to the next filter on the success path in the policy.

Operation Name

Overview

The **Operation Name** filter enables you to identify an incoming XML message based on the SOAP Operation in the message.

The following example shows how to find the SOAP Operation of an incoming message. Consider the following SOAP message:

```
POST /services/timeservice HTTP/1.0
Host: localhost:8095
Content-Length: 374
SOAPAction: TimeService
Accept-Language: en-US
UserAgent: API Gateway
Content-Type: text/XML; utf-8

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:getTime xmlns:ns1="Some-URI">
      <ns1:city>Dublin</ns1:city>
    </ns1:getTime>
  </soap:Body>
</soap:Envelope>
```

The SOAP Operation for this message and its namespace are as follows:

SOAP Operation	`getTime`
SOAP Operation Namespace	`urn:timeservice`

The SOAP Operation is the first child element of the SOAP `<Body>` element.

Configuration

To configure the **Operation Name** filter, complete the following:

1. Enter an appropriate name for the filter in the **Name** field.
2. Enter the name of the SOAP Operation in the **Operation** field. Incoming messages with an operation name matching the value entered here are passed on to the next Success filter in the policy.
3. Enter the namespace to which the SOAP Operation belongs in the **Namespace** field.

Getting Started with Routing Configuration

Overview

This topic describes how to configure the API Gateway to send messages to external Web Services. The API Gateway offers a number of different filters that can be used to route messages. Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used.

For example, the API Gateway can act both as a proxy and as an endpoint (in-line) server for a client, depending on how the client is configured. In each case, the request received by the API Gateway appears slightly different, and the API Gateway can take advantage of this when routing the message onwards. Furthermore, the API Gateway can provide *service virtualization* by shielding the underlying hierarchy of back-end Web Services from clients.

This topic explains how clients can use the API Gateway both as a proxy and as an endpoint server. It then shows how *service virtualization* works. When these basic concepts are explained, this topic helps you to identify the combination of routing filters that is best suited to your deployment scenario.

Proxy or Endpoint Server

The API Gateway can be used by clients as both a proxy server and an endpoint server. When a client uses the API Gateway as a proxy server, it sends up the complete URL of the destination Web Service in the HTTP request line. The API Gateway can use the URL to determine the host and port to route the message to. The following HTTP request shows an example of a request received by the API Gateway when acting as a proxy for a client:

```
POST http://localhost:8080/services/getHello HTTP/1.1
```

Alternatively, when the API Gateway is acting as an endpoint (in-line) server, the client sends the request directly to the API Gateway. In this case, the request line appears as follows:

```
POST /services/getHello HTTP/1.1
```

In this case, only the path on the server is specified, and no scheme, host, or port number is included in the HTTP request line. Because this information is not provided by the client, the API Gateway must be explicitly configured to route the message on to the specific destination.

Service Virtualization

It is sometimes desirable to shield the underlying structure of the directory hierarchy in which Web Services reside from external clients. You can do this by providing a mapping between the path that the client accesses and the actual path at which the Web Service resides.

For example, suppose you have two Web Services accessible at the `/helloService/getHello` and `/financeService/getQuote` URIs. You may wish to hide that these services are deployed under different paths, perhaps exposing them under a common `/services` base URI (for example, `/services/getHello` and `/services/getQuote`). The client is therefore unaware of the underlying hierarchy (for example, directory structure) of the two Web Services. This is termed *service virtualization*.

Choosing the Correct Routing Filters

This section first identifies how your clients perceive the API Gateway, and then determines whether you wish to virtualize your back-end Web Services. Depending on these requirements, you can use different combinations of routing filters, as described in the use cases in the subsequent sections.

Consider the following to determine which combination of routing filters is most appropriate for your scenario:

Proxy or Endpoint?

- If the client is using the API Gateway as a proxy server, see cases 1 or 2 below, depending on whether *service virtualization* is required.
- Alternatively, if the client using the API Gateway as the endpoint of the connection (as an in-line server), see cases 3 or 4 below.

Service Virtualization?

- If you want to shield the hierarchy of protected Web Services by exposing a *virtual* view of these services, see cases 2, 4, and 5 below.
- If *service virtualization* is not important, see cases 1 and 3 below.

These permutations are summarized in the following table:

Proxy or Endpoint?	***Service Virtualization?***	***Example***
Proxy	No	Case 1: Proxy without Virtualization
Proxy	Yes	Case 2: Proxy with Virtualization
Endpoint	No	Case 3: Endpoint without Virtualization
Endpoint	Yes	Case 4: Endpoint with Virtualization
Proxy or Endpoint	Yes	Case 5: Simple Redirect

Case 1: Proxy without Service Virtualization

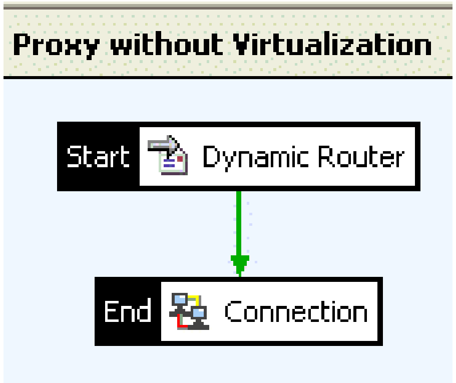
In this case, the API Gateway is configured as an HTTP proxy for the client, and maintains the original path used by the client in the HTTP request. For example, if the API Gateway is listening at `http://localhost:8080/`, and the Web Service is running at `http://localhost:5050/services/getQuote`, the request line of the client HTTP request appears as follows:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

Because the client is configured to use the API Gateway instance running on `localhost` at port `8080` as its HTTP proxy, the client automatically sends all messages to the proxy. However, it includes the full URL of the ultimate destination of the message in the request line of the HTTP request.

When the API Gateway receives the request, it extracts this URL from the request line and uses it to determine the destination of the message. In the above example, the API Gateway routes the message on to `http://localhost:5050/services/getQuote`.

You can configure the following policy to route the message to the URL specified in the request line of the client request:



The following table explains the role of each filter in the policy. For more information on a specific filter, click the appropriate link in the **Details** column.

Filter	***Role in Policy***	***Details***
Dynamic Router	Extracts the URL of the destination Web Service from the request line of the incoming HTTP request. The **Dynamic Router** is normally used when the API Gateway is perceived as a proxy by the client.	*Dynamic Router*
Connection	Establishes the connection to the destination Web Service, and sends the message over this connection. This connection can be mutually authenticated if necessary.	*Connection*

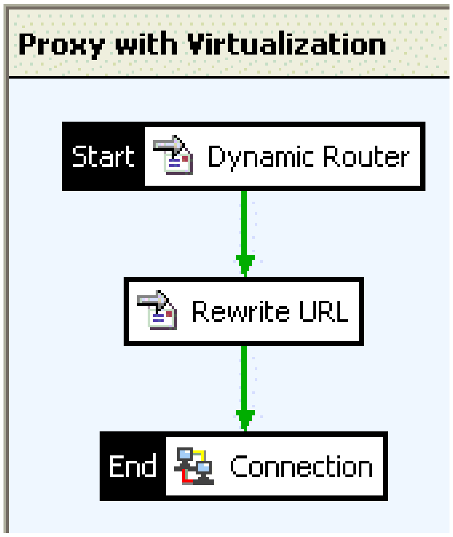
Case 2: Proxy with Service Virtualization

In this case, the API Gateway is also perceived as an HTTP proxy by the client. However, the API Gateway exposes a *virtualized* view of the services that it protects. This is termed *service virtualization*.

To achieve this, the API Gateway must provide a mapping between the path used by the client and the path under which the service is deployed. Assuming the API Gateway is running at `http://localhost:8080/services`, and the Web Service is deployed at `http://localhost:5050/financialServices/quotes/getQuote`, the following example shows what the client may send up in the HTTP request line:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

To achieve this, the API Gateway must provide a mapping between what the client requests (`/services/getQuote`), and the address of the Web Service (`/financialServices/quotes/getQuote`. The **Rewrite URL** filter in the following policy fulfills this role:



The following table explains the roles of the each filter in the policy:

Filter	***Role in Policy***	***Details***
Dynamic Router	Extracts the URL of the destination Web Service from the request line of the incoming HTTP request. The **Dynamic Router** is normally used when the API Gateway is perceived as a proxy by the client.	*Dynamic Router*
Rewrite URL	Specifies the mapping between the path requested by the client and the path under which the Web Service is deployed, therefore providing service virtualization.	*Rewrite URL*
Connection	Establishes the connection to the destination Web Service, and sends the message over this connection. This connection can be mutually authenticated if necessary.	*Connection*

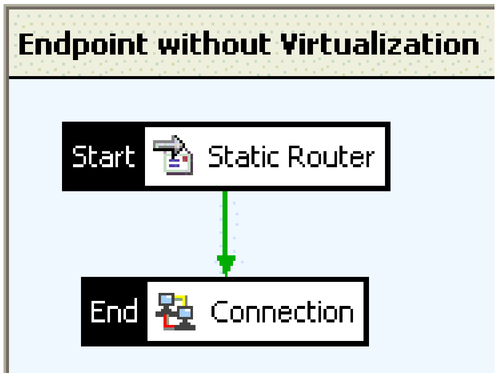
Case 3: Endpoint without Service Virtualization

In this scenario, the client sees the API Gateway as the endpoint to its connection, and the API Gateway must be configured to route messages on to a specific destination. For example, assuming that the API Gateway is running at `http://localhost:8080/services`, the request line of the client's HTTP request is received by the API Gateway as follows:

```
POST /services HTTP/1.1
```

The request line above shows that no information about the scheme, host, or port of the destination Web Service is specified. Therefore, this information must be configured in the API Gateway so that it knows where to route the message on to. The **Static Router** enables the user to enter connection details for the destination Web Service.

Assuming that the Web Service is running at `http://localhost:5050/stockquote/getPrice`, the host, port, and scheme respectively are: `localhost`, `5050`, and `http`. You must explicitly configure this information in the **Static Router**. The following policy illustrates this scenario:



The following table explains the role of each filter in the above policy:

Filter	***Role in Policy***	***Details***
Static Router	Enables the user to explicitly specify the host, port, and scheme at which the Web Service is listening. This filter must be used when the client sees the API Gateway as the endpoint to its connection (the API Gateway is not acting as a proxy for the client).	*Static Router*
Connection	Establishes the connection to the destination Web Service, and sends the message over this connection. This connection can be mutually authenticated if necessary.	*Connection*

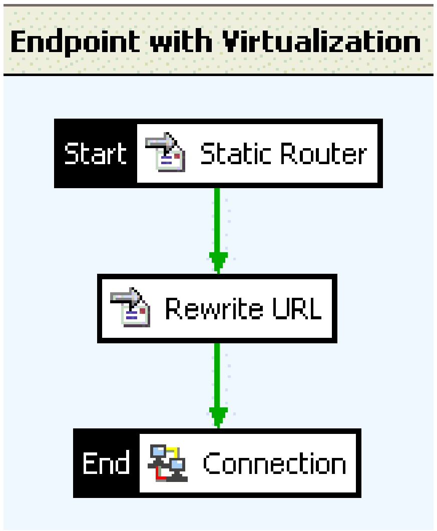
Case 4: Endpoint with Service Virtualization

In this case, the API Gateway acts as the endpoint to the client connection (and not as a proxy), and hides the deployment hierarchy of protected Web Services from clients. In other words, it performs *service virtualization*.

In this scenario, the client sends messages directly to the API Gateway. For example, assuming that the API Gateway is running at `http://localhost:8080/services`, and the Web Service is running at `http://localhost:5050/stockquote/getPrice`, the request line of the client HTTP request is received by the API Gateway as follows:

```
POST /services HTTP/1.1
```

You can then configure the **Static Router** filter to route the message on to port `8080` on `localhost` using the `http` scheme, while the **Rewrite URL** filter provides the mapping between the path requested by the client (`/services`) and the path under which the Web Service is deployed (`/stockquote/getPrice`). The following policy illustrates a sample policy that provides *service virtualization* when the API Gateway is used as an endpoint:



The following table explains the role of each filter in the policy:

Filter	***Role in Policy***	***Details***
Static Router	Enables you to explicitly specify the host, port, and scheme at which the Web Service is listening. This filter can be used when the client sees the API Gateway as the endpoint to its connection (not as a proxy for the client).	*Static Router*
Rewrite URL	Provides the mapping between the path requested by the client and the path under which the Web Service is deployed.	*Rewrite URL*
Connection	Establishes the connection to the destination Web Service, and sends the message over this connection. This connection can be mutually authenticated if necessary.	*Connection*

Important

Alternatively, instead of using the **Static Router**, **Rewrite URL**, and **Connection** filters, you can use the **Connect to URL** filter, which is equivalent to using these three filters combined. You can configure the **Connect to URL** filter to send messages to a Web Service simply by specifying the destination URL. For more details, see the *Connect to URL* topic.

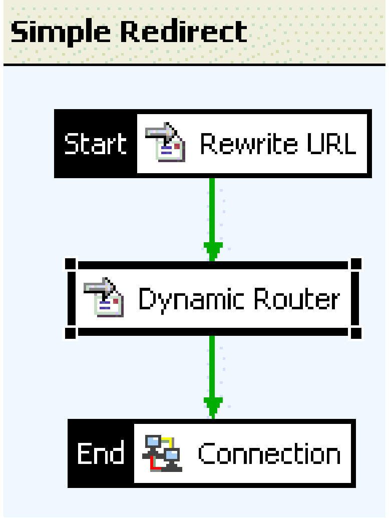
Case 5: Simple Redirect

In some cases, the API Gateway must route the incoming message to an entirely different URL. You can use the **Rewrite URL** filter for this purpose, in addition to rewriting the path on which the request is received (as described in cases 2 and 4 above).

Note

The full URL of the destination Web Service should be specified in this case in the **Rewrite URL** filter.

The following policy illustrates the use of the **Redirect URL** filter to route messages to a fully qualified URL:



The following table describes the role of each filter in the policy:

Filter	***Role in Policy***	***Details***
Rewrite URL	Used to specify the fully qualified URL of the destination Web Service.	*Rewrite URL*
Dynamic Router	In this case, the **Dynamic Router** filter is used to parse the URL specified in the **Rewrite URL** filter into its constituent parts. The HTTP scheme, port, and host of the Web Service are extracted and set to the internal message object for use by the **Connection** filter.	*Dynamic Router*
Connection	Establishes the connection to the destination Web Service, and sends the message over this connection. This connection can be mutually authenticated if necessary.	*Connection*

Case 6: Routing on to an HTTP Proxy

This is a more advanced case where the API Gateway is configured to route on through an HTTP proxy to the back-end Web Service, which sits behind the proxy. When the API Gateway is configured to route through a proxy, it connects directly to the proxy, and sends a request including the full URL of the target Web Service in the HTTP request URI. When the HTTP proxy receives this request, it uses the URL in the request line to determine where to route the message on to. The following example shows the request line of a request made through a proxy:

```
POST http://localhost:8080/services/getQuote HTTP/1.1
```

The following filters are required to configure the API Gateway to route through an HTTP proxy:

Filter	***Role in Policy***	***Details***
Static Router	You must explicitly specify the host, port, and scheme of the HTTP proxy.	*Static Router*
Rewrite URL	Enter the full URL of the Web Service (for example, `http://HOST:8080/myServices`). Because you are routing through a proxy, the full URL is sent in the request line of the HTTP request.	*Rewrite URL*
Connection	In this case, the **Connection** filter connects to the HTTP proxy, which in turn routes the message on to the destination server named in the request URI. The **Send via Proxy** option must be enabled in the **Connection** filter to facilitate this.	*Connection*

Note

It is important to note the differences between how the filters are configured to route on through a proxy and the scenario described in Case 4: Endpoint with Service Virtualization where no proxy is involved:

- **Static Router:**
  When the API Gateway routes on to an HTTP proxy, the **Static Router** filter is configured with the details of the HTTP proxy. Otherwise, the **Static Router** filter is given the details of the Web Service endpoint directly.
- **Rewrite URL:**
  The full URL of the Web Service endpoint must be specified in this filter when the API Gateway routes through a proxy. The full URL is then included in the request line of the HTTP request to the proxy. In cases where no proxy is involved, the **Rewrite URL** filter is only necessary when the back-end Web Services are virtualized. In this case, the API Gateway must send the request to a different URI than that requested by the client.
- **Connection:**
  When routing through a proxy, the **Send via Proxy** option must be enabled in the **Connection** filter. This is not necessary when no proxy sits between the API Gateway and the back-end Web Service.

Summary

The following are the key concepts to consider when configuring the API Gateway to connect to external Web Services:

- The **Connection** or **Connect to URL** filter *must* always be used because it establishes the connection to the Web Service.
- *Service virtualization* can be achieved using the **Rewrite URL** or **Connect to URL** filter.
- If the client is configured to use the API Gateway as a proxy, the API Gateway can use the **Dynamic Router** filter to extract the URL from the request line of the HTTP request. It can then route the message on to this URL.

- If the client sees the API Gateway as the endpoint of the connection (not as a proxy), the **Static Router** filter can be used to explicitly configure the host, port, and scheme of the destination Web Service. Alternatively, you can use the **Connect to URL** to specify a URL.

Finally, the API Gateway provides a **Routing Wizard** to enable you to quickly configure and auto-generate the filters necessary to route messages on to a specific destination. For more details, see the *Routing Wizard* topic.

Routing Wizard

Overview

For convenience, the API Gateway provides a **Routing Wizard** to enable administrators to quickly configure the filters necessary to route messages on to a specific destination. The wizard auto-generates the following filters:

Filter	***Role***	***Details***
Rewrite URL	This filter determines the request URI of the HTTP request that is ultimately made by the **Connection** filter below. You should enter a complete URL (for example, `http://host:8080/services`), from which the host and port is extracted and used to configure the **Static Router** below.	*Rewrite URL*
Static Router	The **Static Router** filter specifies the host of the destination server together with the port to connect to on that host.	*Static Router*
Connection	The **Connection** filter establishes the connection to the URL, host, and port specified in the **Rewrite URL** and **Static Router** filters. If an SSL connection is required, you can select a certificate from the Trusted Certificate Store to use to authenticate to the destination server. You can also select what certificates are considered trusted by the API Gateway so that the destination server's certificate can be trusted.	*Connection*

To use the **Routing Wizard** for a particular policy, right-click the policy under the **Policies** node in the Policy Studio tree, and select **Routing Wizard**. Configuring the **URL** field and/or the **Proxy Settings** tab in the **Routing Wizard** auto-generates a **Rewrite URL** filter and a **Static Router** filter.

Configuration

You can configure the following fields in the **Routing Wizard**:

URL:
Enter the *full* URL of the destination Web Service. The host, port, and scheme (HTTP or HTTPS) are extracted from the URL and used to configure a **Static Router** filter.

Proxy Host:
If you want to send the request using an HTTP proxy, configure the **Proxy Host** and **Proxy Port**. In this case, the **Static Router** filter is configured with the host and port entered in these fields. The **Connection** filter sends the complete URL specified in the **URL** field as the request URI to the proxy, as required by the HTTP specification. The proxy then knows where to route the message on to.

Note

If the **Proxy Host** and **Proxy Port** fields are completed, the wizard automatically selects the **Send via proxy** field on the **Advanced** tab of the auto-generated **Connection** filter.

Proxy Port:
If you want to route messages using a proxy to the destination Web Service, enter the port on the **Proxy Host** specified above on which the proxy accepts requests.

SSL Port:
If the proxy is SSL-enabled, enter the SSL port in this field.

Trusted Certificates, Client SSL Authentication, and HTTP Authentication Tabs
The settings configured on the remaining tabs of the wizard correspond to the settings configured on the tabs displayed on the **Connection** filter. For more information on configuring the fields on these tabs, see the *Connection* topic.

Call Internal Service

Overview

The **Call Internal Service** filter is a special filter that passes messages to an internal servlet application or static content provider that has been deployed at the API Gateway. The appropriate application is selected based on the relative path on which the request message is received.

This filter is used by Management Services that are configured to listen on the Management Interface on port 8090. For more information on how the **Call Internal Service** filter is used by these services, see the section called “Management Services” in the *Configuring HTTP Services* topic.

Configuration

You can configure the following fields on the filter screen:

Name:
Enter an appropriate name for this filter.

Additional HTTP Headers to Send to Internal Service:
You can click the **Add** button to configure additional HTTP headers to send to the internal application. Specify the following fields on the **HTTP Header** dialog:

- **HTTP Header Name**:
  Enter the name of the HTTP header to add to the message.
- **HTTP Header Value**:
  Enter the value of the new HTTP header. You can also enter selectors to represent message attributes. At runtime, the API Gateway expands these selectors to the current value of the corresponding message attribute. For example, the `${id}` selector is replaced by the value of the current message ID. Message attribute selectors have the following syntax:
  `${messsage_attribute}`

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Connection

Overview

The **Connection** filter makes the connection to the remote Web Service. It relies on connection details that are set by the other filters in the **Routing** category. Because the **Connection** filter connects out to other services, it negotiates the SSL handshake involved in setting up a mutually authenticated secure channel.

Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used. For an introduction to using the various filters in the **Routing** category, see the topic on *Getting Started with Routing Configuration*.

General Configuration

Enter an appropriate name for the filter in the **Name** field. Click the tabs to configure the various aspects of the **Connection** filter.

Trusted Certificates

This section lists all CA certificates that have been imported into the Oracle **Certificate Store**. Select the certificates that the API Gateway considers to be trusted when it attempts to establish a connection to a remote server. The remote server's SSL certificate must be issued by one of the selected trusted certificates on this tab.

Client SSL Authentication

You can configure the API Gateway to authenticate itself to the remote Web Service. It does so using the certificate selected on this tab.

HTTP Authentication

The API Gateway can use both HTTP basic and HTTP digest authentication to authenticate to the remote server. In both cases, the **User Name** and **Password** of a user must be specified in the fields provided.

Kerberos Authentication

The settings on this tab enable you to authenticate to a Kerberos Service by sending a Kerberos service ticket in the HTTP request to that service.

Note

You can also configure the API Gateway to authenticate to a Kerberos Service by including the relevant Kerberos tokens inside the XML message. For more details, see the *Kerberos Client Authentication* topic.

Kerberos Client:
The selected Kerberos Client has two roles. First, it must obtain a Kerberos TGT (Ticket Granting Ticket). Second, it uses this TGT to obtain a service ticket for the **Kerberos Service Principal** selected below.

Click the button on the right, and select a previously configured Kerberos Client in the tree. To add a Kerberos Client, right-click the **Kerberos Clients** tree node, and select **Add Kerberos Client**. Alternatively, you can add Kerberos Clients under the **External Connections** node in the Policy Studio tree view. For more details, see the *Kerberos Clients* topic.

Kerberos Service Principal:
The Kerberos Client selected above must obtain a ticket from the Kerberos Ticket Granting Server (TGS) for the selected Kerberos Service Principal. The Service Principal can be used to uniquely identify the Service in the Kerberos realm. The

TGS grants a ticket for the selected Principal, which the client can then send to the Kerberos Service. The Principal in the ticket must match the Kerberos Service's Principal for the client to be successfully authenticated.

Click the button on the right, and select a previously configured Kerberos Principal in the tree (for example, the default `HTTP/host Service Principal`). To add a Kerberos Principal, right-click the **Kerberos Principals** tree node, and select **Add Kerberos Principal**. Alternatively, you can add Kerberos Principals under the **External Connections** node in the Policy Studio tree view. For more details, see the topic on *Kerberos Principals*.

Send token with first request:
In some cases, the client may not authenticate (send the `Authorization` HTTP header) to the Kerberos Service on its first request. The Kerberos Service should then respond with an HTTP 401 response containing a `WWW-Authenticate: Negotiate` HTTP header. This header value instructs the client to authenticate to the server by sending up the `Authorization` header. The client then sends up a second request, this time with the `Authorization` header, which contains the relevant Kerberos token. Select this option to force the **Connection** filter to *always* send the `Authorization` HTTP header that contains the Kerberos Service ticket on its first request to the Kerberos Service.

Send body only after establish context:
You can configure the Kerberos client to only send the message body after the context has been fully established (the client has mutually authenticated with the service).

Pass when service returns 200 even if context not established:
In some rare cases, a Kerberos Service may return a `200 OK` response to a Kerberos Client's initial request even though the security context has not yet been fully established. This `200 OK` response may not contain the `WWW-authenticate` HTTP header.

By selecting this option, you are instructing the **Connection** filter to send the request to the Kerberos Service despite that the context has not been established. It is the responsibility of the Kerberos Service to decide whether to process the request depending on the status of the security context.

Behavior

The **Behavior** tab enables you to configure **Retries** and **Failure Settings**. By default, both of these sections are collapsed. Click a section to expand it.

Retries:
To specify the retry settings for this filter, complete the following fields:

Perform Retries	Select whether the filter performs retries. By default, this setting is not selected, no retries are performed, and all **Retries** settings are disabled. This means that the filter only attempts to perform the connection once.
Retry On	Select the HTTP status ranges on which retries can be performed. If a host responds with an HTTP status code that matches one of the selected ranges, this filter performs a retry. Select one or more ranges in the table (for example, `Client Error 400-499`). For details on adding custom HTTP status ranges, see the next subsection.
Retry Count	Enter the maximum number of retries to attempt. The default is `5`.
Retry Interval (ms)	Enter the amount of time to delay between retries in milliseconds. The default is `500` ms.

Adding HTTP Status Ranges
To add an HTTP status range to the default list displayed in the **Retry On** table, click the **Add** button. In the **Configure**

HTTP Status Code dialog, complete the following fields:

Name	Enter a name for the HTTP status range.
Start status	Enter the first HTTP status code in the range of status codes that you wish to monitor.
End status	Enter the last HTTP status code in the range of status codes that you wish to monitor.

If you wish to monitor one specific status code only, enter the same code in the **Start status** and **End status** fields. Click **OK** to finish. You can manage existing HTTP status ranges using the **Edit** and **Delete** buttons.

Failure Settings:
To specify the failure settings for this filter, complete the following fields:

Consider SLA Breach as Failure	Select whether to attempt the connection if a configured SLA has been breached. This is not selected by default. If this option is selected, and an SLA breach is encountered, the filter returns `false`.
Save Transaction on Failure (for replay)	Select whether to store the incoming message in the specified directory and file if a failure occurs during processing. This is not selected by default.
File name	Enter the name of the file that the message content is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${id}.out`. For more details on selectors, see *Selecting Configuration Values at Runtime*.
Directory	Enter the directory that the file is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${environment.VINSTDIR}/message-archive`, where `VINSTDIR` is the root of your API Gateway installation.
Maximum number of files in directory	Enter the maximum number of files that can be saved in the directory. Defaults to `500`.
Maximum file size	Enter the maximum file size in MB. Defaults to `1000`.
Include HTTP Headers	Select whether to include HTTP headers in the file. HTTP headers are not included by default.
Include Request Line	Select whether to include the HTTP request line from the client in the file. The request line is not included by default.
Call policy on Connection Failure	Select whether to execute a policy in the event of a connection failure. This is not selected by default.
Connection Failure Policy	Click the browse button on the right, and select the policy to run in the event of a connection failure in the dialog.

Advanced

This tab enables you to configure certain advanced features of the **Connection** filter. The following configuration options are available:

Handles Redirects:
Specifies whether the API Gateway handles HTTP redirects, and connects to the redirect URL specified in the HTTP response. This setting is enabled by default.

Forward spurious received Content headers:
Specifies whether the API Gateway sends any content-related message headers when sending an HTTP request with no message body to the HTTP server. For example, you can select this setting if content-related headers are required by an out-of-band agreement. If there is no body in the outbound request, any content-related headers from the original inbound HTTP request are forwarded. These are extracted from the `http.content.headers` message attribute, which is generally populated by the API Gateway for the incoming call. This attribute can be manipulated in a policy using the appropriate filters, if required. This setting is not enabled by default.

Send via Proxy:
Select this option if the API Gateway must connect to the destination Web Service through an HTTP proxy. In this case, the API Gateway includes the full URL of the destination Web Service in the request line of the HTTP request. For example, if the destination Web Service resides at `http://localhost:8080/services`, the request line is as follows:

```
POST http://localhost:8080/services HTTP/1.1
```

If the API Gateway is not routing through a proxy, the request line is as follows:

```
POST /services HTTP/1.1
```

Proxy Server:
When **Send via Proxy** is selected, you can configure a specific proxy server to use for the connection. Click the button next to this field, and select an existing proxy server in the dialog. To add a proxy server, right-click the **Proxy Servers** tree node, and select **Add a Proxy Server**. Alternatively, you can configure Proxy Servers under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Proxy Servers*.

Transparent Proxy (present client's IP address to server)
Enables the use of the API Gateway as a *transparent proxy* on Linux systems with the `TPROXY` kernel option set. When selected, the IP address of the original client connection that caused the policy to be invoked is used as the local address of the connection to the destination server. For more details, see *Configuring a Transparent Proxy*.

Ciphers:
The **Ciphers** field enables the administrator to specify the ciphers that the server supports. The server sends this list of supported ciphers to the destination server, which then selects the highest strength common cipher as part of the SSL handshake. The selected cipher is then used to encrypt the data as it is sent over the secure channel.

HTTP Host Header:
An HTTP 1.1 client *must* send a `Host` header in all HTTP 1.1 requests. The `Host` header identifies the host name and port number of the requested resource as specified in the original URL given by the client.

When routing messages on to target Web Services, the API Gateway can forward on the `Host` as received from the client, or it can specify the address and port number of the destination Web Service in the `Host` header that it routes onwards.

Select **Use Host header specified by client** to force the API Gateway to always forward on the original `Host` header that it received from the client. Alternatively, to configure the API Gateway to include the address and port number of the destination Web Service in the `Host` header, select the **Generate new Host header** radio button.

Connect to URL

Overview

The **Connect to URL** filter is the simplest routing filter to use to connect to a target Web Service. To configure this filter to send messages to a Web Service, you need only enter the URL of the service in the **URL** field. If the Web Service is SSL enabled or requires mutual authentication, you can use the other tabs on the **Connect to URL** filter to configure this.

Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used. Using the **Connect to URL** filter is equivalent to using the following combination of routing filters:

- Static Router
- Rewrite URL
- Connection

The **Connect to URL** filter enables the API Gateway to act as the endpoint to the client connection (and not as a proxy), and to hide the deployment hierarchy of protected Web Services from clients. In other words, the API Gateway performs *service virtualization*. For an introduction to routing scenarios and the filters in the **Routing** category, see the *Getting Started with Routing Configuration* topic.

General Configuration

Configure the following general settings on the **Basic** tab:

Name:
Enter an appropriate name for the filter.

URL:
Enter the complete URL of the target Web Service. You can specify this setting as a selector, which enables values to be expanded at runtime. For more details, see *Selecting Configuration Values at Runtime*. Defaults to `${http.request.uri}`.

Trusted Certificates

When API Gateway connects to a server over SSL, it must decide whether to trust the server's SSL certificate. You can select a list of CA or server certificates from the **Trusted Certificates** tab that are considered trusted by the API Gateway when connecting to the server specified in the **URL** field on this dialog.

The table displayed on the **Trusted Certificates** tab lists all certificates imported into the API Gateway Certificate Store. To *trust* a certificate for this particular connection, select the box next to the certificate in the table.

Client SSL Authentication

In cases where the destination server requires clients to authenticate to it using an SSL certificate, you must select a client certificate on the **Client SSL Authentication** tab. Select the checkbox next to the client certificate that you want to use to authenticate to the server specified in the **URL** field.

HTTP Authentication

If the destination server requires clients to authenticate to it using HTTP basic or digest authentication, use the fields on the **HTTP Authentication** tab.

None, HTTP Basic, or HTTP Digest:
Select the method that you want to use to authenticate to the server.

User Name:
Enter the user name you want to use to authenticate to the server.

Password:
Enter the password for this user.

Kerberos Authentication

The settings on this tab enable you to authenticate to a Kerberos Service by sending a Kerberos service ticket in the HTTP request to that service.

Note

You can also configure the API Gateway to authenticate to a Kerberos Service by including the relevant Kerberos tokens inside the XML message. For more details, see the *Kerberos Client Authentication* topic.

Kerberos Client:
The selected Kerberos Client has two roles. First, it must obtain a Kerberos TGT (Ticket Granting Ticket). Second, it uses this TGT to obtain a service ticket for the **Kerberos Service Principal** selected below.

You can configure Kerberos Clients globally under the **External Connections** node in the Policy Studio tree. These can then be selected in the **Kerberos Client** drop-down list. For more details on configuring Kerberos Clients, see the *Kerberos Clients* topic.

Kerberos Service Principal:
The Kerberos Client selected in the drop-down list must obtain a ticket from the Kerberos Ticket Granting Server (TGS) for the selected Kerberos Service Principal. The Service Principal can be used to uniquely identify the Service in the Kerberos realm. The TGS grants a ticket for the selected Principal, which the client can then send to the Kerberos Service. The Principal in the ticket must match the Kerberos Service's Principal for the client to be successfully authenticated.

You can also configure Kerberos Principals globally under the **External Connections** node in the Policy Studio tree. For more details on configuring Kerberos Principals, see the *Kerberos Principals* topic.

Send token with first request:
In some cases, the client may not authenticate (send the `Authorization` HTTP header) to the Kerberos Service on its first request. The Kerberos Service should then respond with an `HTTP 401` response containing a `WWW-Authenticate: Negotiate` HTTP header. This header value instructs the client to authenticate to the server by sending up the `Authorization` header. The client then sends up a second request, this time with the `Authorization` header, which contains the relevant Kerberos token. Select this option to force the **Connect to URL** filter to *always* send the `Authorization` HTTP header that contains the Kerberos Service ticket on its first request to the Kerberos Service.

Send body only after establish context:
You can configure the Kerberos client to only send the message body after the context has been fully established (the client has mutually authenticated with the service).

Pass when service returns 200 even if context not established:
In some rare cases, a Kerberos Service may return a `200 OK` response to a Kerberos Client's initial request even though the security context has not yet been fully established. This `200 OK` response may not contain the `WWW-authenticate` HTTP header.

By selecting this option, you are instructing the **Connect to URL** filter to send the request to the Kerberos Service even if the context has not been established. It is the responsibility of the Kerberos Service to decide whether to process the request depending on the status of the security context.

Behavior

The **Behavior** tab enables you to configure **Retries** and **Failure Settings**. By default, both of these sections are collapsed. Click a section to expand it.

Retries:
To specify the retry settings for this filter, complete the following fields:

Perform Retries	Select whether the filter performs retries. By default, this setting is not selected, no retries are performed, and all **Retries** settings are disabled. This means that the filter only attempts to perform the connection once.
Retry On	Select the HTTP status ranges on which retries can be performed. If a host responds with an HTTP status code that matches one of the selected ranges, this filter performs a retry. Select one or more ranges in the table (for example, `Client Error 400-499`). For details on adding custom HTTP status ranges, see the next subsection.
Retry Count	Enter the maximum number of retries to attempt. The default is `5`.
Retry Interval (ms)	Enter the amount of time to delay between retries in milliseconds. The default is `500` ms.

Adding HTTP Status Ranges
To add an HTTP status range to the default list displayed in the **Retry On** table, click the **Add** button. In the **Configure HTTP Status Code** dialog, complete the following fields:

Name	Enter a name for the HTTP status range.
Start status	Enter the first HTTP status code in the range of status codes that you wish to monitor.
End status	Enter the last HTTP status code in the range of status codes that you wish to monitor.

If you wish to monitor one specific status code only, enter the same code in the **Start status** and **End status** fields. Click **OK** to finish. You can manage existing HTTP status ranges using the **Edit** and **Delete** buttons.

Failure Settings:
To specify the failure settings for this filter, complete the following fields:

Consider SLA Breach as Failure	Select whether to attempt the connection if a configured SLA has been breached. This is not selected by default. If this option is selected, and an SLA breach is encountered, the filter returns `false`.
Save Transaction on Failure (for replay)	Select whether to store the incoming message in the specified directory and file if a failure occurs during processing. This is not selected by default.
File name	Enter the name of the file that the message content is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${id}.out`. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Directory	Enter the directory that the file is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${environment.VINSTDIR}/message-archive`, where `VINSTDIR` is the root of your API Gateway installation.
Maximum number of files in directory	Enter the maximum number of files that can be saved in the directory. Defaults to `500`.
Maximum file size	Enter the maximum file size in MB. Defaults to `1000`.
Include HTTP Headers	Select whether to include HTTP headers in the file. HTTP headers are not included by default.
Include Request Line	Select whether to include the HTTP request line from the client in the file. The request line is not included by default.
Call policy on Connection Failure	Select whether to execute a policy in the event of a connection failure. This is not selected by default.
Connection Failure Policy	Click the browse button on the right, and select the policy to run in the event of a connection failure in the dialog.

Advanced

This tab enables you to configure certain advanced features of the **Connect to URL** filter. The following configuration options are available:

Handles Redirects:
Specifies whether the API Gateway handles HTTP redirects, and connects to the redirect URL specified in the HTTP response. This setting is enabled by default.

Forward spurious received Content headers:
Specifies whether the API Gateway sends any content-related message headers when sending an HTTP request with no message body to the HTTP server. For example, you can select this setting if content-related headers are required by an out-of-band agreement. If there is no body in the outbound request, any content-related headers from the original inbound HTTP request are forwarded. These are extracted from the `http.content.headers` message attribute, which is generally populated by the API Gateway for the incoming call. This attribute can be manipulated in a policy using the appropriate filters, if required. This setting is not enabled by default.

Send via Proxy:
Select this option if the API Gateway must connect to the destination Web Service through an HTTP proxy. In this case, the API Gateway includes the full URL of the destination Web Service in the request line of the HTTP request. For example, if the destination Web Service resides at `http://localhost:8080/services`, the request line is as follows:

```
POST http://localhost:8080/services HTTP/1.1
```

If the API Gateway was not routing through a proxy, the request line is as follows:

```
POST /services HTTP/1.1
```

Proxy Server:
When **Send via Proxy** is selected, you can configure a specific proxy server to use for the connection. Click the button next to this field, and select an existing proxy server in the dialog. To add a proxy server, right-click the **Proxy Servers** tree node, and select **Add a Proxy Server**. Alternatively, you can configure Proxy Servers under the **External Connec-**

tions node in the Policy Studio tree. For more details, see the topic on *Proxy Servers*.

Transparent Proxy (present client's IP address to server)
Enables the use of the API Gateway as a *transparent proxy* on Linux systems with the `TPROXY` kernel option set. When selected, the IP address of the original client connection that caused the policy to be invoked is used as the local address of the connection to the destination server. For more details, see *Configuring a Transparent Proxy*.

Ciphers:
The **Ciphers** field enables the administrator to specify the ciphers that API Gateway supports. The API Gateway sends this list of supported ciphers to the destination server, which then selects the highest strength common cipher as part of the SSL handshake. The selected cipher is then used to encrypt the data as it is sent over the secure channel.

HTTP Host Header:
An HTTP 1.1 client ***must*** send a `Host` header in all HTTP 1.1 requests. The `Host` header identifies the hostname and port number of the requested resource as specified in the original URL given by the client.

When routing messages on to target Web Services, the API Gateway can forward on the `Host` as received from the client, or it can specify the address and port number of the destination Web Service in the `Host` header that it routes onwards.

Select **Use Host header specified by client** to force the API Gateway to always forward on the original `Host` header that it received from the client. Alternatively, to configure the API Gateway to include the address and port number of the destination Web Service in the `Host` header, select the **Generate new Host header** radio button.

Request Details

On the **Request Details** tab, you can use the API Gateway selector syntax to evaluate and expand request details at runtime. For more details, see *Selecting Configuration Values at Runtime*. You can configure the following settings:

Method:
Enter the HTTP verb used in the incoming request (for example, `GET`). Defaults to `${http.request.verb}`.

Body:
Enter the content of the incoming request message body. Defaults to `${content.body}`.

Important

You must enter the body headers and body content in the **Body** text area. For example, enter the `Content-Type` followed by a return and then the required message payload:

```
Content-Type: text/html

<!DOCTYPE html>
<html>
<body>
<h1>Hello World</h1>
</body>
</html>
```

Protocol Headers:
Enter the HTTP headers associated with the incoming request message. Defaults to `${http.headers}`.

Dynamic Router

Overview

The API Gateway can act as a proxy for clients of the secured Web Service. When a client uses a proxy, it includes the fully qualified URL of the destination in the request line of the HTTP request. It sends this request to the configured proxy, which then forwards the request to the host specified in the URL. The relative path used in the original request is preserved by the proxy on the outbound connection.

The following is an example of an HTTP request line that was made through a proxy, where `WEB_SERVICE_HOST` is the name or IP address of the machine hosting the destination Web Service:

```
POST http://WEB_SERVICE_HOST:80/myService HTTP/1.0
```

When the API Gateway acts as a proxy for clients, it can receive requests like the one above. The **Dynamic Router** filter can route the request on to the URL specified in the request line (`http://WEB_SERVICE_HOST:80/myService`).

Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used. For an introduction to using the various filters in the **Routing** category, see the topic on *Getting Started with Routing Configuration*.

Configuration

Enter an appropriate name for the router in the **Name** field on the **Dynamic Router** filter configuration screen.

Extract Path Parameters

Overview

The **Extract Path Parameters** filter enables the API Gateway to parse the contents of a specified HTTP path into message attributes. This means that you can define HTTP path parameters, and then extract their values at runtime using selectors. For example, this is useful when passing in parameters to REST-based requests. For more details on selectors, see the topic on *Selecting Configuration Values at Runtime*.

Configuration

Complete the following settings:

Name:
Enter a descriptive name for this filter.

URI Template:
Enter the URI template for the path to be parameterized. This is a formatted Jersey `@Path` annotation string, which enables you to parameterize the path specified in the incoming `http.request.path` message attribute. The following is an example URI template entry:

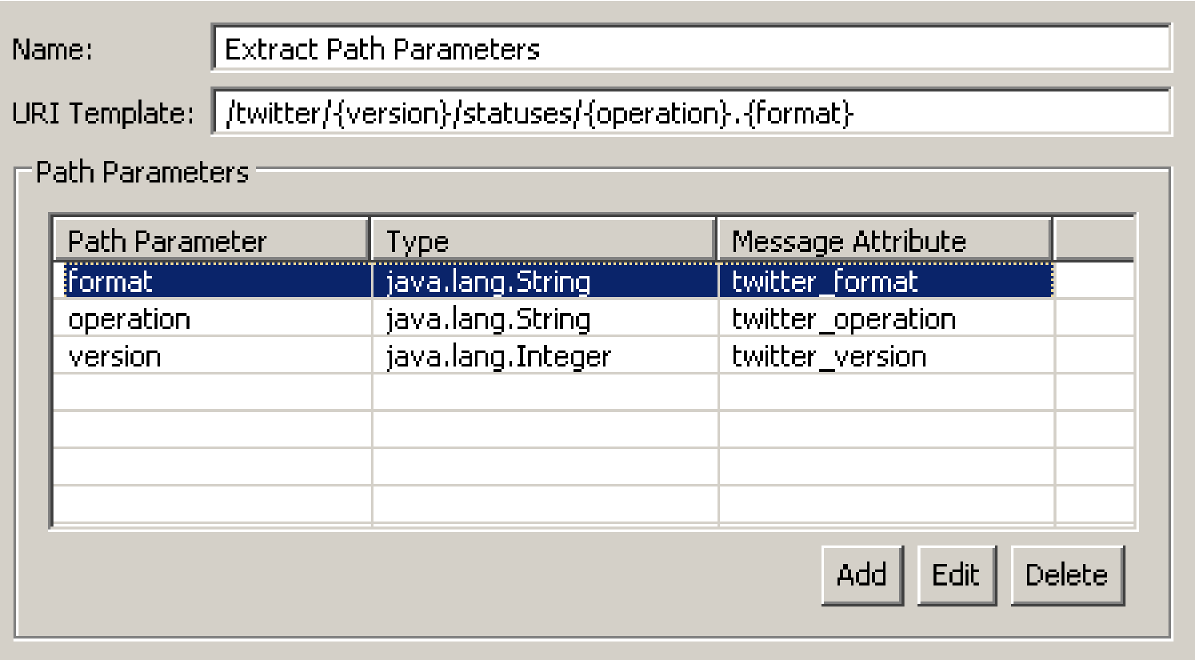
```
/twitter/{version}/statuses/{operation}.{format}
```

For more details on Jersey `@Path` annotations, see the following:
http://jersey.java.net/nonav/documentation/latest/jax-rs.html#d4e104

Path Parameters:
The **Path Parameters** table enables you to map the path parameters specified in the **URI Template** to user-defined message attributes. These attributes can then be used by other filters downstream in the policy. Click **Add** to configure a path parameter, and specify the following in the dialog:

Field	***Description***
Path Parameter	Enter the name of the path parameter (for example, `version`).
Type	Enter the type of the path parameter (for example, `java.lang.String`)..
Message Attribute	Enter the name of the message attribute that stores the parameter value (for example, `twitter_version`).

The following screen shows the example path parameters:



Name: Extract Path Parameters

URI Template: /twitter/{version}/statuses/{operation}.{format}

Path Parameters

Path Parameter	Type	Message Attribute
format	java.lang.String	twitter_format
operation	java.lang.String	twitter_operation
version	java.lang.Integer	twitter_version

Required Input and Generated Output

The incoming `http.request.path` message attribute is required as input to this filter.

This filter generates the message attributes for the parameters that you specify in the **Path Parameters** table. For example, in the previous screen shot, the following attributes are generated:

- `twitter_format`
- `twitter_operation`
- `twitter_version`

Possible Outcomes

The possible outcomes of this filter are as follows:

- `True` if the specified **URI Template** is successfully parsed.
- `False` if an error occurs during **URI Template** parsing.
- `CircuitAbortException` if an exception occurs during **URI Template** parsing.

File Download

Overview

You can use the **File Download** filter to download files from a file transfer server and store their contents in the `content.body` message attribute. The **File Download** filter supports the following protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

Configuring a **File Download** filter can be useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the API Gateway can download files from the other system. The added benefit is that the files are exposed to the full compliment of API Gateway message processing filters. This ensures that only properly validated files are downloaded from the target system. The **File Download** filter is available from the **Routing** category of filters in the Policy Studio.

General Settings

Configure the following general settings:

Name:
Enter a descriptive name for this filter.

Host:
Enter the name of the host machine on which the file transfer server is running.

Port:
Enter the port number to connect to the file transfer server. Defaults to `21`.

Username:
Enter the username to connect to the file transfer server.

Password:
Specify the password for this user.

File Details

Configure the following fields in the **File details** section:

Filename:
Specifies the filename to download from the file transfer server. The default value is `filename.xml`. You can enter a different filename or use a message attribute selector, which is expanded at runtime (for example, `${authentication.subject.id}`).

When downloading a file from the file transfer server, the API Gateway uses a temporary file name of `filename.part`. When the file has been downloaded, it then uses the filename specified in this filter (for example, the default `filename.xml`). This prevents an incomplete file from being downloaded.

Directory:
Specify the directory where the file is stored.

Connection Type

The fields configured in the **Connection Type** section determine the type of file transfer connection. Select the FTP connection type from the drop-down list:

- FTP - File Transfer Protocol
- FTPS - FTP over SSL
- SFTP - SSH File Transfer Protocol

FTP and FTPS Connections

The following general settings apply to FTP and FTPS connections:

Passive transfer mode:
Select this option to prevent problems caused by opening outgoing ports in the firewall relative to the file transfer server (for example, when using *active* FTP connections). This is selected by default.

File Type:
Select **ASCII** mode for sending text-based data, or **Binary** mode for sending binary data over the file transfer connection. Defaults to **ASCII** mode.

FTPS Connections

The following security settings apply to FTPS connections only:

SSL Protocol:
Enter the SSL protocol used (for example, `SSL` or `TLS`). Defaults to `SSL`.

Implicit:
When this option is selected, security is automatically enabled as soon as the **File Download** client makes a connection to the remote file transfer service. No clear text is passed between the client and server at any time. In this case, a specific port is used for secure connections (`990`). This option is not selected by default.

Explicit:
When this option is selected, the remote file transfer service must explicitly request security from the **File Download** client, and negotiate the required security. If the file transfer service does not request security, the client can allow the file transfer service to continue insecure or refuse and/or limit the connection. This option is selected by default.

Trusted Certificates:
To connect to a remote file server over SSL, you must trust that server's SSL certificate. When you have imported this certificate into the Certificate Store, you can select it on the **Trusted Certificates** tab.

Client Certificates:
If the remote file server requires the **File Download** client to present an SSL certificate to it during the SSL handshake for mutual authentication, you must select this certificate from the list on the **Client Certificates** tab. This certificate must have a private key associated with it that is also stored in the Certificate Store.

SFTP Connections

The following security settings apply to SFTP connections only:

Present following key for authentication:
Click the button on the right, and select a previously configured key to be used for authentication from the tree. To add a key, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Certificates and Keys** node in the Policy Studio tree. For more details, see the topic on *Certificates and Keys*.

SFTP host must present key with the following finger print:
Enter the fingerprint of the public key that the SFTP host must present (for example, `43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8`).

File Upload

Overview

You can use the **File Upload** filter to upload processed messages as files to a file transfer server. This enables you to upload the contents of the `content.body` message attribute as a file. The **File Upload** filter supports the following protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

Configuring a **File Upload** filter can be useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the API Gateway can make files available for upload to the other system. The added benefit is that the files are exposed to the full compliment of API Gateway message processing filters. This ensures that only properly validated files are uploaded to the target system.

The **File Upload** filter is available from the **Routing** category of filters in the Policy Studio. This topic describes how to configure the fields on the **File Upload** filter dialog.

General Settings

Configure the following general settings:

Name:
Enter a descriptive name for this filter.

Host:
Enter the name of the host machine on which the file transfer server is running.

Port:
Enter the port number to connect to the file transfer server. Defaults to `21`.

Username:
Enter the username to connect to the file transfer server.

Password:
Specify the password for this user.

File Details

Configure the following fields in the **File details** section:

Filename:
The message body (in the `content.body` message attribute) is stored using this filename on the destination file transfer server. The default value of `${id}.out` enables you to use the unique identifier associated with each message processed by the API Gateway. When this value is specified, messages are stored in individual files on the file transfer server according to their unique message identifier.

Directory:
Specify the directory where this file will be stored on the destination file transfer server.

Use temporary file name during upload:
This option specifies whether to use a temporary file name of `${id}.part` when the file is uploading to the file transfer server. When the file has uploaded, it then uses the filename specified in this filter (for example, the default `${id}.out`

filename). This prevents an incomplete file from being uploaded. This option is selected by default.

Important

You must deselect this option if the file transfer server is the API Gateway. For example, this option applies when the API Gateway uploads to a file transfer server, and then another server (possibly API Gateway) polls the file transfer server for new files to process. The poller server is configured to consume `*.xml` files and ignores the temporary file. When the upload is complete, the file is renamed and the poller sees the new file to process.

Connection Type

The fields configured in the **Connection Type** section determine the type of file transfer connection. Select the FTP connection type from the drop-down list:

- FTP - File Transfer Protocol
- FTPS - FTP over SSL
- SFTP - SSH File Transfer Protocol

FTP and FTPS Connections

The following general settings apply to FTP and FTPS connections:

Passive transfer mode:
Select this option to prevent problems caused by opening outgoing ports in the firewall relative to the file transfer server (for example, when using *active* FTP connections). This is selected by default.

File Type:
Select **ASCII** mode for sending text-based data, or **Binary** mode for sending binary data over the file transfer connection. Defaults to **ASCII** mode.

FTPS Connections

The following security settings apply to FTPS connections only:

SSL Protocol:
Enter the SSL protocol used (for example, `SSL` or `TLS`). Defaults to `SSL`.

Implicit:
When this option is selected, security is automatically enabled as soon as the **File Upload** client makes a connection to the remote file transfer service. No clear text is passed between the client and server at any time. In this case, a specific port is used for secure connections (`990`). This option is not selected by default.

Explicit:
When this option is selected, the remote file transfer service must explicitly request security from the **File Upload** client, and negotiate the required security. If the file transfer service does not request security, the client can allow the file transfer service to continue insecure or refuse and/or limit the connection. This option is selected by default.

Trusted Certificates:
To connect to a remote file server over SSL, you must trust that server's SSL certificate. When you have imported this certificate into the Certificate Store, you can select it on the **Trusted Certificates** tab.

Client Certificates:
If the remote file server requires the **File Upload** client to present an SSL certificate to it during the SSL handshake for mutual authentication, you must select this certificate from the list on the **Client Certificates** tab. This certificate must have a private key associated with it that is also stored in the Certificate Store.

SFTP Connections

The following security settings apply to SFTP connections only:

Present following key for authentication:
Click the button on the right, and select a previously configured key to be used for authentication from the tree. To add a key, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Certificates and Keys** node in the Policy Studio tree. For more details, see the topic on *Certificates and Keys*.

SFTP host must present key with the following finger print:
Enter the fingerprint of the public key that the SFTP host must present (for example, `43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8`).

HTTP Redirect

Overview

You can use the **HTTP Redirect** filter to enable the API Gateway to send an HTTP redirect message. For example, you may wish to send an HTTP redirect to force a client to enter user credentials on an HTML login page if no HTTP cookie already exists. Alternatively, you may wish to send an HTTP redirect if a Web page has moved to a new URL address.

Configuration

Complete the following settings:

Name:
Enter a descriptive name for this filter.

HTTP response code status:
Enter the HTTP response code status to use in the HTTP redirect message. Defaults to `301`, which means that the requested resource has been assigned a new permanent URI, and any future references to this resource should use the returned redirect URL.

Redirect URL:
Enter the URL address to which the message is redirected.

Content-Type:
Enter the `Content-Type` of the HTTP redirect message (for example, `text/xml`).

Message Body:
Enter the message body text that you wish to send in the HTTP redirect message.

HTTP Status Code

Overview

This filter sets the HTTP status code on response messages. This enables an administrator to ensure that a more meaningful response is sent to the client in the case of an error or anomaly occurring in a configured policy.

For example, if a **Relative Path** filter fails, it may be useful to return a `503 Service Unavailable` response. Similarly, if a user does not present identity credentials when attempting to access a protected resource, you can configure the API Gateway to return a `401 Unauthorized` response to the client.

HTTP status codes are returned in the *status-line* of an HTTP response. The following are some typical examples:

```
HTTP/1.1 200 OK
HTTP/1.1 400 Bad Request
HTTP/1.1 500 Internal Server Error
```

Configuration

Name:
Enter an appropriate name for this filter.

HTTP response code status:
Enter the status code returned to the client. For a complete list of status codes, see the HTTP Specification [http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html].

Insert WS-Addressing

Overview

The WS-Addressing specification defines a transport-independent standard for including addressing information in SOAP messages. The API Gateway can generate WS-Addressing information based on a configured endpoint in a policy, and then insert this information into SOAP messages.

Configuration

Complete the following fields to configure the API Gateway to insert WS-Addressing information into the SOAP message header.

Name:
Enter an appropriate name for the filter.

To:
The message is delivered to the specified destination.

From:
Informs the destination server where the message originated from.

Reply To:
Indicates to the destination server where it should send response messages to.

Fault To:
Indicates to the destination server where it should send fault messages to.

MessageID:
A unique identifier to distinguish this message from others at the destination server. It also provides a mechanism for correlating a specific request with its corresponding response message.

Action:
The specified action indicates what action the destination server should take on the message. Typically, the value of the WS-Addressing `Action` element corresponds to the SOAPAction on the request message. For this reason, this field defaults to the `soap.request.action` message attribute.

Relates To:
If responses are to be received asynchronously, the specified value provides a method to associate an incoming reply to its corresponding request.

Namespace:
The WS-Addressing namespace to use in the WS-Addressing block.

Messaging System Filter

Overview

A *messaging system* is a loosely coupled, peer-to-peer facility where clients can send messages to, and receive messages from any other client. In a messaging system, a client sends a message to a messaging agent. The recipient of the message can then connect to the same agent and read the message. However, the sender and recipient of the message do not need to be available at the same time to communicate (for example, unlike HTTP). The sender and recipient need only know the name and address of the messaging agent to talk to.

The Java Messaging System (JMS) is an implementation of such a messaging system. It provides an API for creating, sending, receiving, and reading messages. Java-based applications can use it to connect to other messaging system implementations. A *JMS provider* can deliver messages synchronously or asynchronously, which means that the client can fire and forget messages or wait for a response before resuming processing. Furthermore, the JMS API ensures different levels of reliability in terms of message delivery. For example, it can ensure that the message is delivered once and only once, or at least once.

The API Gateway uses the JMS API to connect to other messaging systems that expose a JMS interface. For example, these include Oracle WebLogic Server, IBM MQSeries, JBoss Messaging, TIBCO EMS, IBM WebSphere Server, and Progress SonicMQ.

Important

You must add the JMS provider JAR files to the API Gateway classpath for this filter to function correctly. If the provider's implementation is platform-specific, copy the provider JAR files to the `INSTALL_DIR/ext/PLATFORM` folder, where `INSTALL_DIR` points to the root of your product installation, and `PLATFORM` is the platform on which the API Gateway is installed (`Win32`, `Linux.i386`, or `SunOS.sun4u-32`). If the provider implementation is platform-independent, you can place the JAR files in `INSTALL_DIR/ext/lib`.

Request Settings

The **Request** tab specifies properties of the request to the messaging system. You can configure the following fields:

JMS Service:
Click the button next to this field, and select an existing JMS service in the tree. To add a JMS Service, right-click the **JMS Services** tree node, and select **Add a JMS Service**. Alternatively, you can configure JMS services under the **External Connections** node in the Policy Studio tree. For more details, see the *Messaging System* topic.

Destination:
Enter the name of the JMS queue or topic that you want to drop messages on to.

Delivery Mode:
The API Gateway supports persistent and non-persistent delivery modes:

- **Persistent:**
  Instructs the JMS provider to ensure that a message is not lost in transit if the JMS provider fails. A message sent with this delivery mode is logged to persistent storage when it is sent.
- **Non-persistent:**
  Does not require the JMS provider to store the message. With this mode, the message may be lost if the JMS provider fails.

Priority Level:
You can use message priority levels to instruct the JMS provider to deliver urgent messages first. The ten levels of prior-

ity range from 0 (lowest) to 9 (highest). If you do not specify a priority level, the default level is 4. A JMS provider tries to deliver higher priority messages before lower priority ones but does not have to deliver messages in exact order of priority.

Time to Live:
By default, a message never expires. However, if a message becomes obsolete after a certain period, you may want to set an expiration time (in milliseconds). If the specified time to live value is 0, the message never expires.

Message ID:
Enter an identifier to be used as the unique identifier for the message. By default, the unique identifier is the ID assigned to the message by the API Gateway (`${id}`). However, you can use a proprietary correlation system, perhaps using MIME message IDs instead of API Gateway message IDs.

Correlation ID:
Enter an identifier for the message that the API Gateway uses to correlate response messages with the corresponding request messages. Usually, if `${id}` is specified in the **Message ID** field above, it is also used here to correlate request messages with their correct response messages.

Message Type:
This drop-down list enables you to specify the type of data to be serialized and sent in the JMS message to the JMS provider. The option selected depends on what part of the message you want to send to the consumer. For example, if you want to send the message body, select the option to format the body according to the rules defined in the SOAP over JMS [http://www.w3.org/TR/soapjms/] recommendation. Alternatively, if you wanted to serialize a list of name-value pairs to the JMS message, choose the option to create a `MapMessage`.

The following list describes the various serialization options available:

- `Use content.body attribute to create a message in the format specified in the SOAP over Java Messaging Service recommendation`:
  If this option is selected, messages are formatted according to the SOAP over JMS [http://www.w3.org/TR/soapjms/] recommendation. This is the default option because, in most cases, the message body is to be routed to the messaging system. If this option is selected, a `javax.jms.BytesMessage` is created and a JMS property containing the content type (`text/xml`) is set on the message.
- `Create a MapMessage from the java.lang.Map in the attribute named below`:
  Select this option to create a `javax.jms.MapMessage` from the API Gateway message attribute named below that consists of name-value pairs.
- `Create a BytesMessage from the attribute named below`:
  Select this option to create a `javax.jms.BytesMessage` from the API Gateway message attribute named below.
- `Create an ObjectMessage from the java.lang.Serializable in the attribute named below`:
  Select this option to create a `javax.jms.ObjectMessage` from the API Gateway message attribute named below.
- `Create a TextMessage from the attribute named below`:
  Select this option to create a `javax.jms.TextMessage` from the message attribute named below.
- `Use the javax.jms.Message stored in the attribute named below`:
  If a `javax.jms.Message` has already been stored in a message attribute, select this option, and enter the name of the attribute in the field below.

Attribute Name:
Enter the name of the API Gateway message attribute that holds the data that is to be serialized to a JMS message and sent over the wire to the JMS provider. The type of the attribute named here must correspond to that selected in the **Message Type** drop-down field above.

Use Shared JMS Session:
By default, each running instance of a **Messaging System** filter creates its own session (using its own thread) with the JMS provider. You can select this option to force all running instances of this filter to share the same JMS session (using a common shared thread) to the JMS provider. Reusing a shared session across multiple filter instances in this manner may result in performance degradation as each connection to the provider using the session blocks until the response (if any) is received.

Custom Message Properties:
You can set custom properties for messages in addition to those provided by the header fields. Custom properties may be required to provide compatibility with other messaging systems. You can use message attribute selectors as property values. For example, you can create a property called `AuthNUser`, and set its value to `${authenticated.subject.id}`. Other applications can then filter on this property (for example, only consume messages where `AuthNUser` equals `admin`). To add a new property, click **Add**, and enter a name and value in the fields provided on the **Properties** dialog.

Use the following policy to change JMS request message:
This setting enables you to customize the JMS message before it is published to a JMS queue or topic. Click the browse button on the right, and select a configured policy in the dialog. The selected policy is then invoked before the JMS request is sent to the queuing system.

When the selected policy is invoked, the JMS request message is available on the white board in the `jms.outbound.message` message attribute. You can therefore call JMS API methods to manipulate the JMS request further. For example, you could configure a policy containing a **Scripting Language** filter that runs a script such as the following against the JMS message:

```
function invoke(msg) {
 var jmsMsg = msg.get("jms.outbound.message");
 jmsMsg.setIntProperty("My_JMS_Report", 123);
 return true;
}
```

Response Settings

The **Response** tab specifies whether the API Gateway uses asynchronous or synchronous communication when talking to the messaging system. For example, if you want the API Gateway to use asynchronous communication, you can select the **Do not set response** option. If synchronous communication is required, you can select to read the response from a temporary queue or from a named queue or topic.

You can also specify whether the API Gateway waits on a response message from a queue or topic from the messaging system. The API Gateway sets the `JMSReplyTo` property on each message that it sends. The value of the `JMSReplyTo` property is the temporary queue, queue, or topic selected in this dialog. It is the responsibility of the application that consumes the message from the queue (JMS Consumer) to send the message back to the destination specified in `JMSReplyTo`.

The API Gateway sets the `JMSCorrelationID` property to the value of the **Correlation ID** field on the **Request** tab to correlate requests messages to their corresponding response messages. If you select to use a temporary queue or temporary topic, this is created when the API Gateway starts up.

Wait for response:
Select whether the API Gateway waits to receive a response:

- **No wait:**
  The API Gateway does not wait to receive a response.
- **Wait with timeout (ms):**
  The API Gateway waits a specific time period to receive a response before it times out. If the API Gateway times out waiting for a response, the **Messaging System** filter fails. Enter the timeout value in milliseconds. The default value of `0` means there is no timeout, and the API Gateway waits for a response indefinitely.

Set where response is to be sent:
Select where the response message is to be placed using one of the following options:

- **Do not set response:**
  Select this option if you do not expect or do not care about receiving a response from the JMS provider.

- **Use temporary queue:**
  Select this option to instruct the JMS provider to place the response message on a temporary queue. In this case, the temporary queue is created when the API Gateway starts up. Only the API Gateway can read from the temporary queue, but any application can write to it. The API Gateway uses the value of the `JMSReplyTo` header to indicate the location where reads responses from.
- **Use named queue or topic:**
  If you want the JMS provider to place response messages on a named queue or topic, select this option, and enter the name of the queue or topic in the text box.

Selector for response:
The expression entered specifies the messages that the consumer is interested in receiving. By using a selector, the task of filtering the messages is performed by the JMS provider instead of by the consumer. The selector is a string that specifies an expression whose syntax is based on the SQL92 conditional expression syntax. The API Gateway instance only receives messages whose headers and properties match the selector.

Important

The JMS Consumer automatically returns the results of the invoked policy to the JMS destination specified in the `JMSReplyTo` header of the request. This means that you do not need to send a reply using the **Messaging System** filter.

If the incoming JMS message contains a `JMSReplyTo` header (a queue or topic that expects a response), when the policy invoked by the JMS Consumer completes, the API Gateway sends a message to the `JMSReplyTo` source using the reverse of the mechanism that it used to read from the queue. For example, if the consumer reads the JMS message and populates an attribute with a value inferred from the message type to Java (for example, from `TextMessage` to `String`), when the policy completes, the consumer looks up this attribute, and infers the JMS response message type based on the object type stored in the message.

Read WS-Addressing

Overview

The WS-Addressing specification defines a transport-independent standard for including addressing information in SOAP messages. The API Gateway can read WS-Addressing information contained in a SOAP message and subsequently use this information to route the message to its intended destination.

Configuration

Complete the following fields to configure the API Gateway to read WS-Addressing information contained in a SOAP message.

Name:
Enter an appropriate name for the filter.

Address location:
Specify the name of the element in the WS-Addressing block that contains the address of the destination server to which the API Gateway routes the message. For more information on configuring XPath expressions, see the *Configuring XPath Expressions* topic.

By default, XPath expressions are available to extract the destination server from the `From`, `To`, `ReplyTo`, and `FaultTo` elements. Click the **Add** button to add a new XPath expression to extract the address from a different location.

Remove enclosing WS-Addressing element:
If this option is selected, the WS-Addressing element returned by the XPath expression configured above is removed from the SOAP Header when it has been consumed.

Rewrite URL

Overview

You can use the **Rewrite URL** filter to specify the path on the remote machine to send the request to. This filter normally used in conjunction with a **Static Router** filter, whose role is to supply the host and port of the remote service. For more details, see the *Static Router* topic.

Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used. For an introduction to using the various filters in the **Routing** category, see the topic on *Getting Started with Routing Configuration*.

Configuration

Configure the following fields on the **Rewrite URL** filter configuration screen:

Name:
Enter an appropriate name for the filter in the **Name** field.

URL:
Enter the relative path of the Web Service in the **URL** field. The API Gateway combine the specified path with the host and port number specified in the **Static Router** filter to build up the complete URL to route to.

Alternatively, you can perform simple URL rewrites by specifying a fully qualified URL into the **URL** field. You can then use a **Dynamic Router** to route the message to the specified URL.

Save to File

Overview

The **Save to File** filter enables you to write the current message contents to a file. For example, you can save the message contents to a file in a directory where it can be accessed by an external application. This can be used to quarantine messages to the file system for offline examination. This filter can also be useful when integrating legacy systems. Instead of making drastic changes to the legacy system by adding an HTTP engine, the API Gateway can save the message contents to the file system, and route them on over HTTP to another back-end system.

Configuration

To configure the **Save to File** filter, specify the following fields:

Name	Name of the filter to be displayed in a policy. Defaults to **Save to File**.
File name	Enter the name of the file that the content is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${id}.out`. For more details on selectors, see *Selecting Configuration Values at Runtime*.
Directory	Enter the directory that the file is saved to. You can specify this using a selector, which is expanded to the specified value at runtime. Defaults to `${environment.VINSTDIR}/message-archive`, where `VINSTDIR` is the root of your API Gateway installation.
Maximum number of files in directory	Enter the maximum number of files that can be saved in the directory. Defaults to `500`.
Maximum file size	Enter the maximum file size in MB. Defaults to `1000`.
Include HTTP Headers	Select whether to include HTTP headers in the file. HTTP headers are not included by default.

SMTP Routing

Overview

You can use the **SMTP Routing** filter to relay messages to an email recipient using a configured SMTP server.

General Settings

Complete the following general settings:

Name:
Specify a descriptive name for this SMTP Server.

SMTP Server Settings:
Click the button on the right, and select a pre-configured SMTP Server in the tree. To add an SMTP Server, right-click the **SMTP Servers** tree node, and select **Add an SMTP Server**. Alternatively, you can configure SMTP Servers under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *SMTP Servers*.

Message Settings

Complete the following fields in the **Message Settings** section of the screen:

To:
Enter the email address of the recipient(s) of the messages. You can enter multiple addresses by separating each one using a semicolon. For example:

```
joe.soap@example.com;joe.bloggs@example.com;john.doe@example.com
```

Subject:
Enter some text as the **Subject** of the email messages.

Send content as attachment:
Select whether to send the message content as an attachment. This setting is not selected by default. In Oracle API Gateway 6.2 and earlier versions, the **SMTP Routing** filter always sent the message content as an attachment.

Static Router

Overview

The API Gateway uses the information configured in the **Static Router** filter to connect to a machine that is hosting a Web Service. You should use the **Static Router** filter in conjunction with a **Rewrite URL** filter to specify the path to send the message to on the remote machine. For more details, see the *Rewrite URL* topic.

Depending on how the API Gateway is perceived by the client, different combinations of routing filters can be used. For an introduction to using the various filters in the **Routing** category, see the topic on *Getting Started with Routing Configuration*.

Configuration

You must configure the following fields must be configured on the **Static Router** configuration screen:

Name:
Enter a name for the filter.

Host:
Enter the host name or IP address of the remote machine that is hosting the destination Web Service.

Port:
Enter the port on which the remote service is listening.

HTTP:
Select this option if the API Gateway should send the message to the remote machine over plain HTTP.

HTTPS:
Select this option if the API Gateway should send the message to the remote machine over a secure channel using SSL. You can use a **Connection** filter to configure the API Gateway to mutually authenticate to the remote system.

TIBCO Rendezvous Routing

Overview

TIBCO Rendezvous® is a low latency messaging product for real-time high throughput data distribution applications. It facilitates the exchange of data between applications over the network.

A TIBCO Rendezvous *daemon* runs on each participating node on the network. All data sent to and read by each application passes through the daemon. API Gateway uses the TIBCO Rendezvous API to communicate with a TIBCO Rendezvous daemon running locally (by default) to send messages to other TIBCO Rendezvous programs.

You can configure the **TIBCO Rendezvous** filter to route messages (using a TIBCO Rendezvous daemon) to other TIBCO Rendezvous programs. This filter is found in the Routing category of filters.

Configuration

Configure the following fields to route messages to other TIBCO Rendezvous programs:

Name:
Enter an appropriate name for this filter in the field provided.

TIBCO Rendezvous Daemon to Use:
Click the button on the right, and select a previously configured TIBCO Rendezvous Daemon from the tree. The API Gateway sends messages to the specified TIBCO **Rendezvous Subject** on this daemon. To add a TIBCO Rendezvous Daemon, right-click the **TIBCO Rendezvous Daemons** tree node, and select **Add a TIBCO Rendezvous Daemon**. For more details, see the *TIBCO Rendezvous Daemon* topic.

TIBCO Rendezvous Subject:
The message is sent with the subject entered here meaning that all other TIBCO daemons on the network that have subscribed to this subject name will receive the message. The subject name comprises a series of elements, including wildcards (for example, `*`), separated by dot characters, for example:

- `news.sport.soccer`
- `news.sport.*`
- `FINANCE.ACCOUNT.SALES`

For more information on the subject name syntax, see the TIBCO Rendezvous documentation.

Field Name:
Click the **Add** button to add details about a particular field to add to the message. On the **Message Field Definition** dialog, enter the name of the field to send in the message in the **Field Name** field, and complete the remaining fields.

Type:
Select the data type of the value specified in either of the following fields:

Set value to the following constant value:
You can explicitly set this value by entering it here.

Set value to the object found in the following attribute:
If you would like to dynamically populate the field value using the contents of a message attribute, you can select this attribute from this drop-down list. At runtime, the contents of the message attribute are placed into the message that is sent to TIBCO Rendezvous.

TIBCO Enterprise Messaging Service Routing Filter

Overview

TIBCO Enterprise Messaging Service™ (EMS) provides a distributed message bus with native support for Java Messaging Service (JMS) and TIBCO Rendezvous, along with other protocols.

In general, TIBCO EMS clients *produce* messages and send them to the TIBCO EMS Server. Similarly, TIBCO EMS clients can connect to the TIBCO EMS Server and declare an interest in a particular queue or topic on that server. In doing so, it can *consume* messages that have been produced by another TIBCO EMS client.

The API Gateway can act as a message producer by sending messages to the TIBCO EMS Server and as a message consumer by listening on a queue or topic at the server. The TIBCO EMS Routing filter can be used as a message producer in this manner.

Connection

On the **Connection** tab, click the button on the right, and select a previously configured TIBCO EMS Connection in the tree. Messages are sent to this TIBCO EMS connection and are dropped on the queue or topic specified on the **Request** tab.

To add a TIBCO EMS Connection, right-click the **TIBCO Enterprise Messaging Service Connections** node, and select **Add a TIBCO EMS connection**. For more information on configuring TIBCO EMS connections, see the *TIBCO Enterprise Messaging Service Connection* topic.

Request

The **Request** tab is used to configure properties of the request to the messaging system. You can configure the following fields:

Destination Type:
You must specify whether the name specified in the **Queue or Topic Name** field below is a `Queue` or `Topic`.

Queue or Topic Name:
Enter the name of the queue or topic that you want to drop messages on to.

Delivery Mode:
The API Gateway supports the following delivery modes:

- **Persistent:**
  Instructs the TIBCO EMS Server to ensure that a message is not lost in transit if the server fails. A message sent with this delivery mode is logged to persistent storage when it is sent.
- **Non-persistent:**
  Does not require the TIBCO EMS Server to store the message. With this mode, the message may be lost if the server fails.
- **Reliable:**
  When using reliable mode the TIBCO EMS Server never sends an acknowledgment or confirmation receipt back to the producer. This greatly decreases the volume of traffic on the network and can result in improved performance.

Priority Level:
You can use message priority levels to instruct the TIBCO EMS server to deliver urgent messages first. The ten levels of priority range from 0 (lowest) to 9 (highest). If you do not specify a priority level, the default level is 1. The TIBCO EMS Server tries to deliver higher priority messages before lower priority ones but does not have to deliver messages in exact order of priority.

Time to Live:
By default, a message never expires. However, if a message becomes obsolete after a certain period, you may want to set an expiration time (in milliseconds). If the specified time to live value is 0, the message never expires.

Message ID:
Enter an identifier to be used as the unique identifier for the message. By default, the unique identifier is the ID assigned to the message by the API Gateway (`${id}`). However, you can use a proprietary correlation system, perhaps using MIME message IDs instead of Oracle message IDs.

Correlation ID:
Enter an identifier for the message that the API Gateway uses to correlate response messages with the corresponding request messages. Usually, if `${id}` is specified in the **Message ID** field above, it is also used here to correlate request messages with their correct response messages.

Message Type:
This enables you to specify the type of data to be serialized and sent in the message to the TIBCO EMS Server. The option selected depends on what part of the message you want to send to the consumer.

For example, if you wish to send the message body you should select the option to format the body according to the rules defined in the SOAP over JMS [http://www.w3.org/TR/soapjms/] recommendation. Alternatively, if you wish to serialize a list of name-value pairs to the message, choose the option to create a `MapMessage`.

The following list describes the various serialization options available:

- `Use content.body attribute to create a message in the format specified in the SOAP over Java Messaging Service recommendation:`
  If this option is selected, messages are formatted according to the SOAP over JMS [http://www.w3.org/TR/soapjms/] recommendation. This is the default option.
- `Create a MapMessage from the java.lang.Map in the attribute named below:`
  Select this option to create a `javax.jms.MapMessage` from the Oracle message attribute (named below) that consists of name-value pairs.
- `Create a ByteMessage from the attribute named below:`
  Select this option to create a `javax.jms.ByteMessage` from the Oracle message attribute named below.
- `Create an ObjectMessage from the java.lang.Serializable in the attribute named below:`
  Select this option to create a `javax.jms.ObjectMessage` from the Oracle message attribute named below.
- `Create a TextMessage from the attribute named below:`
  A `javax.jms.TextMessage` can be created from the message attribute named below by selecting this option.

Attribute Name:
Enter the name of the Oracle message attribute that holds the data to be serialized to a JMS message and sent over the wire to the TIBCO EMS Server. The type of the attribute named here must correspond to that selected in the **Message Type** field above.

Custom Message Properties:
You can set custom properties for messages in addition to the standard JMS header fields. Custom properties may be useful to pass additional information to the TIBCO EMS Server. You can use message attribute selectors as property values. For example, you can create a property called `AuthNUser`, and set its value to `${authenticated.subject.id}`. When this message is routed to the specified queue or topic, other consumers can then filter on this property (for example, only consume messages where `AuthNUser` equals `admin`). For more details on selectors, see *Selecting Configuration Values at Runtime*.

Response

The **Response** tab is used to configure whether the API Gateway should use asynchronous or synchronous communication when talking to the messaging system. If the API Gateway is to use asynchronous communication, select `No Response` from the **Response** drop-down list. If synchronous communication is required, you must configure where to read the response from the TIBCO EMS Server.

When synchronous communication is selected, the API Gateway waits on a message from a queue/topic from the TIBCO EMS Server. The API Gateway sets the `JMSReplyTo` property on each message that it sends. The value of the `JMSReplyTo` property is the queue, temporary queue, topic, or temporary topic that was selected in the **Response** drop-down list. It is the responsibility of the application that consumes the message from the queue to send the message back to the destination specified in the `JMSReplyTo` property.

The API Gateway sets the `JMSCorrelationID` property to the value of the **Correlation ID** field on the **Response** tab to correlate requests messages to their corresponding response messages. If the user has selected to use a temporary queue or temporary topic, this is created when the API Gateway starts up.

Response:
Select where to read the response message from. The following options are available:

- **No Response:**
  Select this option if you do not expect or require a response from the TIBCO EMS Server.
- **Response in Queue Named Below:**
  If you want the TIBCO EMS Server to place response messages into a named queue, select this option, and enter the name of the queue in the field below.
- **Response in Temporary Queue:**
  You can also instruct the TIBCO EMS Server to place response messages on a temporary queue from which the API Gateway can pick them up.
- **Response in Topic Named Below:**
  Select this option to tell the TIBCO EMS Server to place response messages in the topic named in the field below.
- **Response in Temporary Topic:**
  If you want to read response messages from a temporary topic, select this option.

Note

When a temporary destination is selected, this destination is created at start-up of the API Gateway. Only the API Gateway can read from the temporary destination, however, any application can write to it. The API Gateway uses the value of the `JMSReplyTo` header to indicate the location where it reads responses from.

Reply Topic/Queue Name:
If you have selected a named queue or topic (*not* a temporary queue or topic) from the **Response** field above, enter its name here.

Time Out:
The API Gateway waits a certain time period for a response to be received before times out. If the API Gateway does time out waiting for a response, this filter fails. Enter the time out value in milliseconds.

TIBCO Enterprise Messaging Service Connection

Overview

TIBCO Enterprise Messaging Service™ (EMS) provides a distributed message bus with support for JMS (Java Messaging Service) and TIBCO Rendezvous, along with other protocols.

In general, TIBCO EMS clients *produce* messages and send them to the TIBCO EMS Server. Similarly, TIBCO EMS clients can connect to the TIBCO EMS Server and declare an interest in a particular queue or topic on that server. In doing so, it can *consume* messages that have been produced by another TIBCO EMS client.

The API Gateway can act as a message producer by sending messages to the TIBCO EMS Server and as a message consumer by listening on a queue or topic at the server. Both configurations require a connection to the TIBCO EMS Server.

For more information on consuming and producing messages to and from TIBCO EMS, see the following topics:

- *TIBCO Integration*
- *TIBCO Enterprise Messaging Service Consumer*
- *TIBCO Enterprise Messaging Service Routing Filter*

This topic describes how to configure a connection to an TIBCO EMS Server. For more detailed information on configuring TIBCO EMS Connections, see the TIBCO EMS documentation.

Configuration

The TIBCO EMS Connection is configured globally so that it can be referenced when configuring TIBCO EMS consumers and TIBCO EMS producers in the API Gateway. To configure a global connection to an TIBCO EMS Server, right-click the **External Connections** -> **TIBCO Enterprise Messaging Service Connections** node in the Policy Studio tree, and select **Add a TIBCO EMS Connection** from the context menu. The remainder of this topic describes how to configure the tabs and fields on the **TIBCO Enterprise Messaging System Connection** dialog.

Before configuring the following fields you must enter a name for this TIBCO EMS Connection in the **Name** field. This connection is then available when configuring a TIBCO EMS Consumer and when configuring a TIBCO EMS Routing filter.

General Tab:
The following fields are available on the **General Tab**:

Server URL:
Enter the full URL of the TIBCO EMS Server in this field, for example `tcp://hostname:7222` for non-SSL connections or `ssl://server:7243` for SSL-enabled TIBCO EMS Servers.

User Name:
Enter a username to use when the API Gateway connects to the TIBCO EMS Server.

Password:
Enter the password for this user.

SSL Tab:
The following tabs and fields are available on the **SSL Tab**:

Limit the use of SSL to improve performance:
If this option is selected, SSL is only used for establishing (mutual) authentication with the TIBCO EMS Server, which takes place during the initial SSL handshaking process. When the channel is set up, data sent over this channel is sent

in the clear and is not encrypted like in a typical SSL session.

Enable client verification of the host certificate or host name:
Select this option if you want to compare the Common Name (`cn`) X.509 attribute of the Distinguished Name in the TIBCO EMS Server's certificate. Typically, the SSL handshake requires that the common name in the host's certificate matches the name of the host machine. For example, to trust the certificate associated with the `www.abc.com site`, the certificate must have the common name attribute set to this name (`cn=www.abc.com`). If you wish to perform this check on the TIBCO EMS Server's certificate presented to the API Gateway during SSL setup, select this setting.

Expected Host Name:
In cases where the common name in the certificate is *not* the same as the host machine, you can override the default validation by specifying a host name that you expect instead of the host given in the common name of the server's certificate.

For example, a generic TIBCO EMS Server certificate is issued for testing purposes, and this certificate is created with a common name of `server` (`cn=server`). Now, assume that you want to create an SSL session with a TIBCO EMS Server running on a machine that is called `host`.

The default client verification of the host name setting checks to make sure that the host on which the TIBCO EMS Server is running is called `server` because this is what is in the common name of the certificate. However, the host name of this machine is `host`, and so this check fails.

In such cases, you must override the default host checking behavior by specifying the *expected* host name in this field. In this case, enter `host` in the **Expected Host Name** field.

Cipher suites to be used:
Specify the OpenSSL cipher suites that the API Gateway supports. The ciphers are negotiated during the SSL handshake with the TIBCO EMS Server so that the strongest and most secure ciphers that are common to both parties are used.

Trusted Certificates Tab:
You can select the CA (Certificate Authority) certificates that you consider trusted for setting up the connection to the TIBCO EMS Server on this tab.

The TIBCO EMS Server's certificate can be explicitly trusted by importing it into the Certificate Store and selecting it in the list. Alternatively, in a solution more typical for a Public Key Infrastructure, the CA certificate that issued the TIBCO EMS Server's certificate is imported into the Certificate Store and is selected in the list. In this case, a chain of trust is established because all certificates issued by the CA are implicitly trusted if the CA is considered trusted.

Client Identity Tab:
If you want to configure mutual authentication to the TIBCO EMS Server you must select a client certificate from the list that the API Gateway can use to authenticate to the TIBCO EMS Server. For the SSL channel to be established successfully, the TIBCO EMS Server must trust the client certificate selected here.

Important

If the selected client certificate has been issued by a CA (it is not self-signed), the certificate of this CA *must* be imported into the Trusted Certificate Store. If a chain of certificates exists (for example, the client certificate was issued by an intermediary CA, which was issued by the root CA), all intermediary CA certificates must be imported into the Certificate Store.

Wait for Response Packets

Overview

Packet Sniffers are a type of Passive Service. Rather than opening up a TCP port and *actively* listening for requests, the Packet Sniffer *passively* reads data packets off the network interface. The Sniffer assembles these Packets into complete messages that can then be passed into an associated policy.

Because the Packet Sniffer operates passively (does not listen on a TCP port) and transparently to the client, it is most useful for monitoring and managing Web Services. For example, you can deploy the Sniffer on a machine running a Web Server acting as a container for Web Services. Assuming that the Web Server is listening on TCP port 80 for traffic, the Packet Sniffer can be configured to read all packets destined for port 80 (or any other port, if necessary). The packets can then be marshaled into complete HTTP/SOAP messages by the Sniffer and passed into a policy that, for example, logs the message to a database.

Packet Sniffer Configuration

Because Packet Sniffers are mainly used as passive monitoring agents, they are usually created in their own Service Group. For example, you can create a new group for this purpose by right-clicking the API Gateway instance under the **Listeners** node in the Policy Studio tree, and selecting **Add Service Group**. Enter `Packet Sniffer Group` on the **Add Service Group** dialog.

You can then add a Relative Path Service to this Group by right-clicking the `Packet Sniffer Group`, and selecting **Add Relative Path**. Enter a path in the field provided, and select the policy that you want to dispatch messages to when the Packet Sniffer detects a request for this path (after it assembles the packets). For example, if the Relative Path is configured as `/a`, and the Packet Sniffer assembles packets into a request for this path, the request is dispatched to the policy selected in the Relative Path Service.

Finally, you can add the Packet Sniffer by right-clicking the `Packet Sniffer Group` node, selecting **Packet Sniffer** -> **Add**. Complete the following fields on the **Packet Sniffer** dialog:

Device to Monitor:
Enter the name or identifier of the network interface that the Packet Sniffer monitors. The default entry is `any`, but it is this is only valid on Linux. On UNIX-based systems, network interfaces are usually identified using names like `eth0`, `eth1`, and so on. On Windows, these names are more complicated (for example, `\Device\NPF_{00B756E0-518A-4144 ... }`).

Filter:
You can configure the Packet Sniffer to only intercept certain types of packets. For example, it can ignore all UDP packets, only intercept packets destined for port 80 on the network interface, ignore packets from a certain IP address, listen for all packets on the network, and so on.

The Packet Sniffer uses the **libpcap** library filter language to achieve this. This language has a complicated but powerful syntax that enables you to *filter* what packets are intercepted, and what packets are ignored. As a general rule, the syntax consists of one or more expressions combined with conjunctions, such as `and`, `or`, and `not`. The following table lists a few examples of common filters and explains what they filter:

Filter Expression	*Description*
`port 80`	Captures only traffic for the HTTP Port (i.e. 80).
`host 192.168.0.1`	Captures traffic to and from IP address 192.168.0.1.
`tcp`	Captures only TCP traffic.
`host 192.168.0.1 and port 80`	Captures traffic to and from port 80 on IP address 192.168.0.1.

`tcp portrange 8080-8090`	Captures all TCP traffic destined for ports from 8080 through to 8090.
`tcp port 8080 and not src host 192.168.0.1`	Captures all TCP traffic destined for port 8080 but not from IP address 192.168.0.1.

The default filter of `tcp` captures all TCP packets arriving on the network interface. For more information on how to configure filter expressions like these, see the **tcpdump** man page [http://www.tcpdump.org/tcpdump_man.html].

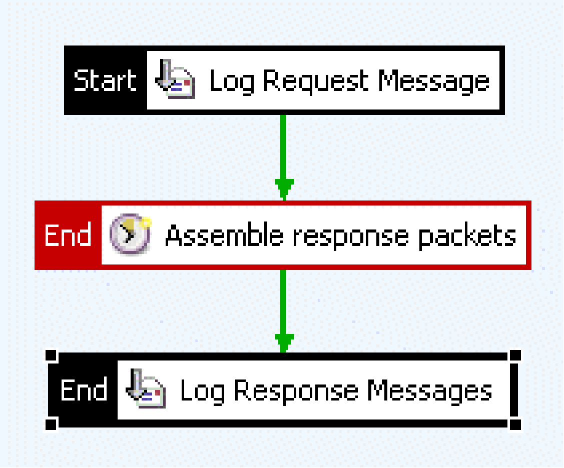
Promiscuous Mode:
When listening in promiscuous mode, the Packet Sniffer captures all packets on the same Ethernet network, regardless of whether the packets are addressed to the network interface that the Sniffer is monitoring.

Sniffing Response Packets

The API Gateway can capture both incoming and outgoing packets when it is listening passively (not opening any ports) on the network interface. For example, a Web Service is deployed in a web server that listens on port 80. The API Gateway can be installed on the same machine as the web server. It is configured *not* to open any ports and to use a Packet Sniffer to capture all packets destined for TCP port 80.

When packets arrive on the network interface that are destined for this port, they are assembled by the Packet Sniffer into HTTP messages and passed into the configured policy. Typically, this policy logs the message to an audit trail, and so usually consists of just a **Log Message** filter.

Assuming that you also want to log response messages passively, as is typically required for a complete audit trail, you can use the **Wait for Response Packets** filter to correlate response packets with their corresponding requests. The **Wait for Response Packets** filter assembles the response messages into HTTP messages and can then log them again using the **Log Message Payload** filter. The following policy logs both request and response messages captured transparently by the Packet Sniffer:



You can see from the policy that the first logging filter logs the *request* message. By this stage, the Packet Sniffer has assembled the request packets into a complete HTTP request, and this is what is passed to the **Log Request Message** filter. The **Assemble response packets** filter is a **Wait for Response Packets** filter that assembles response packets into complete HTTP response messages and passes them to the **Log Response Message** filter, which logs the complete response message. More information on the **Log Message Payload** filter is available in the *Log Message Payload* topic.

Proxy Servers

Overview

You can configure settings for individual proxy servers under the **External Connections** node in the Policy Studio tree, which you can then specify at the filter level (in the **Connection** and **Connect To URL** filters). When configured, the filter connects to the HTTP proxy server, which in turn routes the message on to the destination server named in the request URI. For more details, see *Connection* and *Connect to URL* .

These proxy server settings are different from the global proxy settings in the **Preferences** dialog in the Policy Studio, which apply only when downloading WSDL, XSD, and XSLT files from the Policy Studio. For more details, see the *Policy Studio Preferences* topic.

Configuration

To configure a proxy server under the **External Connections** tree node, right-click the **Proxy Servers** node, and select **Add a Proxy Server**. You can configure the following settings in the dialog:

Proxy Server Setting	***Description***
Name	Unique name or alias for these proxy server settings.
Host	Host name or IP address of the proxy server.
Port	Port number on which to connect to the proxy server.
Username	Optional user name when connecting to the proxy server.
Password	Optional password when connecting to the proxy server.
Scheme	Specifies whether the proxy server uses the HTTP or HTTPS transport. Defaults to HTTP.

DSS Signature Generation Service

Overview

This filter enables the API Gateway to generate XML Signatures as a service according to the OASIS Digital Signature Services (DSS) specification. The DSS specification describes how a client can send a message containing an XML Signature to a DSS Signature Web Service that can sign the relevant parts of the message, and return the resulting XML Signature to the client.

The advantage of this approach is that the Signature generation code is abstracted from the logic of the Web Service and does not have to be coded into the Web Service. Furthermore, in a Services Oriented Architecture (SOA), a centralized DSS server provides a single implementation point for all XML Signature related services, which can then be accessed by all services running in the SOA. This represents a much more manageable solution that one in which the security layer is coded into each Web Service.

Configuration

Complete the following fields to configure the **Sign Web Service** filter.

Name:
Enter a descriptive name for the filter in this field.

Signing Key:
Click the **Signing Key** button to select a private key from the Certificate Store. This key will be used to perform the signing operation.

DSS Signature Verification

Overview

This filter enables the API Gateway to verify XML Signatures as a service according to the OASIS Digital Signature Services (DSS) specification. The DSS specification describes how a client can send a message containing an XML Signature to a DSS Signature verification Web Service that can verify the Signature and return the result of the verification to the client.

The advantage of this approach is that the Signature verification code is abstracted from the logic of the Web Service and does not have to be coded into the Web Service. Furthermore, in a Services Oriented Architecture (SOA), a centralized DSS server provides a single implementation point for all XML Signature related services, which can then be accessed by all Services running in the SOA. This represents a much more manageable solution that one in which the security layer is coded into each Web Service.

Configuration

Complete the following fields to configure the **Verify Signature Web Service** filter.

Name:
Enter a descriptive name for the filter.

Find Signing Key:
The public key to be used to verify the signature can be retrieved from one of the following locations:

- **Via KeyInfo in Message:**
  The verification certificate can be located using the `<KeyInfo>` block in the XML Signature. For example, the certificate could be contained in a `<BinarySecurityToken>` element in a WSSE Security header. The `<KeyInfo>` section of the XML Signature can then reference this `BinarySecurityToken`. The API Gateway can automatically resolve this reference to locate the certificate that contains the public key necessary to perform the signature verification.
- **Via Selector Expression:**
  The certificate used to verify the signature can be extracted from the message attribute specified in the selector expression (for example, `${certificate}`). The certificate must have been placed into the specified attribute by a predecessor of the **Verify Signature Web Service** filter. For more details on selector expressions, see *Selecting Configuration Values at Runtime*.
- **Via Certificate in LDAP:**
  The certificate used to verify the Signature can be retrieved from an LDAP directory. Click the button next to this field, and select a previously configured LDAP directory in the tree. To add an LDAP directory, right-click the **LDAP Connections** tree node, and select **Add an LDAP Connection**. Alternatively, you can configure LDAP Connections under the **External Connections** node in the Policy Studio tree. For more details, see the topic on *Configuring LDAP Directories*.
- **Via Certificate in Store:**
  Finally, the verification certificate can be selected from the Certificate Store. Click the **Select** button to view the certificate that has been added to the store. Select the verification certificate by selecting the checkbox next to it in the table.

Encrypt and Decrypt Web Services

Overview

This filter allows the API Gateway to act as an XML *encrypting* Web Service, where clients can send up XML blocks to the API Gateway that are required to be encrypted. The API Gateway can then encrypt the XML data, replacing it with <EncryptedData> blocks in the message. The encrypted content is then returned to the client.

Similarly, the API Gateway can act as an XML *decrypting* Web Service, where clients can send up <EncryptedData> blocks to the API Gateway, which can then decrypt them and return the plain-text data back to the client.

By deploying the API Gateway as a centralized encryption/decryption service, clients distributed throughout an SOA (Services Oriented Architecture) can abstract out the security layer from their core business logic. This simplifies the logic of the client applications and makes the task of managing and configuring the security aspect a lot simpler since it is centralized.

Furthermore, the API Gateway's XML and cryptographic acceleration capabilities ensure that the process of encrypting and decrypting XML messages - a task that involves some very CPU-intensive operations - is performed at optimum speed.

Configuration

To configure both the **Encrypt Web Service** and **Decrypt Web Service** filters you simply need to enter a descriptive name for the filter in the **Name** field.

STS Web Service

Overview

This filter can be used to expose a Security Token Service, allowing clients to obtain security tokens for use within a SOA (Services Oriented Architecture) network.

Configuration

Complete the following field to configure the **STS Web Service** filter.

Name:
Enter a descriptive name for the filter in this field.

Consume WS-Trust Message

Overview

You can configure the API Gateway to consume various types of WS-Trust messages, including `RequestSecurityToken` (RST), `RequestSecurityTokenResponse` (RSTR), and `RequestSecurityTokenResponseCollection` (RSTRC) messages.

For more information on the various types of WS-Trust messages and their semantics and format, please see the WS-Trust specification.

Consume WS-Trust Message Types

The API Gateway can consume the following types of WS-Trust messages. Select the appropriate message type based on your requirements:

- **RST: RequestSecurityToken**
  The RST message contains a request for a single token to be issued by the Security Token Service (STS).
- **RSTR: RequestSecurityTokenResponse**
  The RSTR message is sent in response to an RST message from a token requestor. It contains the token issued by the STS.
- **RSTRC: RequestSecurityTokenResponseCollection**
  The RSTRC message contains an RSTR (containing a single issued token) for each RST that was received in an RSTC message.

Message Consumption

The configuration options available in this section enable you to extract various parts of the WS-Trust message and store them in message attributes for use in subsequent filters.

Extract Token:
Extracts a `<RequestedSecurityToken>` from the WS-Trust message and stores it in a message attribute. Select the expected value of the `<TokenType>` element in the `<RequestSecurityToken>` block. The default URI is `http://schemas.xmlsoap.org/ws/2005/02/sc/sct`.

Extract BinaryExchange:
Extracts a `<BinaryExchange>` token from the message and stores it in a message attribute. You should select the `ValueType` of the token from the drop-down list.

Extract Entropy:
The client can provide its own key material (entropy) that the token issuer may use when generating the token. The issuer can use this entropy as the key itself, it can derive another key from this entropy, or it can choose to ignore the entropy provided by the client altogether in favor of generating its own entropy.

Extract RequestedProofToken:
Select this option if you want to extract a `<RequestedProofToken>` from the WS-Trust message and store it in a message attribute for later use. You must select the type of the token (`encryptedKey` or `computedKey`) from the drop-down list.

Extract CancelTarget:
You can select this option to extract a `<CancelTarget>` block from the WS-Trust message and store it in a message attribute.

Extract RequestedTokenCancelled:
You can select this option to extract a `<RequestedTokenCancelled>` block from the WS-Trust message and store it

in a message attribute.

Match Context ID:
Select this option if you wish to correlate the response message from the STS with a specific request message. The `Context` attribute on the `RequestSecurityTokenResponse` message is compared to the value of the `ws.trust.context.id` message attribute, which contains the context ID of the current token request.

Extract Lifetime:
Select this option to remove the `<Lifetime>` elements from the WS-Trust token.

Extract Authenticator:
Select this option to extract the `<Authenticator>` from the WS-Trust token and store it in a message attribute.

Advanced

The following fields can be configured on the **Advanced** tab: **WS-Trust Namespace:**
Enter the WS-Trust namespace that you expect all WS-Trust elements to be bound to in tokens that are consumed by this filter. The default namespace is `http://schemas.xmlsoap.org/ws/2005/02/trust`.

Cache Security Context Session Key:
Click the button on the right, and select the cache to store the security context session key. The session key (the value of the `security.context.session.key` attribute), is cached using the value of the `security.context.token.unattached.id` message attribute as the key into the cache.

You can select a cache from the list of currently configured caches in the tree. To add a cache, right-click the **Caches** tree node, and select **Add Local Cache** or **Add Distributed Cache**. Alternatively, you can configure caches under the **Libraries** node in the Policy Studio tree. For more details, see the topic on *Global Caches*.

Lifetime of ComputedKey:
The settings in this section enable you to add a timestamp to the extracted `computedKey` using the values specified in the `<Lifetime>` element. This section is enabled only after selecting the **Extract RequestedProofToken** checkbox above, selecting the `computedKey` option from the associated dropdown list, and finally by selecting the **Extract Lifetime** checkbox. Configure the following fields in this section:

- **Add Lifetime to ComputedKey:**
  Adds the `<Lifetime>` details to the `security.context.session.key` message attribute. This enables you to check the validity of the key every time it is used against the details in the `<Lifetime>` element.
- **Format of Timestamp:**
  Specify the format of the timestamp using the Java date and time pattern settings.
- **Timezone:**
  Select the appropriate time zone from the drop-down list.
- **Drift:**
  To allow for differences in the clock times on the machine on which the WS-Trust token was generated and the machine running the API Gateway, you can enter a drift time here. The drift time allows for differences in the clock times on these machines and is used when validating the timestamp on the `computedKey`.

Verify Authenticator Using:
You can verify the authenticator using either the **Generated** or **Consumed** message. In either case you should select the appropriate type of WS-Trust message from the available options.

Create WS-Trust Message

Overview

You can configure the API Gateway to create various types of WS-Trust messages. The API Gateway can act both as a WS-Trust client when generating a `RequestSecurityToken` (RST) message, but also as a WS-Trust service, or Security Token Service (STS), when generating `RequestSecurityTokenResponse` (RSTR) and `RequestSecurityTokenResponseCollection` (RSTRC) messages.

A token requestor generates an RST message and sends it to the STS, which generates the required token and returns it in an RSTR message. If several tokens are required, the requestor can send up multiple RST messages in a single `RequestSecurityTokenCollection` (RSTC) request. The STS generates an RSTR for each RST in the RSTC message and returns them all in batch mode in an RSTRC message.

For more information on the various types of WS-Trust messages and their semantics and format, please see the WS-Trust specification.

Create WS-Trust Message Type

The **Create WS-Trust Message** filter can create the following types of WS-Trust message. Select the appropriate message type based on your requirements:

- **RST: RequestSecurityToken**
  The RST message contains a request for a single token to be issued by the STS.
- **RSTR: RequestSecurityTokenResponse**
  The RSTR message is sent in response to an RST message from a token requestor. It contains the token issued by the STS.
- **RSTRC: RequestSecurityTokenResponseCollection**
  The RSTRC message contains an RSTR (containing a single issued token) for each RST that was received in an RSTC message.

Message Creation

The settings on this tab specify characteristics of the WS-Trust message. The following fields are available:

Insert Token Type:
Select the type of token requested from the drop-down list. The type of token selected here is returned in the response from the STS. By default, the Security Token Context type is used, which is identified by the URI `http://schemas.xmlsoap.org/ws/2005/02/sc/sct`.

Binary Exchange:
You can use a `<BinaryExchange>` when negotiating a secure channel that involves the transfer of binary blobs as part of another security negotiation protocol (for example, SPNEGO). The contents of the blob are always Base64-encoded to ensure safe transmission.

Select the **Binary Exchange** option if you wish to use a negotiation-type protocol for the exchange of keys, such as SPNEGO. The URI selected in the **Value Type** field identifies the type of the negotiation in which the blob is used. The URI is placed in the `ValueType` attribute of the `<BinaryExchange>` element.

Entropy:
The client can provide its own key material (entropy) that the token issuer may use when generating the token. The issuer can use this entropy as the key itself, it can derive another key from this entropy, or it can choose to ignore the entropy provided by the client altogether in favor of generating its own entropy.

Select this option to generate some entropy, which is included in the `<wst:entropy>` element of the

`<wst:RequestSecurityToken>` block.

Insert Key Size:
The client can request the key size (in number of bits) required in a `<RequestSecurityToken>` request. However, the WS-Trust token issuer does not have to use the requested key size. It is merely intended as an indication of the strength of security required. The default request key size is 256 bits.

Insert Lifetime:
Select this option to insert a `<Lifetime>` element into the WS-Trust message. Use the associated fields to specify when the message expires. The lifetime of the WS-Trust message is expressed in terms of `<Created>` and `<Expires>` elements.

Lifetime Format:
The specified date/time pattern string determines the format of the `<Created>` and `<Expires>` elements. The default format is `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`, which can be altered if necessary. For more details on how to use this format, see the Javadoc for the `java.text.SimpleDateFormat` Java class in the Java Platform, Standard Edition 6 API Specification [http://java.sun.com/javase/6/docs/api/index.html].

Insert RequestedTokenCancelled:
Select this option to insert a `<RequestedTokenCancelled>` element into the generated WS-Trust message.

RST Creation

The following configuration fields specify the way in which a WS-Trust RST message is created:

Insert Request Type:
You can create two types of RST message. Select one of the following request types from the drop-down list:

- **Issue:** This type of RST message is used to request the STS to *issue* a token for the requestor.
- **Cancel:** This type of RST message is used to cancel a specific token.

Insert Key Type:
Select this option to insert the key type into the RST WS-Trust message.

Insert Computed Key Algorithm:
Select this option to insert the computed key algorithm into the message.

Insert Endpoint Reference:
Select this option and enter a suitable endpoint if you want to include an endpoint reference in the RST message.

RSTR Creation

The following configuration fields determine the way in which a WS-Trust RSTR message is created:

Insert RequestedProofToken:
Select this checkbox to insert a `<RequestedProofToken>` element into the generated WS-Trust message. The type of this token can be set to either `computedKey` or `encryptedKey` using the associated drop-down list.

Insert Authenticator:
Select this option to insert an authenticator into the RSTR message.

Advanced Settings

This section enables you to configure certain advanced aspects of the SOAP message that is sent to the WS-Trust Service.

WS-Trust Namespace:
Enter the WS-Trust namespace to bind all WS-Trust elements to in this field. The default namespace is `ht-`

`tp://schemas.xmlsoap.org/ws/2005/02/trust`.

WS-Addressing Namespace:
Select the WS-Addressing namespace version to use in all created WS-Trust messages.

WS-Policy Namespace:
Select the appropriate WS-Policy namespace from the drop-down list. The selected version selected can affect the ordering of tokens that are inserted into the WS-Security header of the SOAP message.

SOAP Version:
Select the SOAP version to use when creating the WS-Trust message.

Overwrite SOAP Method:
Select this option if you want the WS-Trust token to overwrite the SOAP method in the request. In this case, the token appears as a direct child of the SOAP Body element. You should use this option if you wish to preserve the contents of the SOAP Header, if present.

Overwrite SOAP Envelope:
Select this option if you want the generated WS-Trust message to form the entire contents of the message. In other words, the generated WS-Trust message replaces the original SOAP request.

Content-Type:
Specify the HTTP content-type of the WS-Trust message. For example, for Microsoft Windows Communication Foundation (WCF), you should use `application/soap+xml`.

Generate Authenticator Using:
You can verify the authenticator using the **Generated** or **Consumed** message. In either case, you should select the appropriate type of WS-Trust message from the available options.

Advanced Filter View

Overview

You can use the advanced filter view in the Policy Studio to edit all filter settings as text values. This enables you to edit each field as a text value regardless of whether the field is displayed as a radio button, checkbox, or drop-down list in the default user-friendly view for the filter.

This also means that you can specify all filter fields using the API Gateway selector syntax. This enables settings to be evaluated and expanded at runtime using metadata (for example, from message attributes, a Key Property Store (KPS), or environment variables). This is a powerful feature for System Integrators (SIs) and Independent Software Vendors (ISVs) when integrating with other systems.

Important

You should only modify filter settings using the advanced filter view under strict advice and supervision from the Oracle Support team.

Configuration

To enable the advanced filter view for a filter in the Policy Studio, press the **Shift** key when opening the filter. For example, you can press **Shift**, and double-click a filter on the policy canvas. Alternatively, you can press **Shift**, right-click the filter in the Policy Studio tree or policy canvas, and select **Edit**.

In the advanced filter view, settings are displayed with the following characters before the field name:

- Required: ***** (for example, ***name**)
- Reference: **^** (for example, **^proxyServer**)
- Radio attribute: **(:)** (for example, **(:)httpAuthType**)

Editing Filter Settings
You can specify all fields in this view using text values (for example, values such as `http://stockquote.com/stockquote/instance1`, `false`, `0`, `-1`, `500`, and so on). Alternatively, you can use the API Gateway Selector syntax to expand values at runtime. The following example selector expands the user agent header sent by the client in the `http.headers` message attribute:

```
${http.headers["User-Agent"]}
```

For example, this selector might return a user agent header such as the following at runtime:

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77
Safari/535.7
```

For more details on the API Gateway selector syntax, see the topic on *Selecting Configuration Values at Runtime*.

To confirm your updates, you must click **Save Changes** at the bottom right of the dialog. Alternatively, at any stage, you can click **Restore Defaults** to return to the original factory settings.

Returning to the Default Filter View
When you have finished editing filter settings in the Advanced Filter View, deselect the **Show Advanced Filter View** setting in **Preferences**. Then when you edit a selected filter on the policy canvas, the default user-friendly view for the filter is displayed.

Selecting Configuration Values at Runtime

Overview

A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata values (for example, from message attributes, a Key Property Store (KPS), or environment variables). The selector syntax uses the Java Unified Expression Language (JUEL) to evaluate and expand the specified values. Selectors provide powerful a feature when integrating with other systems or when customizing and extending the API Gateway.

When you press the **Shift** key and open a filter, you can edit all filter settings using text values in the advanced filter view. This means that you can specify all filter fields using the API Gateway selector syntax. For more details on the advanced view, see *Advanced Filter View*.

Selector Syntax

The API Gateway selector syntax uses JUEL to evaluate and expand the following types of values at runtime:

- Message attribute properties configured in message filters, for example:

  ```
  ${authentication.subject.id}
  ```

- Environment variables specified in `envSettings.props` and `system.properties` files, for example:

  ```
  ${env.PORT.MANAGEMENT}
  ```

- Values specified in a configured Key Property Store, for example:

  ```
  ${kps.CustomerProfiles[JoeBloggs].age}
  ```

Accessing Fields
A message attribute selector can refer to a field of that message (for example `certificate`), and you can use `.` characters to access subfields. For example, the following selector expands to the `username` field of the object stored in the `profile` attribute in the message:

```
${profile.username}
```

You can also access fields indirectly using square brackets (`[` and `]`). For example, the following selector is equivalent to the previous example:

```
${profile[field]}
```

You can specify literal strings as follows:

```
${profile[“a field name with spaces”]}
```

For example, the following selector gets a certificate from a Keyed Property Store by specifying its Distinguished Name:

```
${kps.certsByDName[“CN=Joe, O=Vordel”].certificate}
```

Note

For backwards compatibility with the `.` spacing characters used in previous versions of the API Gateway, if a selector fails to resolve with the above rules, the flat, dotted name of a message attribute still works. For example, `${content.body}` returns the item stored with the `content.body` key in the message.

Special Selector Keys
The following top-level keys are resolved specially:

Key	***Description***
`kps`	Subfields of the `kps` key reflect the alias names of Keyed Property Store objects configured for the local API Gateway. Further indexes represent objects looked up in that KPS (for example, `${kps.certsByDName["CN=Joe, O=Vordel"].certificate}`).
`env`, `system`	In previous versions, fields from the `envSettings.props` and `system.properties` files had restrictions on the prefixes used. The selector syntax does not require the `env` and `system` prefixes in these files. For example, conceptually `${env.` selects the settings from `envSettings.props`, and the rest of the selector chooses any properties in it. For compatibility, if a setting in either file starts with this prefix, it is stripped away so the selectors still behave correctly with previous installations.

Resolving Selectors
Each `${...}` selector string is resolved step-by-step, passing an initial context object (for example, `Message`). The top-level key is offered to the context object, and if it resolves the field (for example, the message contains the named attribute), the resolved object is indexed with the next level of key. At each step, the following rules apply:

1. At the top level, test the key for the global values (for example, `kps`, `system`, and `env`) and resolve those specially.
2. If the object being indexed is a Dictionary, KPS, or Map, use the index as a key for the item's normal indexing mechanism, and return the resulting lookup.
3. If all else fails, attempt Java reflection on the indexed object.

Note

Method calls are currently only supported using Java reflection. There are currently no supported functions as specified by the Unified Expression Language (EL) standard. For more details on JUEL, see `http://juel.sourceforge.net/`.

Example Selector Expressions

This section lists some example selectors that use expressions to evaluate and expand their values at runtime.

Message Attribute
The following message attribute selector returns the HTTP `User-Agent` header:

```
${http.headers["User-Agent"]}
```

For example, this might expand to the following value:

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77
Safari/535.7
```

Environment Variable

In a default configuration, the following environment variable selector returns port 8091:

```
${env.PORT.MANAGEMENT + 1}
```

Key Property Store

The following KPS selector returns the certificate for the entry in the Certificate Store with the `Joe Soap` alias:

```
${kps.certsByAlias[“Joe Soap”].certificate}
```

This returns the Distinguished Name in that certificate:

```
${kps.certsByAlias[“Joe Soap”].certificate.getSubjectDN()}
```

This returns the certificate identified by the alias stored in the `authentication.subject.id` attribute:

```
${kps.certsByAlias[authentication.subject.id]}
```

Examples Using Reflection

The following message attribute selector returns the CGI argument from an HTTP URL (for example, returns `bar` for `http://localhost/request.cgi?foo=bar`):

```
${http.client.getCgiArgument("foo")}
```

This returns the name of the top-level element in an XML document:

```
${content.body.getDocument().getDocumentElement().getNodeName()}
```

This returns true if the HTTP response code lies between 200 and 299:

```
${http.response.status / 200 == 2}
```

Extracting Message Attributes

There are a number of API Gateway filters that extract message attribute values (for example, **Extract Certificate Attributes** and **Retrieve from HTTP Header**). Using selectors to extract message attributes offers a more flexible alternative to using such filters. For more details on using selectors instead of these filters, contact Oracle Support (see *Oracle Contact Details*).

Key Property Stores

Overview

A Key Property Store (KPS) is an external data store of policy properties (typically, read frequently, and seldom written to). Using a KPS enables metadata-driven policies. Policy configuration data is stored in an external data store, which is looked up dynamically when policies are executed. You can specify configuration settings in Policy Studio and API Service Manager using selectors, which are evaluated and expanded at runtime.

For example, the following is a simple keyed property in an external data store:

- **Key**: `customerId`
- **URL**: `http://myapp.test.com`
- **Username**: `john.doe`
- **Password**: `changeme`

In this example, the KPS is configured in Policy Studio using an alias of `oauth`. You can use the following selector to specify the username in Policy Studio and API Service Manager:

```
${kps.oauth[customerId].username}
```

For more details on selectors, see *Selecting Configuration Values at Runtime*.

KPS Backing Data Stores

A KPS provides a consistent interface to object data in different backing data stores. By default, the API Gateway provides support for the following KPS backing stores:

- JSON file
- Oracle certificate store (DNAME to certificate, alias to certificate)
- Database (Oracle, Microsoft SQL Server, DB2, MySQL, JPA, DB schema)

The database support includes generic Java Persistence API (JPA) for serializing Java objects to a database, and existing or custom database schema for writing beans that map keyed property attributes to the schema.

Note

JSON files and certificate stores are cached on startup. A database-backed KPS is read each time.

You can configure a KPS using Policy Studio (for example, specify details such as store name, alias, type, and provider, or import and export a KPS). For web-based user interfaces, JavaScript-based KPS browser widgets are also available. You can use the KPS Service Provider Interface (SPI) to add custom stores. KPS functionality is exposed using a Java API and a REST API.

Configuring a Key Property Store

To configure a KPS in the Policy Studio, perform the following steps:

1. In the main Policy Studio tree, select **Libraries** -> **Key Property Stores**.
2. Click **Add** at the bottom of the screen.
3. Complete the following fields in the **Add Store** dialog:

- **Name**:
  Enter the KPS name (for example, `ApplicationRegister`).
- **Description**:
  Enter a description for your KPS.
- **Alias**:
  Enter an alias used to identify your KPS (for example, `appregister`).
- **Type**:
  Enter a type for your KPS (for example, `AppDetails`).
- **Store Implementation**:
  Enter an implementation for your KPS (for example, `com.vordel.kps.storeImpl.StoreImpl`).

4. Right-click the **Store Configuration Properties** table, and select **New Property**.
5. Enter a name-value pair in the **Add Property** table (for example, **Name** of `key` and **Value** of `clientID`).
6. Click **OK**.
7. Repeat to add multiple properties.

Further Information
For more detailed information on using a KPS, please contact the Oracle Support Team with your queries (see *Oracle Contact Details*).

Scripting Language Filter

Overview

The **Scripting Language** filter uses Java Specification Request (JSR) 223 [http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/] to embed a scripting environment in the API Gateway's core engine. This enables you to write bespoke JavaScript or Groovy code to interact with the message as it is processed by the API Gateway. You can get, set, and evaluate specific message attributes with this filter.

Because the scripting environment is embedded in the API Gateway engine, it has access to all Java classes on the API Gateway's classpath, including all JRE classes. If you wish to invoke a Java object, you must place its corresponding class file on the API Gateway classpath. The recommended way to add classes to the API Gateway classpath is to place them (or the JAR files that contain them) in the `INSTALL_DIR/ext/lib` folder. For more details, see the `readme.txt` in this folder.

Some typical uses of the **Scripting Language** filter include the following:

- Check the value of a specific message attribute
- Set the value of a message attribute
- Remove a message attribute
- DOM processing on the XML request or response

Writing a Script

To write a script filter, you must implement the `invoke()` method. This method takes a `com.vordel.circuit.Message` object as a parameter and returns a boolean result.

The API Gateway provides a **Script Library** that contains a number of pre-written `invoke()` methods to manipulate specific message attributes. For example, there are `invoke()` methods to check the value of the `SOAPAction` header, remove a specific message attribute, manipulate the message using the DOM, and assign a particular role to a user.

You can access the script examples provided in the **Script library** by clicking the **Show script library** button on the filter's main configuration screen. For a complete list of available message attributes, see the Message Attribute Reference.

Important

When writing the JavaScript or Groovy code, you should note the following:

- The `invoke()` method *must* be implemented.
- The `invoke()` method takes a `com.vordel.circuit.Message` object as a parameter, and returns a boolean.
- You can obtain the value of a message attribute using the `getProperty` method of the `Message` object.

Use Local Variables

The API Gateway is a multi-threaded environment, therefore, at any one time multiple threads can be executing code in a script. When writing JavaScript or Groovy code, always declare variables locally using `var`. Otherwise, the variables are global, and global variables can be updated by multiple threads.

For example, always use the following approach:

```
var myString = new java.lang.String("hello word");
for (var i = 100; i < 100; i++) {
    java.lang.System.out.println(myString + java.lang.Integer.toString(i));
}
```

Do not use the following approach:

```
myString = new java.lang.String("hello word");
for (i = 100; i < 100; i++) {
    java.lang.System.out.println(myString + java.lang.Integer.toString(i));
}
```

Using the second example under load, you cannot guarantee which value is output because both of the variables (`myString` and `i`) are global.

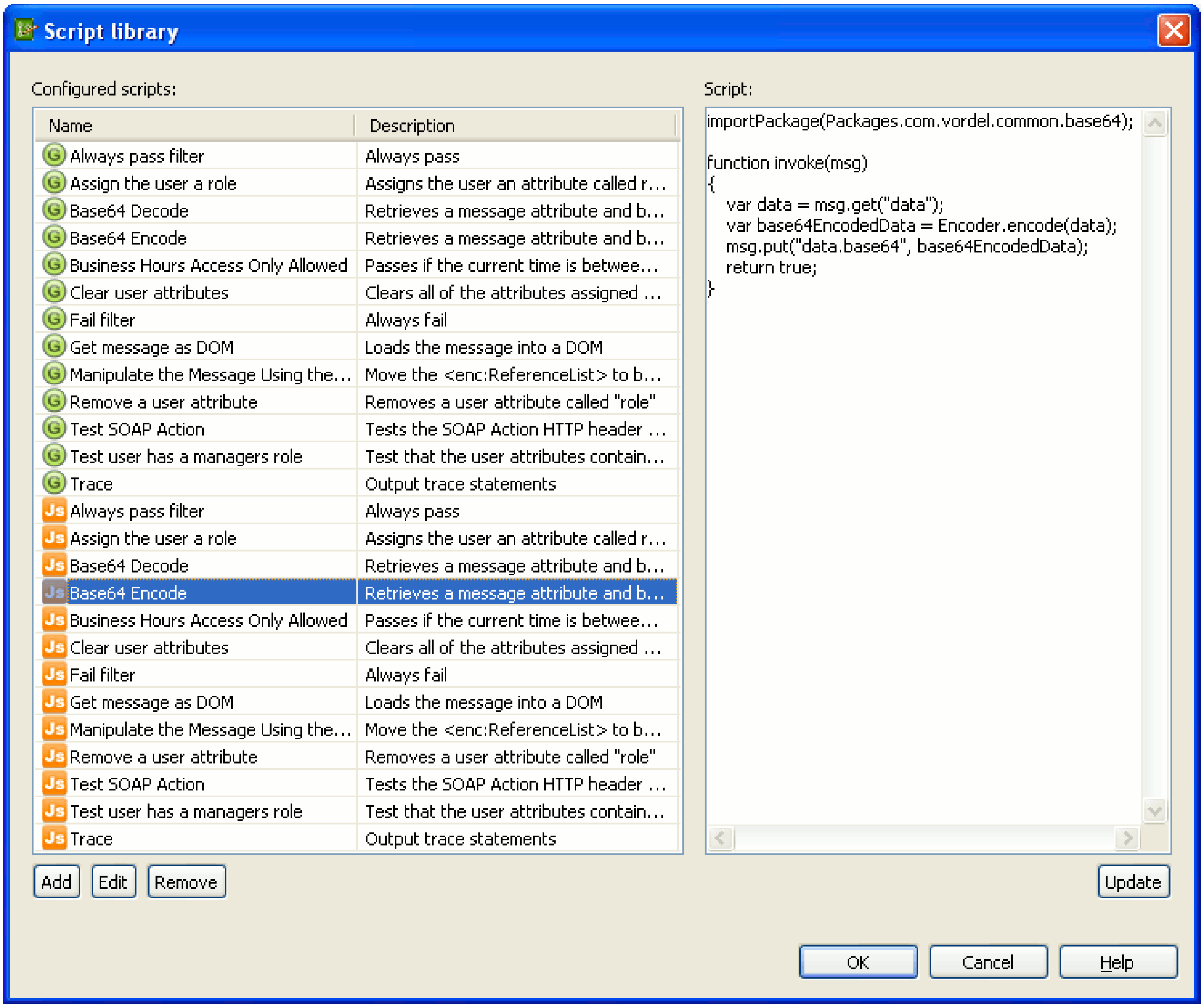
Configuring a Script Filter

You can write or edit the JavaScript or Groovy code in the text area on the **Script** tab. A JavaScript function skeleton is displayed by default. Use this skeleton code as the basis for your JavaScript code. You can also load an existing JavaScript or Groovy script from the **Script library** by clicking the **Show script library** button.

On the **Script library** dialog, click any of the **Configured scripts** in the table to display the script in the text area on the right. You can edit a script directly in this text area. Make sure to click the **Update** button to store the updated script to the **Script library**.

Adding a Script to the Library

You can add a new script to the library by clicking the **Add** button, which displays the **Script Details** dialog. Enter a **Name** and a **Description** for the new script in the fields provided. By default, the **Language** field is set to JavaScript, but you can also select Groovy from the drop-down list. You can then write the script in the **Script** text area.



Script library

Configured scripts:

Name	Description
G Always pass filter	Always pass
G Assign the user a role	Assigns the user an attribute called r…
G Base64 Decode	Retrieves a message attribute and b…
G Base64 Encode	Retrieves a message attribute and b…
G Business Hours Access Only Allowed	Passes if the current time is betwee…
G Clear user attributes	Clears all of the attributes assigned …
G Fail filter	Always fail
G Get message as DOM	Loads the message into a DOM
G Manipulate the Message Using the…	Move the <enc:ReferenceList> to b…
G Remove a user attribute	Removes a user attribute called "role"
G Test SOAP Action	Tests the SOAP Action HTTP header …
G Test user has a managers role	Test that the user attributes contain…
G Trace	Output trace statements
Js Always pass filter	Always pass
Js Assign the user a role	Assigns the user an attribute called r…
Js Base64 Decode	Retrieves a message attribute and b…
Js Base64 Encode	Retrieves a message attribute and b…
Js Business Hours Access Only Allowed	Passes if the current time is betwee…
Js Clear user attributes	Clears all of the attributes assigned …
Js Fail filter	Always fail
Js Get message as DOM	Loads the message into a DOM
Js Manipulate the Message Using the…	Move the <enc:ReferenceList> to b…
Js Remove a user attribute	Removes a user attribute called "role"
Js Test SOAP Action	Tests the SOAP Action HTTP header …
Js Test user has a managers role	Test that the user attributes contain…
Js Trace	Output trace statements

Script:

```
importPackage(Packages.com.vordel.common.base64);

function invoke(msg)
{
   var data = msg.get("data");
   var base64EncodedData = Encoder.encode(data);
   msg.put("data.base64", base64EncodedData);
   return true;
}
```

Writing a Custom Filter using the Oracle API Gateway SDK

Overview

Oracle API Gateway exposes several powerful APIs as part of its Software Development Kit (SDK) to enable users to build their own bespoke message filters. Users can leverage the API Gateway's pluggable and extensible architecture to enhance the message processing capabilities of the API Gateway core processing engine.

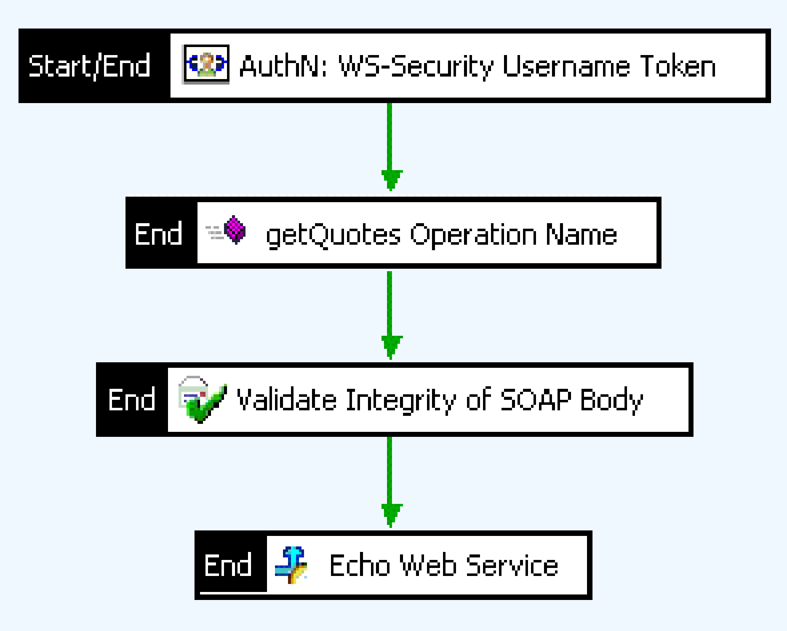
This tutorial walks you through a step-by-step example of how to build a custom message filter using the SDK, and integrate it into a policy. This tutorial shows how to build the two main aspects of an Addition filter: the server runtime component, and the Policy Studio configuration component. It then integrates these components into a policy, and shows how the filter adds the values of two parameters of a SOAP message together, and returns the result to the client. By the end of this tutorial, you should be able to write and test your own message filters by following a similar procedure.

Before going any further, it is important to explain exactly what are Oracle API Gateway filters, and how you can wire them together to create message processing policies.

Policies, Filters, and Message Attributes

A policy consists of a network of message filters where each filter is a modular executable unit that performs a specific type of processing on a message. The policy arranges these filters into sequences called paths. The filters then act as decision-making points along these paths, determining which filters are run on the message, and in what order.

The following four-node *policy* contains a single path with four *message filters*. The filter marked as *Start* (**AuthN: WS-Security Username Token**) is executed first. If this filter runs successfully, the next filter in the path (**getQuotes Operation Name**) is run, and so on until the last filter (**Echo Web Service**) in the path is executed.


Start/End
AuthN: WS-Security Username Token
End
getQuotes Operation Name
End
Validate Integrity of SOAP Body
End
Echo Web Service

When an HTTP request is received, it is converted into a set of message attributes. Each message attribute represents a specific characteristic of the HTTP request, such as the HTTP headers, HTTP body, and MIME parts, amongst others.

Every Filter declares the message attributes that it *requires*, *generates*, and *consumes* from the *attributes blackboard*. The blackboard contains all the available message attributes. When a filter *generates* message attributes, it puts them up on the blackboard so that when another Filter *requires* them it can pull them off the blackboard. If a filter *consumes* a message attribute, it is wiped from the blackboard so that no other filter in the policy can use it.

For example, the following table summarizes the attributes required and generated by the **Operation Name** resolver, which filters incoming requests based on their SOAP operation and namespace:

Filter Name:	Operation Name
Description:	Filters incoming SOAP requests based on their SOAP operation and namespace.
Required Attributes:	`content.body` `http.request.uri`
Generated Attributes:	`soap.request.method`

For more information on policy, policy building, filters, see the API Gateway documentation. The following list summarizes the important concepts introduced in this section:

- **Policy:**
  A network of interconnected message filters.
- **Message Filters:**
  An executable unit that performs a specific type of processing on message attributes.
- **Message Attributes:**
  Message attributes represent specific characteristics of a message.

Oracle API Gateway SDK Overview

The Oracle API Gateway Software Development Kit (SDK) comprises three Java packages that provide programmatic access to Oracle API Gateway policies, message objects, and the Entity Store. The following table describes these packages:

Package	***Description***
com.vordel.circuit [../../javadoc/com/vordel/circuit/Circuit.html]	This package is responsible for implementing the core Oracle API Gateway policy. It includes the base classes for all message filters and their associated classes.
com.vordel.mime [../../javadoc/com/vordel/mime/Multipart.html]	Includes classes that encapsulate the message as it passes through the Oracle API Gateway policy. It also provides programmatic access to HTTP headers, HTTP body, request query string (if present), and MIME parts.
com.vordel.es [../../javadoc/com/vordel/es/EntityStore.html]	These classes provide access to the underlying Entity Store where all configuration data is stored.

Tutorial Prerequisites

You should read *Oracle API Gateway Concepts* to make sure you understand the concepts of policies and filters before continuing.

The Oracle API Gateway SDK requires a JDK 1.6, and is supported for the Windows, Linux, and Solaris packages.

Oracle API Gateway SDK Sample Overview

The Oracle API Gateway SDK ships with a working example of a message filter, called the `SimpleFilter`, which demonstrates how to use the SDK to build a filter. The filter extracts two integer parameters from a SOAP message, adds the integers, and returns the result of the addition in a SOAP response to the client.

This tutorial documents the steps required to build, integrate, configure, and test the supplied `SimpleFilter` and `SimpleProcessor` classes. The steps are as follows:

Step	***Description***
Step 1: Create TypeDocs	Every filter has an associated XML-based TypeDoc that contains the entity's type definition. It defines the configuration field names for that filter and their corresponding data types.
Step 2: Create Filter Class	Every message filter has an associated Filter class that encapsulates the configuration data for a particular instance of the filter. It also returns the corresponding Processor and Policy Studio classes.
Step 3: Create Processor Class	The Processor class is the server runtime component that is responsible for processing the message. Every message filter has an associated Processor and Filter class.
Step 4: Create Policy Studio Classes	All Filters are configured using the Policy Studio. Every Filter has a configuration wizard that enables you to set each of the fields defined in the entity that corresponds to that Filter. You can then add the filter to a policy to process messages.
Step 5: Build Classes	When the classes are written, you must build them, and add them to the server and client classpaths. The Oracle API Gateway SDK provides example classes.
Step 6: Load TypeDocs	You must register the TypeDoc created for the filter in Step 1 with the Entity Store.
Step 7: Construct a Policy	Construct a policy that echoes messages back to the client, and then adds the newly created filter to it.
Step 8: Configure the SimpleFilter	Use the GUI component of the newly added filter to specify its configuration. Then test the functionality of the filter (and its configuration) using the Oracle **API Gateway Explorer** testing tool.

Step 1: Create the Typedocs

All configuration data is stored as entities in the Oracle API Gateway Entity Store. The Entity Store is an XML-based store that holds all configuration data required to run the Oracle core processing engine. Each configurable item has an entity type definition. The entity type definition is defined in an XML file known as the TypeDoc.

Entity types are analogous to class definitions in an object-oriented programming language. In the same way that instances of a class can be created in the form of objects, an instance of an entity type can also be created. Therefore it is useful to think of the entity type defined in a TypeDoc as a header file, and the entity itself as a class instance. All entities and their entity type definitions are stored in the Entity Store.

Every filter requires specific configuration data to perform its processing on the message. For example, the `SimpleFilter`, which extracts the values of two elements from a SOAP message, and adds them together, must be primed with the names and namespaces of those two elements.

Because a filter is a configurable item, it requires a new XML typedoc to be written containing an entity type definition for it. The entity type for a filter contains a set of configuration parameters and their associated data types and default values.

When an instance of the filter is added to a policy using the Policy Studio, a corresponding entity instance is created and stored in the Entity Store. Whenever the filter instance is invoked, its configuration data is read from the entity instance in the Entity Store.

TypeDoc Syntax

The following example XML shows how the TypeDoc lists the various fields that form the configuration data for the Filter.

```
<entityStoreData>
  <entityType name="SimpleFilter" extends="Filter">
    <!-- Name of filter class that encapsulates the config data -->
    <constant name="class" type="string"
              value="com.vordel.example.filter.SimpleFilter"/>
    <!-- List of config fields, their types, and their default values -->
    <field ... />
    <field ... />
    <field ... />
  </entityType>
<entityStoreData>
```

All TypeDocs must obey the following simple rules:

- Extend the Filter type
- Define a constant Filter class
- List the configuration fields for the entity

SimpleFilter Elements and Attributes

The following table describes the important elements and attributes from the `SimpleFilter` TypeDoc listed above:

Element	***Attribute***	***Description***
`<entityStoreData>`		The topmost wrapper element for the entire type definition.
`<entityType>`		Contains the type definition, including all its fields and their types.
`<entityType>`	`name`	The unique name for this type.
`<entityType>`	`extends`	Entity definitions are hierarchical and can inherit from other higher level types. All filters must extend the `Filter` type.
`<constant>`		A `<constant>` element is used to represent a read-only immutable property of the type.
`<constant>`	`name`	This attribute contains the name of the read-only property. In the example above, the named property is `class`, indicating that the value of this constant is the Java class that encapsulates the defined type. The name of this class must be specified as a `<constant>`.

Element	***Attribute***	***Description***
`<constant>`	`type`	Specifies the type of the `value` attribute. In this case, the value is the name of a Java class, which is a string.
`<constant>`	`value`	Contains the value of the named property, which is the name of the Java class that encapsulates this type `com.vordel.example.filter.Si` (`mpleFilter`).
`<field>`		Contains the definition of a single configuration field for this filter.
`<field>`	`name`	The name of the configuration field. You can see later in this tutorial how this name is used to get and set this property.
`<field>`	`type`	Specifies the data type of the named configuration field. For example, supported types include `string`, `boolean`, `encrypted`, and `integer`.
`<field>`	`cardinality`	Stipulates how many times this field can appear in an instance of the entity. For example, a cardinality of `1` means that this field can occur only once in an entity.
`<field>`	`default`	Specifies a default value for the configuration field, if appropriate.

SimpleFilter TypeDoc

The TypeDoc for the `SimpleFilter` is as follows:

```
<entityStoreData>
  <entityType name="SimpleFilter" extends="Filter">

    <!-- Name of Filter class that encapsulates this config entity -->
    <constant name="class" type="string"
                 value="com.vordel.example.filter.SimpleFilter"/>

    <!-- List of config params, their types, and their default values -->
    <field name="param1" type="string" cardinality="1" default="a"/>
    <field name="param1Namespace" type="string"
              cardinality="1" default="http://startvbdotnet.com/web/"/>
    <field name="param2" type="string" cardinality="1" default="b"/>
    <field name="param2Namespace" type="string"
              cardinality="1" default="http://startvbdotnet.com/web/"/>
  </entityType>
</entityStoreData>
```

All type and related information for the Filter is contained in the top-level `<entityStoreData>` element. The Filter type declaration together with its field definitions and types are child elements of the `<entityType>` element. Each field name is specified in the `name` attribute of the `<field>` element, while the type and default value for the field are specified in the `type` and `default` attributes, respectively.

You can also provide internationalized log messages by specifying an `<entity>` block of type `InternationalizationFilter` within the `<entityStoreData>` elements.

Now that you understand how the configuration data for the filter is defined, you can create the filter class.

Step 2: Create the Filter Class

A filter class encapsulates the type information defined in an entity's type definition. There are class members that correspond to each of the fields in the type definition. At runtime, when the filter is invoked, the filter class is instantiated with the configuration data for the appropriate entity instance. The filter class is responsible for the following tasks:

- Storing member variables corresponding to fields in the type definition
- Specifying the message attributes it requires, consumes, and generates
- Returning the corresponding server runtime class (the Processor)
- Returning the corresponding Policy Studio class

SimpleFilter Class
The following code shows the members and methods of the `SimpleFilter.java` example:

```
package com.vordel.example.filter;

import com.vordel.circuit.DefaultFilter;
import com.vordel.circuit.FilterConfigureContext;
import com.vordel.circuit.MessageProperties;
import com.vordel.es.EntityStoreException;

 /**
 SimpleFilter contains the local name of the two
 parameters (a and b) and contains the namespace that
 these elements belong to (http://startvbdotnet.com/web/)
  **/

public class SimpleFilter extends DefaultFilter {

    // element name of the first parameter
    String param1;
    // namespace of the first element
    String param1Namespace;
    // element name of the second parameter
    String param2;
    // namespace of the second parameter
    String param2Namespace;

    /**
     * Set the message attributes used by this filter
     */
    protected final void setDefaultProperties() {
        requiredProperties.add(MessageProperties.CONTENT_BODY);
    }

    /**
     * This method is called to set the config fields for the filter
     * @param ctx The configuration context for this filter
     * @param entity The entity object
     */
    public void configure(FilterConfigureContext ctx,
                            com.vordel.es.Entity entity)
                            throws EntityStoreException {

        super.configure(ctx, entity);
        // read the settings for the processor
        param1  = entity.getStringValue("param1");
        param1Namespace  = entity.getStringValue("param1Namespace");
        param2  = entity.getStringValue("param2");
        param2Namespace  = entity.getStringValue("param2Namespace");
    }

    /**
     * Returns the server runtime Processor class associated
     * with this Filter class.
     */
    public Class getMessageProcessorClass() {
        return SimpleProcessor.class;
    }

    /**
     * Returns the GUI component for this Filter
     */
    public Class getConfigPanelClass() throws ClassNotFoundException {
        // Avoid any compile or runtime dependencies on SWT and other UI
        // libraries by lazily loading the class when required.
        return
            Class.forName("com.vordel.example.filter.simple.SimpleFilterUI");
    }
}
```

SimpleFilter TypeDoc

At this point, it is worth revisiting the entity definition for the `SimpleFilter` entity to see how the class members correlate to the defined fields.

```
<entityType name="SimpleFilter" extends="Filter">

  <!-- Name of Filter class that encapsulates this config entity -->
  <constant name="class" type="string"
               value="com.vordel.example.filter.SimpleFilter"/>

  <!-- List of config params, their types, and their default values -->
  <field name="param1" type="string" cardinality="1" default="a"/>
  <field name="param1Namespace" type="string"
            cardinality="1" default="http://startvbdotnet.com/web/"/>
  <field name="param2" type="string" cardinality="1" default="b"/>
  <field name="param2Namespace" type="string"
            cardinality="1" default="http://startvbdotnet.com/web/"/>
</entityType>
```

SimpleFilter Methods

The Filter class members (`param1`, `param1Namespace`, `param2`, and `param2Namespace`) directly correspond to the field definitions in the type definition. These members are populated in the `configure` method of the Filter class, which is called by the framework when the server is started up initially, and whenever the server is refreshed. The `Entity` class provides getter and setter methods for the different data types (for example, string, boolean, integer, and so on). For more details, see the Entity Javadoc [../../javadoc/com/vordel/es/Entity.html].

There are two more important methods implemented in this class: `setDefaultProperties` and `getMessageProcessorClass`. The `setDefaultProperties` method enables the Filter to define the message attributes that it *requires*, *generates*, and *consumes* from the attributes blackboard. The blackboard contains all the available message attributes. When a filter *generates* message attributes, it puts them up on the blackboard so that when another Filter *requires* them, it can pull them off the blackboard. If a filter *consumes* a message attribute, it is wiped from the blackboard so that no other filter in the policy can use it.

The attributes are stored in String arrays (`reqProps`, `genProps`, `conProps`), which are inherited from the `VariablePropertiesFilter` class. In the case of the `SimpleFilter` class, the `content.body` attribute is required because the SOAP parameters must be extracted from the body of the HTTP request.

The next important method here is the `getMessageProcessorClass` method, which returns the server runtime component (the Processor class) that is associated with this Filter class. Each Filter class has a corresponding Processor class, which is responsible for using the configuration data stored in the Filter class to process the message. The next step looks at the `SimpleProcessor` class to see how it acts on the data stored in the `SimpleFilter` class.

Finally, the corresponding Policy Studio configuration class is returned by the `getConfigPanelClass` method, which in this case is the `com.vordel.example.filter.simple.SimpleFilterUI` class. This class is described in detail in Step 4 of this tutorial.

Step 3: Create Processor Class

The Processor class is responsible for performing the processing on the message. It uses the configuration data stored in the Filter class to determine how to process the message.

Note

This is the server runtime component of the filter that is returned by the `getMessageProcessorClass` of the Filter class described in the previous section.

Example Skeleton Code

The following skeleton code shows how the Processor *attaches* to the Filter class and uses its data to process the message. The following code is for illustration purposes only, and some of the `SimpleProcessor` code has been omitted.

```
package com.vordel.example.filter;

import java.io.ByteArrayInputStream;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import com.vordel.circuit.Circuit;
import com.vordel.circuit.CircuitAbortException;
import com.vordel.circuit.Filter;
import com.vordel.circuit.Message;
import com.vordel.circuit.MessageProcessor;
import com.vordel.circuit.MessageProperties;
import com.vordel.es.EntityStore;
import com.vordel.mime.Body;
import com.vordel.mime.ContentType;
import com.vordel.mime.HeaderSet;
import com.vordel.mime.XMLBody;
import com.vordel.trace.Trace;

/**
 * This Processor acts as a simple Addition Web Service.
 * It extracts two parameters from a SOAP message and adds them together.
 * The result is then returned to the client.
 * The incoming message is expected in the following format:

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://startvbdotnet.com/web/">
```

```
      <a>1</a>
      <b>1</b>
    </Add>
  </soap:Body>
</soap:Envelope>

 The SimpleFilter contains the local name of the two parameters (a
 and b), and contains the namespace that these elements belong to
 (http://startvbdotnet.com/web/).
 */
public class SimpleProcessor extends MessageProcessor {

     /**
     * This method attaches the Filter to the Processor object.
     * This is called at startup and on every refresh.
     * This should contain server-side config/initialization.
     * For example, if this filter is required to establish
     * connections to any 3rd party products/servers, the
     * connection setup should be done here.
     * @param ctx Configuration context for the filter.
     * @param entity The Entity object
     */
    public void filterAttached(FilterConfigureContext ctx,
                                com.vordel.es.Entity entity)
                                throws EntityStoreException {
        // nothing to do here for initialisation
        super.filterAttached(ctx, entity);
    }

    /**
     * The invoke method performs the filter processing.
     * @param c The policy circuit
     * @param message The message
     * @return true or false.
     */
    public boolean invoke(Circuit c, Message message)
      throws CircuitAbortException {

        try {
            // Get the incoming request message as a DOM
            Document doc = getDOM(message);

            // Default result
            String result = "UNKNOWN";

            // Cast the filter member variable to a SimpleFilter so that
            // you may access the values stored in the SimpleFilter's
            // fields (for example, param1, param1Namespace, and so on).
            SimpleFilter f = (SimpleFilter)filter;

            // Look into the DOM to get the two parameters.
            // Get the 1st parameter
            NodeList param1 =
                doc.getElementsByTagNameNS(f.param1Namespace, f.param1);
            if (param1 == null || param1.getLength() <= 0)
                throw new CircuitAbortException(
                    "Could not find " + f.param1 + "in message");
            // Get the value passed in the 1st parameter
            String a = getElementContent((Element)param1.item(0));

            // Get the 2nd parameter
            NodeList param2 =
                doc.getElementsByTagNameNS(f.param2Namespace, f.param2);
            if (param2 == null || param2.getLength() <= 0)
                throw new CircuitAbortException(
                    "Could not find " + f.param2 + "in message");

            // Get the value of the 2nd parameter
            String b = getElementContent((Element)param2.item(0));

            // Calculate the result by adding the two parameter values
            result =
                Integer.toString(Integer.parseInt(a) + Integer.parseInt(b));

            // Set the response by setting the content body
            // to be the response
            HeaderSet responseHeaders = new HeaderSet();
            responseHeaders.putString("Content-Type", "text/xml");
            StringBuffer response = new StringBuffer(RESPONSE_START);
            response.append(result);
            response.append(RESPONSE_END);
            Body convertedBody =
                Body.create(responseHeaders, new ContentType("text/xml"),
                    new ByteArrayInputStream(
                        response.toString().getBytes()));
            message.setProperty(
                MessageProperties.CONTENT_BODY, convertedBody);

            return true;
        }
        catch (IOException exp) {
            Trace.error("IOException in SimpleProcessor: " + exp.getMessage());
            return false;
        }
    }
}
```

Processor Methods

There are two important methods that *must* be implemented by every Processor: the `filterAttached` method and the `invoke` method. The `filterAttached` method associates an appropriate Filter class with the Processor. The Processor can then access the configuration data stored in the Filter class. In this case, the `SimpleProcessor` attaches to the `SimpleFilter` class. The `filterAttached` method should contain any server-side initialization or configuration that is to be performed by the Filter, such as connecting to third-party products or servers.

The `invoke` method is responsible for using the data stored in the attached Filter class to perform the message processing. This method is called by the server as it executes the series of filters in any given policy. In the case of the `SimpleFilter`, the `invoke` method extracts the values of the <a> and <b> elements from the SOAP message, and adds the two values together. The result is then returned to the client in a templated SOAP response.

Important

The `invoke` method can have the following possible results:

Result	***Description***
`True`	If the filter processed the message successfully (for example, successful authentication, schema validation passed, and so on), the `invoke` method should return a true result, meaning that the next filter on the success path for the filter is invoked.
`False`	If the filter's processing fails (for example, the user was not authenticated, message failed integrity check, and so on), the `invoke` method should return false, meaning that the next filter on the failure path for the filter is invoked.
`CircuitAbortException`	If for some reason the filter cannot process the message at

Result	***Description***
	all (for example, if it can not connect to an Identity Management server to authenticate a user), it should throw a `CircuitAbortException`. If a `CircuitAbortException` is thrown in a policy, the designated Fault Processor (if any) is invoked instead of any successive filters on either the success or failure paths.

Now that you have a class to encapsulate the configuration data and another class to act on that data, it is now time to create some GUI classes where the user can configure the fields stored in the Filter class.

Step 4: Create Policy Studio Classes

The next step involves writing two GUI classes that enable the fields defined in the `SimpleFilter` type definition to be configured. When the GUI classes and resources are built, the visual components can be used in the Policy Studio to configure the Filter and add it to a policy.

SimpleFilter GUI Classes and Resources
The following table describes the GUI classes and resources for the `SimpleFilter`:

Class or Resource	***Description***
`SimpleFilterUI.java`	This class lists the pages that are involved in a Filter's configuration screen. Each Filter has at least two pages: the main configuration page, and a page where log messages related to the filter can be customized. This class is returned by the `getConfigPanelClass` method of the `SimpleFilter` class.
`SimpleFilterPage.java`	This class defines the layout of the visual fields on the Filter's main configuration screen. For example, there are four text fields on the configuration screen for the `SimpleFilter` corresponding to the four fields defined in the entity type definition.
`resources.properties`	This file contains all text displayed in the GUI configuration screen (for example, dialog titles, field names, and error messages). This means that the text can be customized or internationalized easily without needing to change code.
`simple.gif`	This image file is the icon that identifies the Filter in the Management Console, and is displayed in the Filter Palette.

This step first looks at the `SimpleFilterUI` class, which is returned by the `getConfigPanelClass` method of the `SimpleFilter` class. It is responsible for the following:

- Listing the configuration pages that make up the interface for the filter
- Naming the category of filters to which this filter belongs
- Specifying the name of the images to use as the icons/images for this filter

SimpleFilterUI Class

The code for the `SimpleFilterUI` is as follows:

```
package com.vordel.example.filter.simple;

import java.util.Vector;

import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.swt.graphics.Image;

import com.vordel.client.manager.Images;
import com.vordel.client.manager.filter.DefaultGUIFilter;
import com.vordel.client.manager.wizard.VordelPage;

/**
 * Filter configuration GUI for 'Simple' example filter.
 * This class shows how to code simple text fields for configuring a
 * custom filter.
 */
public class SimpleFilterUI
    extends DefaultGUIFilter
{
    /**
     * Add the pages you want to show in the configuration wizard for the filter.
     */
    public Vector<VordelPage> getPropertyPages() {
        Vector<VordelPage> pages = new Vector<VordelPage>();

        // Add the panel for configuring the specific fields
        pages.add(new SimpleFilterPage());

        // Add the page which allows the user to set the log strings for the
        // audit trail, for the pass/fail/error cases
        pages.add(createLogPage());

        return pages;
    }

    /**
     * Set the categories in which you want to display this Filter. The
     * categories define the sections of the palette in which the Filter
     * appears. The values returned should be the localized name of the
     * palette section, so ensure that the property is defined in the
     * resources.properties in this class's package. You will add this
     * file to the "Example Filters" category.
     */
    public String[] getCategories() {
        return new String[]{_("FILTER_GROUP_EXAMPLE")};
    }

    /*
     *  Register our custom images with the image registry
     */
    private static final String IMAGE_KEY = "simpleFilter";
    static {
        Images.getImageRegistry().put(IMAGE_KEY,
                Images.createDescriptor(SimpleFilterUI.class, "simple.gif"));
    }

    /**
     *  The icon image needs to be added in images.properties in com.vordel.client.manager
     *  the id used there is used as a reference here.
     *  Use this method to get image id for the small icon image in Images.get(id), etc.
     */
    public String getSmallIconId() {
        return IMAGE_KEY;
    }

    /**
     * Implement this method if you want to display a non-default image
     * for your filter in the policy editor canvas and navigation tree.
     */
    public Image getSmallImage() {
        return Images.get(IMAGE_KEY);
    }

    /**
     * Implement this method to display a non-default icon for your filter in
     * the palette.
     */
    public ImageDescriptor getSmallIcon() {
        return Images.getImageDescriptor(IMAGE_KEY);
    }
}
```

SimpleFilterUI Methods
The following table describes the important methods:

Method	***Description***
`public Vector getPropertyPages()`	Initializes a Vector of the Pages that makeup the total configuration screens for this Filter. Successive Pages are accessible by clicking the **Next** button on the Policy Studio configuration screen.
`public String[] getCategories()`	This method returns the names of the Filter categories that this Filter belongs to. The Filter is displayed under these categories in the Filter Palette in the Policy Studio. The `SimpleFilter` is added to the **Example Filters** category.
`public Image getSmallImage()`	The default image for the Filter, which is registered in the static block in the code above, can be overridden by returning a different image here.
`public ImageDescriptor getSmallIcon()`	The default icon for the Filter can be overridden by returning a different icon here.

A *Page* only represents a single configuration screen in the Policy Studio. You can chain together several Pages to form a series of configuration screens that together make up the overall configuration for a given Filter. By default, all Filters consist of two pages: one for the configuration fields for the Filter, and the other to allow per-Filter logging. However, there is no reason why more Pages can not be chained together. Successive Pages should be added to the configuration in the `getPropertyPages` method.

From looking at the `getPropertyPages` method of the `SimpleFilterUI`, it is clear that the `SimpleFilterPage` class forms one of the configuration screens (or pages) for the `SimpleFilter` filter. The `SimpleFilterPage` class is responsible for the layout of all the input fields that make up the configuration screen for the `SimpleFilter`.

SimpleFilterPage Class
The code for the `SimpleFilterPage` class is shown below:

```

package com.vordel.example.filter.simple;
```

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;

import com.vordel.client.manager.wizard.VordelPage;

public class SimpleFilterPage extends VordelPage
{
    /**
     * Create the configuration page. Set the title and description
     * here. The title and description are maintained in the
     * resources.properties file for customization and
     * internationalization purposes.
     */
    public SimpleFilterPage() {
        // Call the super constructor with a unique name for this
        // page to bind it with its corresponding wizard.
        super("simplePage");
        setTitle(_("SIMPLE_PAGE"));
        setDescription(_("SIMPLE_PAGE_DESCRIPTION"));
        setPageComplete(false);
    }

    /**
     * Get the unique identifier for the help page for this filter.
     * The ID-to-HTML page mapping is maintained in the following file:
     * <policy_studio_build>/plugins/
     * com.vordel.client.rcp.common.resources_version/contexts.xml.
     */
    public String getHelpID() {
        return "simple.help";
    }

    /**
     * Any post-processing of the configuration values can happen
     * here, before they are persisted to the Entity Store. If the
     * configuration is not complete and valid, notify the user here
     * with a dialog box and return false, otherwise return true.
     *
     * @see com.vordel.client.manager.util.MsgBox
     */
    public boolean performFinish() {
        // Simple mutually independent values here, no checking required,
        // so return true
        return true;
    }

    /**
     * Create the main control for the Filter configuration dialog.
     * This will include the text fields for setting the particular
     * Field Values in the Entity.
     */
    public void createControl(Composite parent) {
        // Create a Panel with two columns
        GridLayout layout = new GridLayout();
        layout.numColumns = 2;
        Composite container = new Composite(parent, SWT.NULL);
        container.setLayout(layout);

        // Add controls to populate the appropriate Entity Fields
        // You use the localization keys for the field names and
        // descriptions which will map to entries in the
        // resources.properties file.
        createLabel(container, "SF_NAME");
        createTextAttribute(container, "name", "SF_NAME_DESC");

        createLabel(container, "SF_PARAM1");
        createTextAttribute(container, "param1", "SF_PARAM1_DESC");

        createLabel(container, "SF_PARAM1NS");
        createTextAttribute(container, "param1Namespace", "SF_PARAM1NS_DESC");

        createLabel(container, "SF_PARAM2");
        createTextAttribute(container, "param2", "SF_PARAM2_DESC");

        createLabel(container, "SF_PARAM2NS");
        createTextAttribute(container, "param2Namespace", "SF_PARAM2NS_DESC");

        // Finish up the page definition
        setControl(container);
        setPageComplete(true);
    }
}
```

SimpleFilterPage Methods

There are four important interface methods that must be implemented in this class:

Method	***Description***
`public SimpleFilterPage()`	The constructor performs some basic initialization, such as setting a unique ID for the page, and setting the title and description for the page. The text representing the page title and description are kept in the `resources.properties` file so that they can be localized or customized easily, if necessary.

`public String getHelpID()`	This method is called by the Policy Studio help system. There is a **Help** button on every configuration page in the Policy Studio. When this button is pressed, the help system is invoked. Every page has a help ID (for example, `simple.help`) associated with it, which is mapped to a help page. This mapping is defined in the `/plugins/com.vordel.rcp.policystudio.resources_<version>/contexts.xml` file under the directory where you have installed Policy Studio. The following shows an example mapping in this file:
	`<context id="simple_help">`
	`<description>>Simple filter test</description>`
	`<topic label="Simple Filter" href="html/general_filter.html"/>`
	`</context>`
	Note
	A dot in a help ID is replaced by an underscore in the `contexts.xml` file as in the example above. All URLs specified in the `contexts.xml` file are relative from the `/plugins/com.vordel.rcp.policystudio.resou`

Method	***Description***
	rces_<version> folder of your Policy Studio installation. You can add the lines from the example above to the contexts.xml file now, and you can check that the help page works at the end of this tutorial.
public boolean performFinish()	This method gives you the chance to process the user-specified data before it is submitted to the Entity Store. For example, any validation on the data should be added to this method.
public void createControl(Composite parent)	This method is responsible for creating and ordering the input fields on the configuration page. Again, localization keys from the resource.properties file are used to give labels for the input fields. **Note** createTextAttribute takes a String as its second parameter, which corresponds to a field defined for an entity (for example, param, param1Namespace, param2, and param2Namespace are all defined in the SimpleFilter entity type). When the user submits the values entered in these fields, the values are set to the corresponding fields in the entity instance in the Entity Store.

resources.properties File

Both the SimpleFilterUI and the SimpleFilterPage classes use localized keys for all text that is displayed on the configuration screen. This makes it easy to localize or customize all text that is displayed in the Policy Studio. The localization keys and their corresponding strings are stored in the resources.properties file, which takes the following format:

```
# Palette category for example filters
FILTER_GROUP_EXAMPLE=Example Filters

# Title and Description for the SimpleFilter
SIMPLE_PAGE=Simple Filter Configuration
SIMPLE_PAGE_DESCRIPTION=Configure parameter values for the Simple Filter

# Field labels and descriptions
SF_NAME=Filter Name:
SF_NAME_DESC=The name of the Simple Filter
SF_PARAM1=Parameter 1
SF_PARAM1_DESC=the first parameter
SF_PARAM1NS=Parameter 1 Namespace
SF_PARAM1NS_DESC=the first parameter namespace field
SF_PARAM2=Parameter 2
SF_PARAM2_DESC=the second parameter
SF_PARAM2NS=Parameter 2 Namespace
SF_PARAM2NS_DESC=the second parameter namespace field
```

The final resource is the `simple.gif` image file, which is displayed as the icon for the `SimpleFilter` in the Policy Studio. You can see this icon later in this tutorial when you configure the `SimpleFilter`.

Now that all classes and resources have been written, it is now time to build the relevant JAR files and incorporate them into the product.

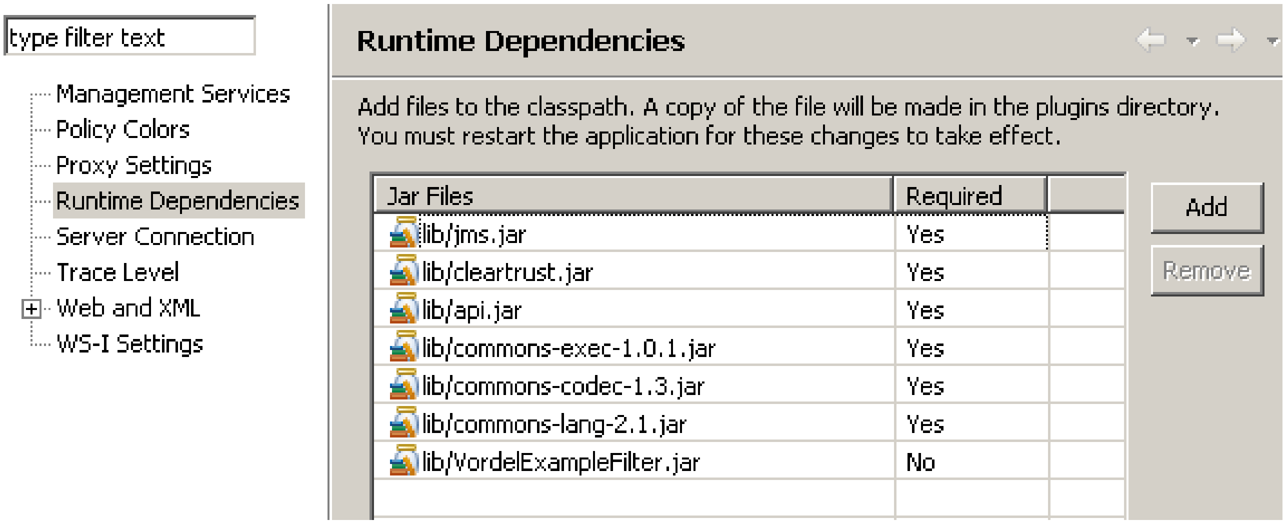
Step 5: Build Classes

You must perform the following steps to build the Java classes and resources described in the previous sections.

⚠ Important

The classes must be built against a 1.6 JDK because this is used to build the API Gateway, which contains the JAR files for the API Gateway SDK API.

1. Build the classes and associated resources into a JAR file using your chosen build technology (see example below).
2. Place the new JAR in the `INSTALL_DIR/ext/lib` directory, where `INSTALL_DIR` refers to the root of your API Gateway installation.
3. In the Policy Studio main menu, select **Window** -> **Preferences** -> **Runtime Dependencies**, and click **Add** to browse to the new JAR, and add it to the list (for example, `INSTALL_DIR/ext/lib/VordelExampleFilters.jar`).
4. Place any third-party JAR files used by your classes into the `INSTALL_DIR/ext/lib`, and add them to the list of **Runtime Dependencies** in the Policy Studio.
5. Restart the API Gateway.
6. Restart the Policy Studio using the following command:
   `policystudio -clean`



Runtime Dependencies

Add files to the classpath. A copy of the file will be made in the plugins directory.
You must restart the application for these changes to take effect.

Jar Files	Required
lib/jms.jar	Yes
lib/cleartrust.jar	Yes
lib/api.jar	Yes
lib/commons-exec-1.0.1.jar	Yes
lib/commons-codec-1.3.jar	Yes
lib/commons-lang-2.1.jar	Yes
lib/VordelExampleFilter.jar	No

Example Build File
For an example of building the `SimpleFilter` classes, see the Apache Ant `build.xml` file supplied in the `SDK_HOME/example/filter/src` directory, where `SDK_HOME` points to the root of your Oracle API Gateway SDK installation. For more details, see the instructions in `SDK_HOME/example/readme.html`.

The Ant file builds the `SimpleFilter` classes and packages all associated resources into the `VordelExampleFilters.jar` file. You must then place this file into the `INSTALL_DIR/ext/lib` folder, and add it to the **Runtime Dependencies** in the Policy Studio **Preferences**.

When both the server and the Policy Studio boot up, they automatically pick up the new JAR file. The remaining steps of this tutorial describe how to configure a policy that includes the `SimpleFilter`, and then test its functionality.

Step 6: Load TypeDocs

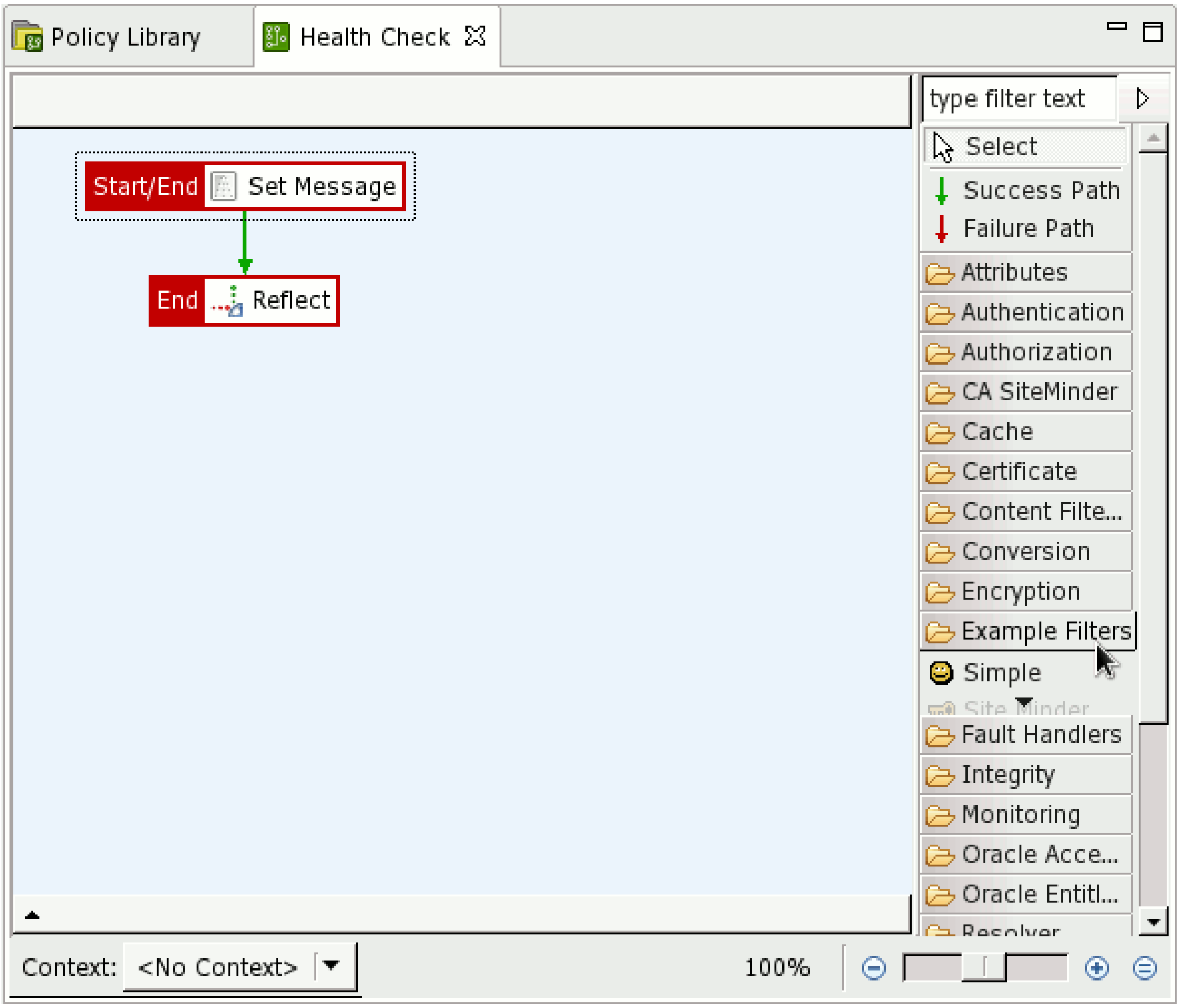
You must now register the type definition for the `SimpleFilter` with the Entity Store using the Policy Studio. When the entity type is registered, any time the server needs to create an instance of the `SimpleFilter`, the instance contains the correct fields with the appropriate types.

Register using the Policy Studio
To register the type definition using the Policy Studio, perform the following steps:

1. Start the Policy Studio, and connect to the API Gateway.
2. Select **Window** -> **Show View** -> **Tag/Profile Manager** from the main menu to display the **Tag/Profile Manager** tab.
3. Create a copy of the active configuration. To do this, expand the **Core Configurations** tree node, and right-click the active configuration (for example, **Default Core Configuration**). Select **Create via Copy** in the context menu, and give the new configuration a meaningful name (for example, `Custom Filter Config`).
4. Right-click the new configuration, and select **Edit** from the context menu. You are asked to enter a passphrase. If this has not been changed from the default, you can leave the field blank and proceed. This displays the **Profile Editor** tab with **Custom Filter Config** as a root node. Right-click this root node, and select **Import Custom Filter Types** from the menu.
5. Browse to the `sampleTypeSet.xml` file, which is located in the `SDK_HOME/example/src/com/vordel/example/filter` directory. A *TypeSet* file is used to group together one or more TypeDocs. This enables multiple TypeDocs to be added to the Entity Store in batch mode. The `sampleTypeSet.xml` file includes the following:

```
<typeSet>
    <!-- SimpleFilter TypeDoc -->
   <typedoc file="SimpleFilter.xml"/>
</typeSet>
```

6. After selecting the TypeSet, the workspace refreshes. To verify that the filter is available, select an existing policy in the Policy Studio tree, and you should see the **Example Filters** category in the palette, which contains the new custom filters.



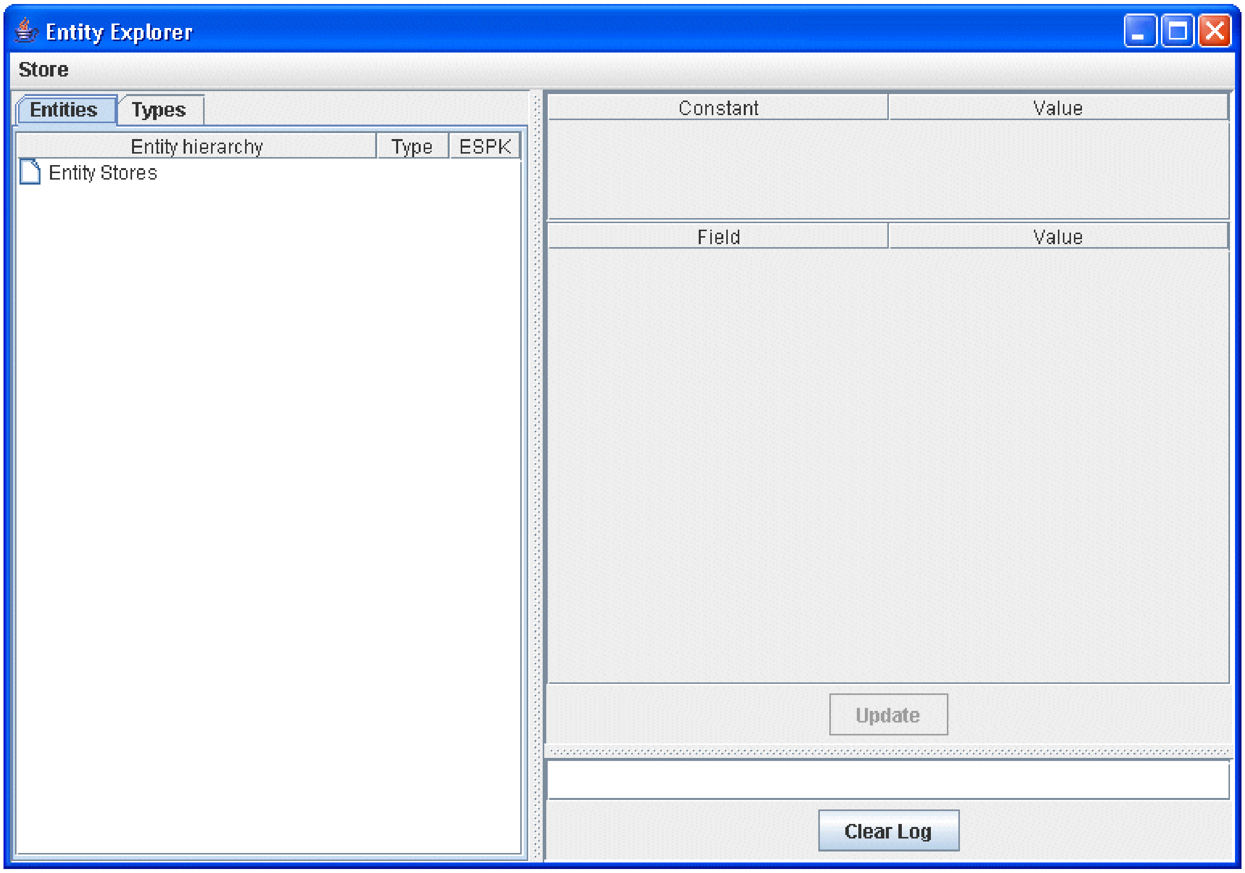
7. Right-click the configuration, and commit this version. From now on, any further revisions of this version, or configurations copied from this configuration contain the new custom filter types.
8. Click the **Deploy** button in the toolbar to deploy the new `Custom Filter Config` with the latest version containing the new filter types.

Verifying using the Entity Explorer

Another way to verify that your new filter has been installed is to use the **Entity Explorer**. You can use the **Entity Explorer** tool for browsing the entity types and entity instances that have been registered with the Entity Store.

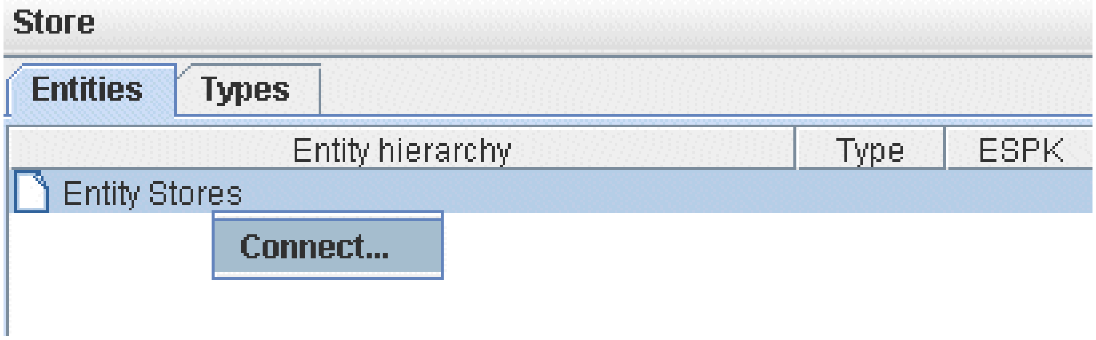
To verify that the filter has been installed, perform the following steps:

1. Start the Entity Explorer from the `/bin` directory of your installation using the `esexplorer` startup script. The Entity Explorer is displayed as follows:

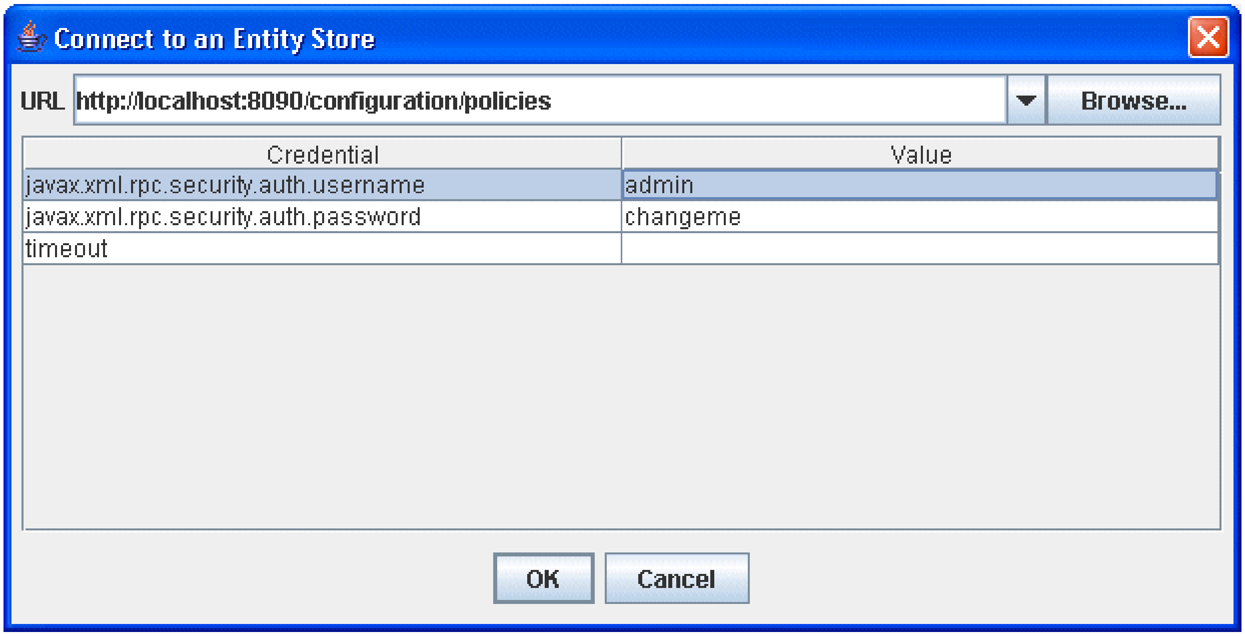


There are two main tabs on the Entity Store's interface: **Entities** and **Types**. The **Types** tab lists all currently defined entity types that have been registered with the Entity Store, while the **Entities** tab lists all the instances of entities that have been committed to the Entity Store (for example, configured filters).

2. You must first point the Entity Explorer at the management interface exposed by a running instance of the API Gateway. Right-click the **Entity Stores** node in the **Entity Hierarchy** tree:



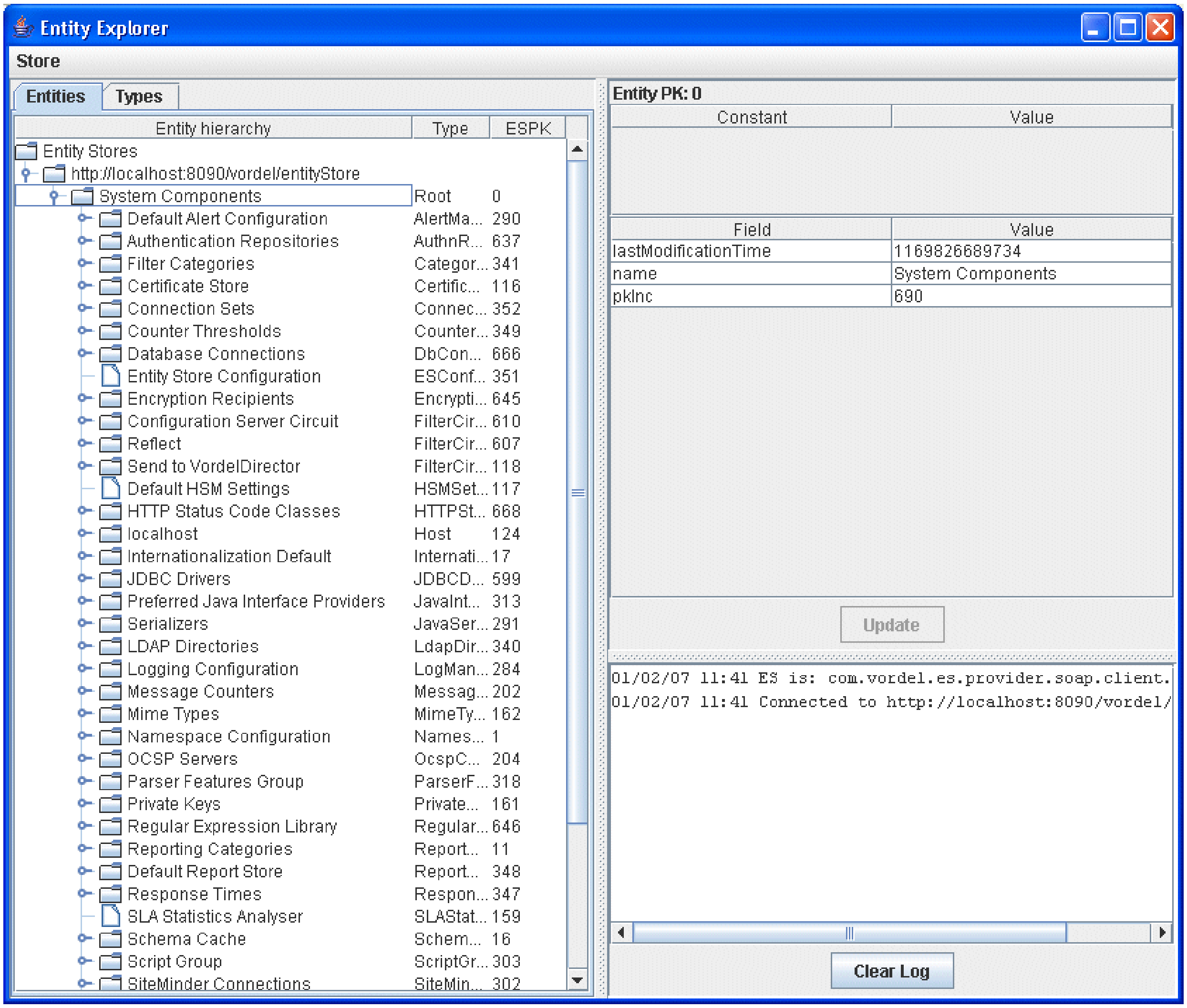
The **Connect to an Entity Store** dialog is displayed as follows:



URL http://localhost:8090/configuration/policies

Credential	Value
javax.xml.rpc.security.auth.username	admin
javax.xml.rpc.security.auth.password	changeme
timeout	

3. The API Gateway exposes a management service that interfaces to the underlying Entity Store. This is the preferred method of managing the Entity Store. You can now start the API Gateway (which connects to the Entity Store), and point the Entity Explorer at the management service exposed by the API Gateway. Start the API Gateway from the `/bin` directory of your product installation.
4. You can now configure the Entity Explorer to talk to the management service exposed by the API Gateway. By default, this service is available at the following URL, where `HOST` refers to the host name or IP address of the machine on which the API Gateway is running:
   `http://HOST:8090/configuration/policies`.
5. Enter this address in the **URL** field of the **Connect to an Entity Store** dialog.
6. If you have not already changed the default username and password for the entity store, use the default username `admin` with password `changeme`. Otherwise, specify the alternative username and password in the fields provided.
7. Click **OK** to connect to the management service on the API Gateway. A log message is displayed in the Log panel at the bottom right corner of the screen to confirm that you are connected to the API Gateway at the specified URL.
8. Expand the Entity Store filename, and then expand the **System Components** node to display the list of entity instances stored in the Entity Store:

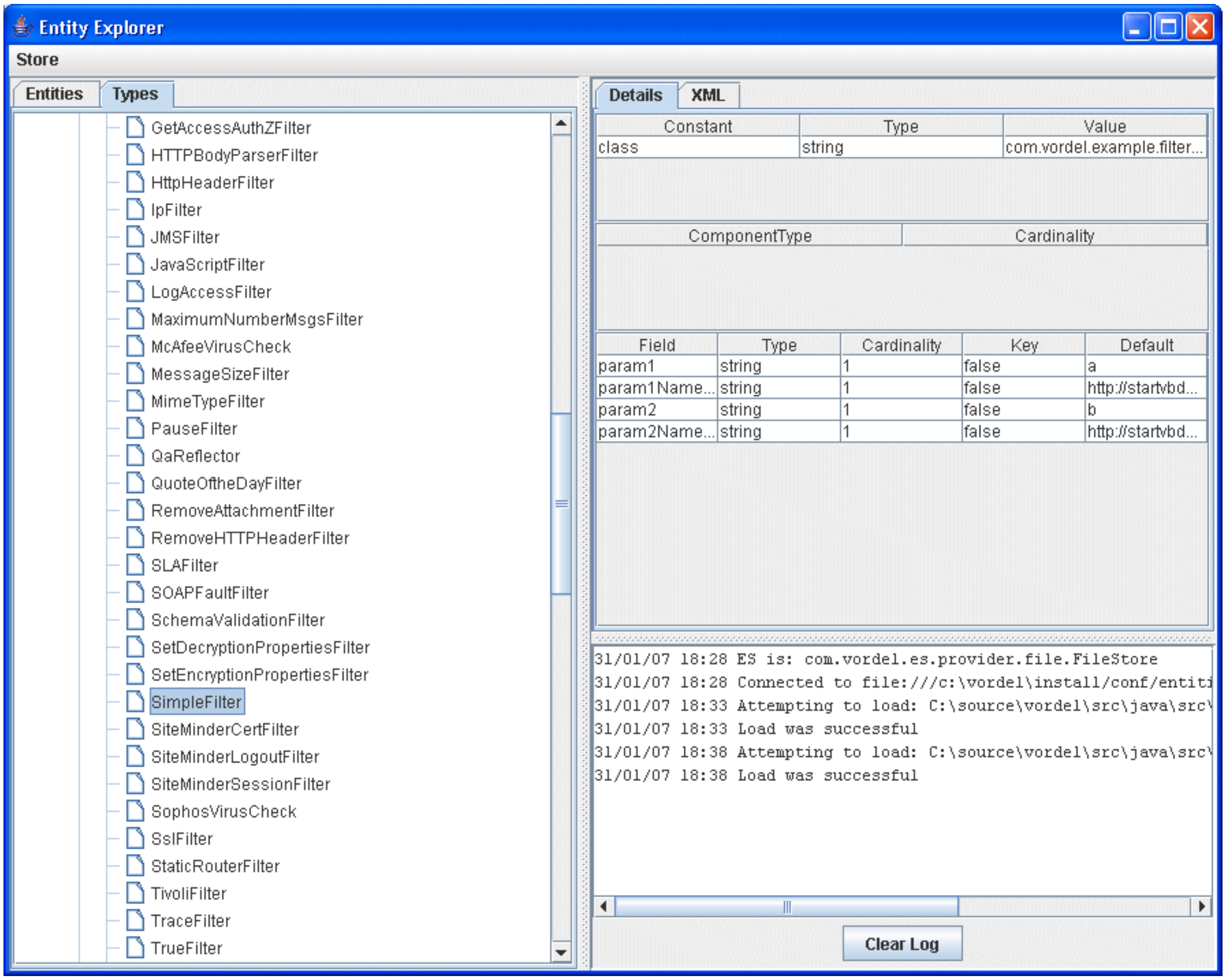


Entity hierarchy	Type	ESPK
Entity Stores		
http://localhost:8090/vordel/entityStore		
System Components	Root	0
Default Alert Configuration	AlertMa...	290
Authentication Repositories	AuthnR...	637
Filter Categories	Categor...	341
Certificate Store	Certific...	116
Connection Sets	Connec...	352
Counter Thresholds	Counter...	349
Database Connections	DbCon...	666
Entity Store Configuration	ESConf...	351
Encryption Recipients	Encrypti...	645
Configuration Server Circuit	FilterCir...	610
Reflect	FilterCir...	607
Send to VordelDirector	FilterCir...	118
Default HSM Settings	HSMSet...	117
HTTP Status Code Classes	HTTPSt...	668
localhost	Host	124
Internationalization Default	Internati...	17
JDBC Drivers	JDBCD...	599
Preferred Java Interface Providers	JavaInt...	313
Serializers	JavaSer...	291
LDAP Directories	LdapDir...	340
Logging Configuration	LogMan...	284
Message Counters	Messag...	202
Mime Types	MimeTy...	162
Namespace Configuration	Names...	1
OCSP Servers	OcspC...	204
Parser Features Group	ParserF...	318
Private Keys	Private...	161
Regular Expression Library	Regular...	646
Reporting Categories	Report...	11
Default Report Store	Report...	348
Response Times	Respon...	347
SLA Statistics Analyser	SLAStat...	159
Schema Cache	Schem...	16
Script Group	ScriptGr...	303
SiteMinder Connections	SiteMin...	302

Field	Value
lastModificationTime	1169826689734
name	System Components
pkInc	690

```
01/02/07 11:41 ES is: com.vordel.es.provider.soap.client.
01/02/07 11:41 Connected to http://localhost:8090/vordel/
```

9. Click the **Types** tab, and expand the node representing the API Gateway's management interface, (for example, `http://HOST:8090/configuration/policies`).
10. Expand the **Entity** node to display the list of registered entity types. Each of these entity types has a corresponding type definition.
11. Expand the **Filter** node, and click the **SimpleFilter** entity type in the tree. The names and data types of the fields for this entity type are displayed under the **Details** tab on the right.
12. To view the type definition for this entity, click the **XML** tab. These details should match the filter that has just been added.



- GetAccessAuthZFilter
- HTTPBodyParserFilter
- HttpHeaderFilter
- IpFilter
- JMSFilter
- JavaScriptFilter
- LogAccessFilter
- MaximumNumberMsgsFilter
- McAfeeVirusCheck
- MessageSizeFilter
- MimeTypeFilter
- PauseFilter
- QaReflector
- QuoteOftheDayFilter
- RemoveAttachmentFilter
- RemoveHTTPHeaderFilter
- SLAFilter
- SOAPFaultFilter
- SchemaValidationFilter
- SetDecryptionPropertiesFilter
- SetEncryptionPropertiesFilter
- SimpleFilter
- SiteMinderCertFilter
- SiteMinderLogoutFilter
- SiteMinderSessionFilter
- SophosVirusCheck
- SslFilter
- StaticRouterFilter
- TivoliFilter
- TraceFilter
- TrueFilter

Constant	Type	Value
class	string	com.vordel.example.filter...

ComponentType	Cardinality

Field	Type	Cardinality	Key	Default
param1	string	1	false	a
param1Name...	string	1	false	http://startvbd...
param2	string	1	false	b
param2Name...	string	1	false	http://startvbd...

```
31/01/07 18:28 ES is: com.vordel.es.provider.file.FileStore
31/01/07 18:28 Connected to file:///c:\vordel\install/conf/entiti
31/01/07 18:33 Attempting to load: C:\source\vordel\src\java\src\
31/01/07 18:33 Load was successful
31/01/07 18:38 Attempting to load: C:\source\vordel\src\java\src\
31/01/07 18:38 Load was successful
```

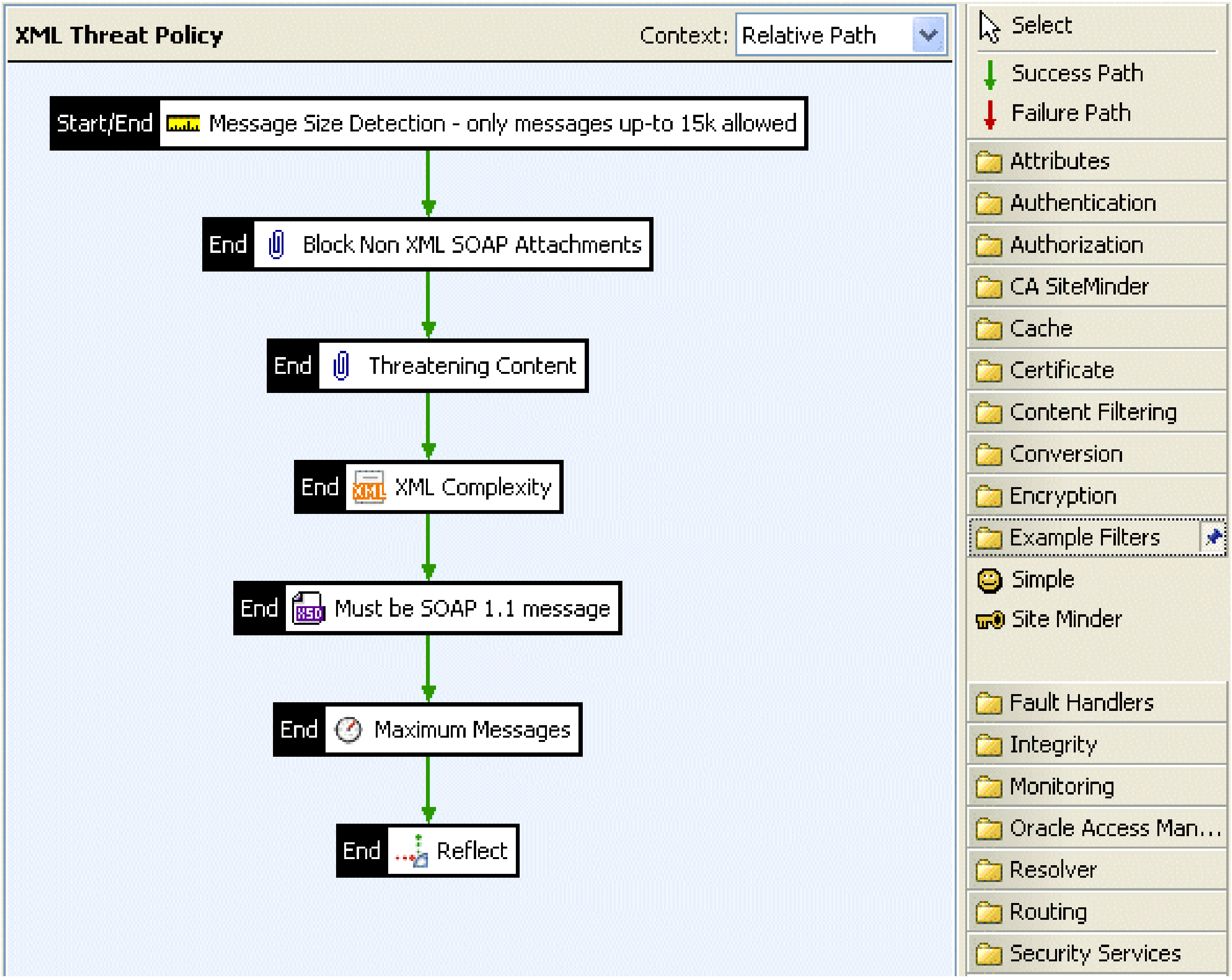
The Entity Store is now aware of the `SimpleFilter` type. You can now create an instance of a Filter class that encapsulates the fields defined in the `SimpleFilter` type.

The remaining steps in this tutorial show how to configure a policy that includes the `SimpleFilter`, and then tests its functionality.

Step 7: Construct a Policy

This section first shows how to build a simple policy that echoes messages back to the client. The next step then adds the **SimpleFilter** to the policy.

You can build policies using the policy editor in the Policy Studio. To build a policy, you can drag message filters from the filters palette on the left on to the policy canvas on the right. You can then link these filters using *Success Paths* or *Failure Paths* to create a network of filters. The following screenshot shows the policy editor screen in the Policy Studio:



The policy canvas is the large blank area on the screen, while the filters palette is the area on the right that contains the filters. Message filters are grouped together by category. For example, all the content-based filters are displayed together in one group, while all the authentication filters are displayed in a different group. You can build policies by dragging these filters and dropping them on to the canvas.

Important

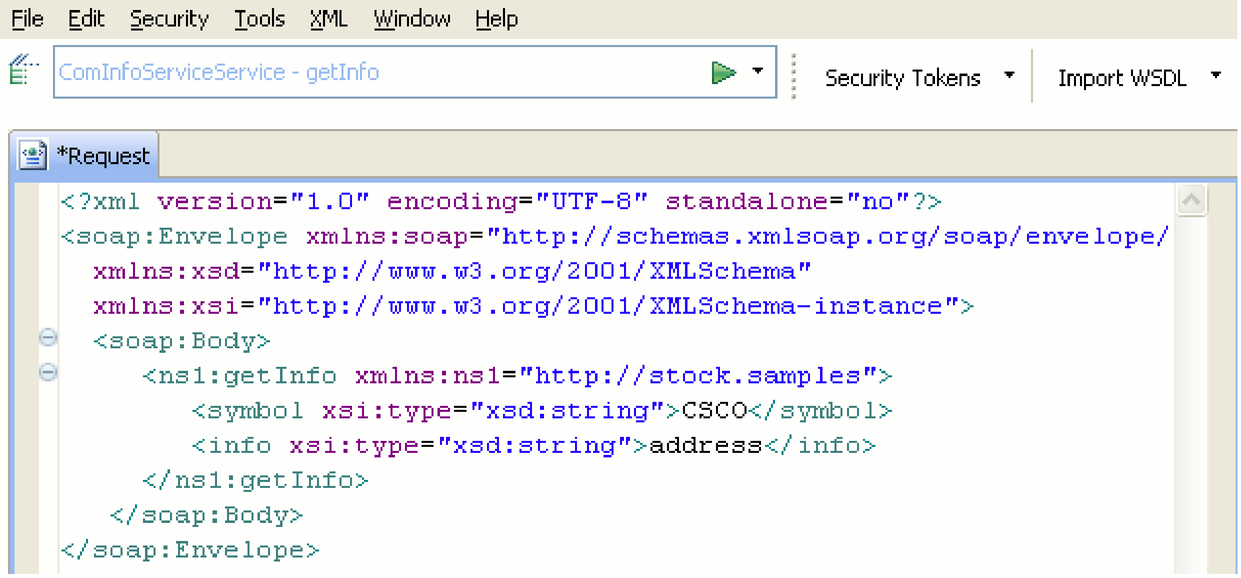
If you have followed the steps outlined above, you can see a new category of filters named **Example Filters** in the filter palette. The **Example Filters** category is expanded in the example screenshot.

Creating the Policy

To create a policy, perform the following steps:

1. Right-click the **Policies** node in the tree view on the left of the Policy Studio, and select **Add Policy**.
2. Enter **Circuit 1** as the name of the new policy in the **Policy** dialog.
3. This example creates a policy containing only one filter: the **Reflect** filter. This filter simply echoes the client message back to the client. The **Reflect** filter is found in the **Utility** filter group. Drag this filter on to the canvas.

4. Enter a name for the filter (or use the default) in the field provided. Select the default value (`200`) for the **HTTP response status code**, and click **Finish**.
5. The policy needs to have a start filter, so right-click the **Reflect** filter, and choose **Set as Start**.
6. You must now configure the Process to invoke the new policy. Under the **Listeners** node in Policy Studio, select the Process (for example **API Gateway**) -> **Default Services**. Right-click **Path: /**, and select **Edit**.
7. Enter the following values on the **Configure Relative Path** dialog:
   - **Relative Path:**
     Keep `/` in this field, meaning that the Process invokes the policy selected below for all requests received on this path.
   - **Policy:**
     Select `Circuit 1` to configure the server to send all requests received on the path configured above to our newly configured policy.
8. To force the server to pick up the new configuration, you must deploy the configuration to the server. Click the **Deploy** or press F6.
9. To test this, start up the **API Gateway Explorer** testing tool (distributed separately):



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
     <ns1:getInfo xmlns:ns1="http://stock.samples">
        <symbol xsi:type="xsd:string">CSCO</symbol>
        <info xsi:type="xsd:string">address</info>
     </ns1:getInfo>
   </soap:Body>
</soap:Envelope>
```

10. Assuming your HTTP interface is listening on port 8080, you can configure **API Gateway Explorer** to send a request to: `http://HOST:8080/`, where `HOST` is the hostname or IP address of the machine on which the server is running.

Note

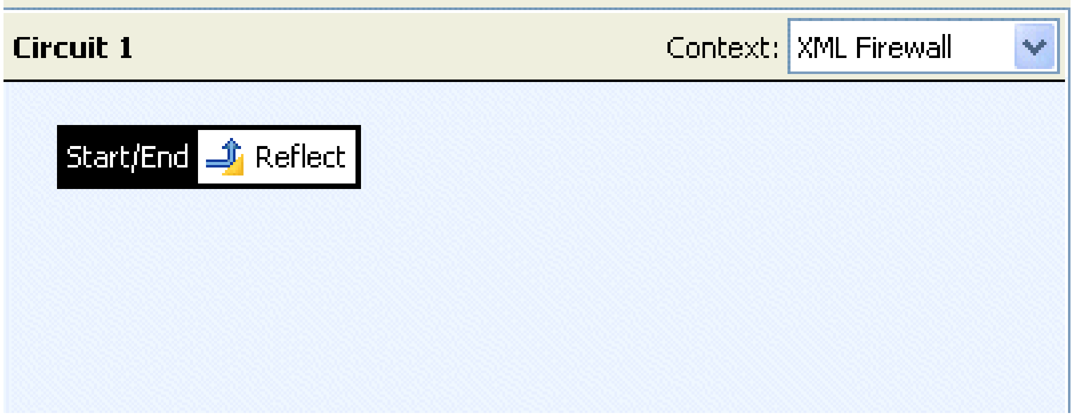
You send to `/` because, earlier, you configured the firewall to filter requests received on this relative path.

11. Copy any SOAP message into the **Request** panel of **API Gateway Explorer**. Click the triangular green **Send** button to send the message to the server, which echoes it back to the client using the **Reflect** filter configured earlier. When the message has been returned to **API Gateway Explorer**, try changing the message slightly to assure yourself that the correct message is actually being returned.

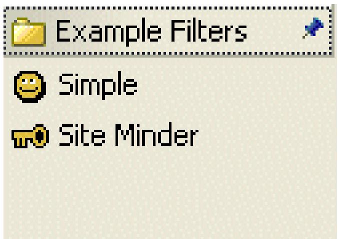
Finally, it is time to add the `SimpleFilter` to the policy, and to test its functionality.

Step 8: Configure the SimpleFilter

The final section of this tutorial adds the `SimpleFilter` to the policy built in the previous section. Currently, the policy consists of only one filter, the **Reflect** filter:



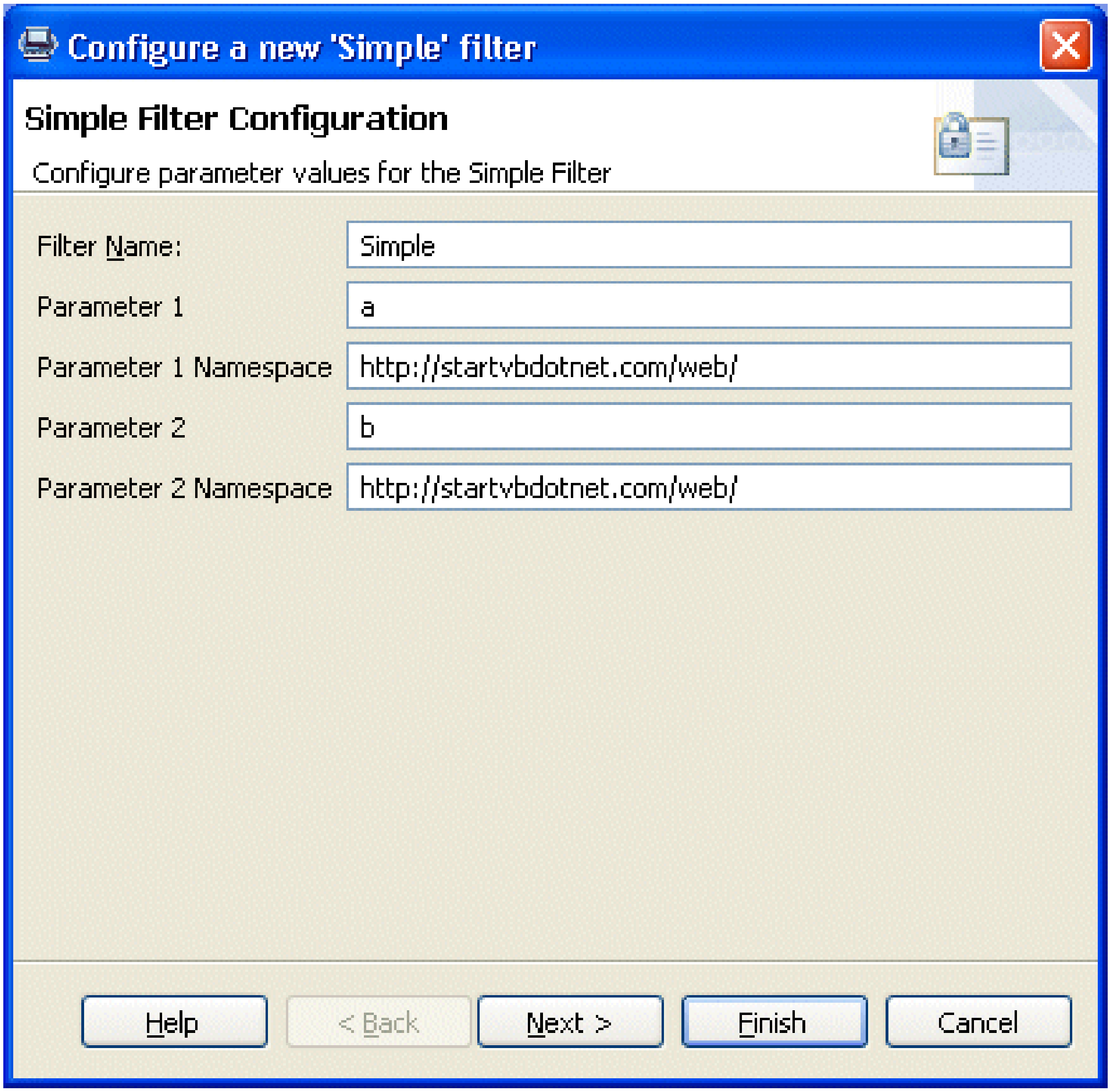
The **SimpleFilter** is found in the **Example Filters** category of the **Filter Palette** on the Policy Studio.



Configuring the Filter

To configure the new filter, perform the following steps:

1. Drag and drop the **SimpleFilter** on to the policy canvas. The configuration screen is displayed as follows:


Configure a new 'Simple' filter
Simple Filter Configuration
Configure parameter values for the Simple Filter
Help
< Back
Next >
Finish
Cancel

Filter Name: Simple

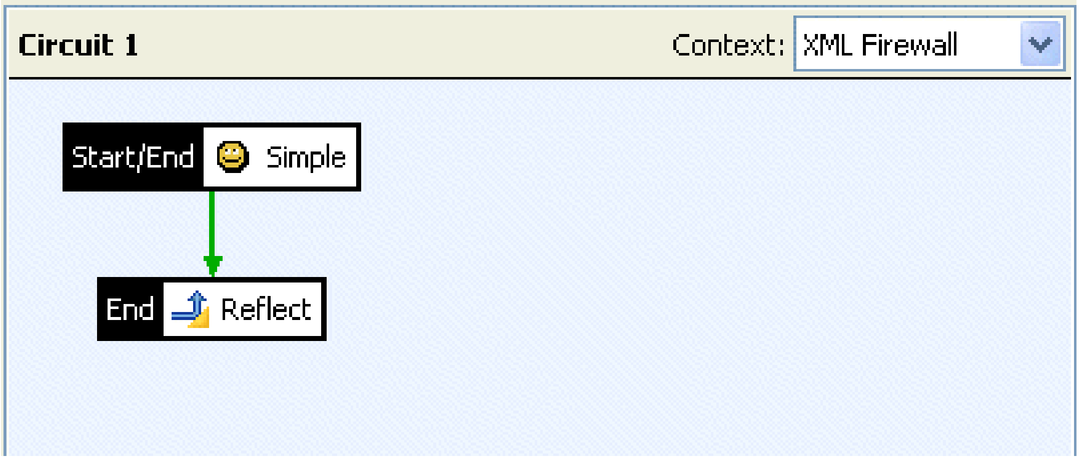
Parameter 1: a

Parameter 1 Namespace: http://startvbdotnet.com/web/

Parameter 2: b

Parameter 2 Namespace: http://startvbdotnet.com/web/

2. Because the type definition for the `SimpleFilter` entity contained default values, the input fields on the configuration screen are pre-populated with these default values.
3. Before completing the configuration, make sure the help system is working correctly. Remember that in Step 4 of this tutorial, you added a mapping to the `contexts.xml` file (in the `/plugins/com.vordel.rcp.policystudio.resources_<version>` folder of your Policy Studio installation.) If you have not done so yet, this is explained in Step 4. After restarting Policy Studio, you can try clicking the **Help** button while editing the `SimpleFilter` configuration.
4. Right-click the **Simple** node, and select **Set as Start** from the context menu.
5. Connect the **Simple** node to the **Reflect** node with a *success* path. You can do this by clicking the **Success Path** arrow, and then clicking the **Simple** node, followed by clicking the **Reflect** node. The policy is now displayed as follows:



6. To force the server to pick up the new configuration, you must refresh the server. Click the **Deploy** button in Policy Studio (or press F6).
7. You can now test the configuration to make sure that it performs as expected (that it can correctly add the two numbers together). Load the appropriate SOAP message into the **API Gateway Explorer** by selecting the **File** -> **Samples** -> **Add two numbers** menu option. The following SOAP message is loaded:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://startvbdotnet.com/web/">
      <a>1</a>
      <b>2</b>
    </Add>
  </soap:Body>
</soap:Envelope>
```

Important

Note the presence of the <a> and <b> elements in the SOAP message, and the namespace declaration in the <Add> element. These elements and their corresponding namespaces match the values configured in the **SimpleFilter** earlier.

Make sure to send the message to the same address as before by entering `http://localhost:8080/` as the URL. The **Wsdl** field is not needed and can be removed. Press the **Run** (Send) button when you have done this to send the message to the server.

The following response is returned to **API Gateway Explorer**:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance\"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <AddResponse xmlns="http://startvbdotnet.com/web/">
      <AddResult>3</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

The value of the `<AddResult>` element is `3`, which indicates that the newly added filter has worked successfully.

Conclusion

This tutorial described a working example of how to write a message processing filter using the Oracle API Gateway, and how to integrate it into a policy. You should now try to build your own filter by following a similar sequence of steps to those outlined in this tutorial.

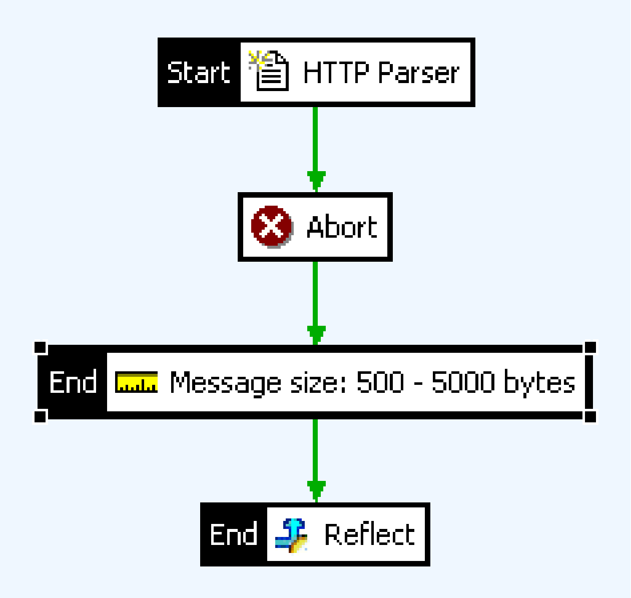
If you have any queries on the content of this document, please contact the Oracle Support Team with your questions (see *Oracle Contact Details*).

Abort Filter

Overview

The **Abort** filter can be used to force a policy to throw an exception. It can be used to test the behavior of the policy when an exception occurs.

For example, to quickly test how the policy behaves when a **Message Size** filter throws an exception, it is possible to place an **Abort** filter before it in the policy. The following policy diagram illustrates the setup:



Configuration

Enter a name for the filter in the **Name** field.

Check Group Membership

Overview

The **Check Group Membership** filter checks whether the specified API Gateway User is a member of the specified API Gateway Group. The User and the Group are both stored in the API Gateway User Store. For more details, see *API Gateway Users*.

Configuration

Configure the following required fields:

Name:
Enter an appropriate name for this filter.

User:
Enter the User name. You can specify this as a selector that expands to the value of the specified message attribute at runtime. Defaults to `${authentication.subject.id}`. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Group:
Select the Group name from the drop-down list. This list contains the Groups currently configured in the API Gateway User Store.

Possible Paths

The possible paths through this filter are as follows:

Outcome	***Description***
`True`	The specified user is a member of the specified group.
`False`	The specified user is not a member of the specified group.
`CircuitAbort`	An exception has occurred while executing the filter.

Configuration Web Service

Overview

This filter is only used by the configuration Management Service and should not need to be configured.

Copy/Modify Attributes

Overview

The **Copy/Modify Attributes** filter copies the values of message or user attributes to other message or user attributes. It is also possible to set the value of a message or user attribute to a user-specified value.

Configuration

The configured attribute-copying rules are listed in the table. To add a new rule, click the **Add** button.

The **Copy/Modify Attributes** screen can be used to copy a message or user attribute to a different message or user attribute. The **From Attribute** represents the source attribute, while the **To Attribute** represents the destination attribute.

The attribute value can be copied from 3 possible sources:

- **Message:**
  Select this option to copy the value of a message attribute. The name of the source attribute should be specified in the **Name** field.
- **User:**
  This option should be selected if a user attribute stored in the `attribute.lookup.list` is to be copied. Enter the name (and namespace if the attribute was extracted from a SAML attribute assertion) of the user attribute in the **Name** and **Namespace** fields.
  If there are multiple values stored in the `attribute.lookup.list` for the attribute entered in the **Name** field, only the first value will be copied.
- **User Entered Value:**
  Select this option to copy a user-specified value to an attribute. Enter the new attribute value in the **attribute** field. You can enter a selector to represent the value of a message attribute instead of entering a specific value directly. The syntax for entering message attribute selectors is as follows:
  `${authentication.subject.id}`
  In this case the value of the `authentication.subject.id` attribute is copied to the named attribute.

The message can be copied to one of the following types of attributes:

- **Message:**
  The attribute can be copied to any message attribute. The name of the attribute should be specified in the **Name** field.
- **User:**
  Select this option if the attribute or value should be copied to a user attribute stored in the `attribute.lookup.list`. Specify the name and namespace (if necessary) of this attribute in the **Name** and **Namespace** fields.
  If there are multiple values stored in the `attribute.lookup.list` for the attribute entered in the **Name** field of the **From attribute** section, the attribute value will be copied to the first occurrence of the attribute name in list.

Select the **Create list attribute** checkbox if the new attribute can contain several items.

Evaluate Expression

Overview

The **Evaluate Expression** filter enables you to extract a string, convert it into a selector expression, and evaluate the contents of that expression. This filter is useful for testing purposes.

For example, you could configure an **Evaluate Expression** filter with the following default value:

```
${http.client.cgiArgument('expr')}
```

You could then chain this filter to a **Set Message** filter that specifies the `${value}` message attribute generated by the **Evaluate Expression** filter in its message body. Then in this case, the following HTTP client request would result in a value of 3 in the response message body:

```
http://localhost:8080/req?expr=${1+2}
```

For more details, see the following topics:

- *Set Message*
- *Selecting Configuration Values at Runtime*

Configuration

Name:
Enter a descriptive name for this filter.

Expression location:
Enter the selector expression to be evaluated. Defaults to the following:

```
${http.client.cgiArgument('expr')}
```

Expression type:
Enter the type of the selector expression to be evaluated. Defaults to `java.lang.String`.

Execute External Process

Overview

This filter enables you to execute an external process from a policy. You can use this filter to execute any external process (for example, start an SSH session to connect to another machine, run a script, or send an SMS message).

Configuration

To configure the **Execute Process** filter, specify the following fields:

Name:
Name of the filter to be displayed in a policy. Defaults to **Execute process**.

Command tab
This tab includes the following fields:

Command to execute	Specify the full path to the command that you wish to execute (for example, `c:\cygwin\bin\mkdir.exe`).
Arguments	Click the **Add** button to add arguments to your command. Specify an argument in the **Value** field (for example, `dir1`), and click **OK**. Repeat these steps to add multiple arguments (for example, `dir2` and `dir3`).
Working directory	Specify the directory to run the command from. You can specify this using a selector that is expanded to the specified value at runtime. Defaults to `${environment.VINSTDIR}`, where `VINSTDIR` is the root of your installation. For more details on selectors, see *Selecting Configuration Values at Runtime*.
Expected exit code	Specify the expected exit code for the process when it has finished. Defaults to `0`.
Kill if running longer than (ms)	Specify the number of milliseconds after which the running process is killed. Defaults to `60000`.

Advanced tab
This tab includes the following fields:

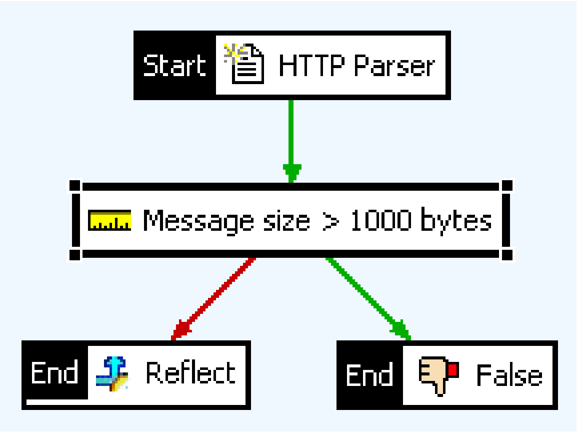
Environment variables to set	Click **Add** to add environment variables. In the dialog, specify an **Environment variable name** (for example, `JAVA_HOME`) and a **Value** (for example, `c:\jdk1.6.0_18`), and click **OK**. Repeat to add multiple variables.
Block till process finished	Select whether to block until the process is finished in the checkbox. This is enabled by default.

False Filter

Overview

The **False** filter can be used to force a path in the policy to return false. This can be useful in cases where you want to create a *false positive* path in a policy.

The following policy parses the HTTP request and then runs a **Message Size** filter on the message to make sure that the message is no larger than 1000 bytes. If we want to make sure that the message cannot be greater than this size, we can connect a **False** filter to the *success* path of the **Message Size** filter. This means that an exception will be raised if a message exceeds 1000 bytes in size.



Configuration

Enter a name for the filter in the **Name** field.

HTTP Parser

Overview

The **HTTP Parser** parses the HTTP request headers and body. As such, it acts as a barrier in the policy to guarantee that the entire content has been received before any other filters are invoked. It requires the `content.body` attribute.

The **HTTP Parser** filter forces the server to do "store-and-forward" routing instead of the default "cut-through" routing, where the request is only parsed on-demand. This filter can be used as a simple test to ensure that the message is XML, for example.

Configuration

Enter a name for the filter in the **Name** field.

Insert BST

Overview

You can use the **Insert BST** filter to insert a Binary Security Token (BST) into a message. A BST is a security token that is in binary form, and therefore not necessarily human readable. For example, an X.509 certificate is a binary security token. Inserting a BST into a message is normally performed as a side effect of signing or encrypting a message. However, there are also some scenarios where you may wish to insert a certificate into a message in a BST without signing or encrypting the message.

For example, you can use the **Insert BST** filter when the API Gateway is acting as a client to a Security Token Service that issues security tokens (for example, to create `OnBehalfOf` tokens). For more details, see the topic on the *Security Token Service Client* filter. Finally, you can also use the **Insert BST** filter to generate XML nodes without inserting them into the message. In this case, the **WS-Security Actor** is set to blank.

Configuration

You can configure the following settings on the filter dialog:

Name:
Enter an appropriate name for this filter.

WS-Security Actor:
Select or enter the WS-Security element in which to place the BST. Defaults to `Current actor / role only`. If you wish to use the **Insert BST** filter to generate XML nodes without inserting them into the message, you must ensure that this field is set to blank.

Message Attribute:
Select or enter the message attribute that contains the BST. The message attribute type can be `byte[]`, `String`, `X509Certificate`, or `X509Certificate[]`.

Value Type:
Select the BST value type, or enter a custom type. Example value types include the following:

- `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3`
- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`
- `http://xmlns.oracle.com/am/2010/11/token/session-propagation`

Base64 Encode:
This option applies only when the data in the message attribute is not already Base64 encoded. In some cases, the input may already be Base64 encoded, so you should deselect this setting in these cases.

Invoke Policy per Message Body

Overview

In cases where API Gateway receives a multipart related MIME message, the **Invoke Policy per Message Body** filter can be used to pass each body part to a specified policy for processing.

So, for example, if other XML documents have been attached to an XML message (using the SOAP with Attachments specification perhaps), each of these documents can be passed to an appropriate policy where they can be processed by the full compliment of message filters.

Configuration

Complete the following fields:

Name:
Enter a name for the filter in this field.

Policy Shortcut:
Select the policy to invoke for each MIME body part in the message. Each body part will be passed to the selected policy in turn. The filter will fail if the selected policy fails for *any* of the passed body parts.

Maximum Level to Unzip:
In cases where a MIME body part is a MIME message itself, which may, in turn, contain more multi-part messages, the setting here determines how many levels of enveloped MIME messages to attempt to unzip. A default value of "2" levels ensures that the server will not attempt to unwrap unnecessarily *deep* MIME messages.

If one of the body parts is actually an archive file (e.g. tar or zip), this setting determines the maximum depth of files to unzip in cases where the archive file contains other archive files, which may contain others, and so on.

Locate XML Nodes

Overview

You can use the **Locate XML Nodes** filter to select a number of nodes from an XML message. The selected nodes are stored in a message attribute, which is typically used by a Signature or XML Encryption filter later in a policy.

The primary use of the **Locate XML Nodes** filter is when a series of policies is auto-generated by importing a Web Services Description Language (WSDL) file that contains WS-Policy assertions. For example, because there may be many different WS-Policy assertions that describe elements in the message that must be signed, the **Locate XML Nodes** filter can be used to build up the node list of elements. Eventually, this node list is passed into the **Sign Message** filter (using a message attribute) so that a single Signature can be created that covers all the relevant parts.

However, you can also use this filter in similar cases where the message content that must be signed depends on content of the message. For example, a given policy runs a number of XPath expressions on a message where each XPath expression checks for a particular element. If that element is found, it can be marked as an element to be signed/encrypted by selecting that element in the **Locate XML Nodes** filter. This means that only a single Signature/XML Encryption filter must be configured, with each path feeding back into this filter and passing in the message attribute that contains the nodes set for each specific case.

Configuration

As explained earlier, nodes can be selected using any combination of **Node Locations**, **XPaths**, and/or **Message Attributes**. The following sections explain how to use each different mechanism and how to store the selected nodes in a message attribute.

Node Locations:
The simplest way to select nodes is using the pre-configured elements listed in the table on the **Node Locations** tab. The table is pre-populated with elements that are typically found in secured SOAP messages, including, the SOAP Body, WSSE Security Header, WS-Addressing headers, SAML Assertions, WS UsernameToken, and so on.

The elements selected here are found by traversing the SOAP message as a DOM and finding the element name with the correct namespace and with the selected index position (for example, the first `Signature` element from the `http://www.w3.org/2000/09/xmldsig#` namespace).

You can select the checkbox in the **Name** column of the table to select the corresponding node. You can select any number of **Node Locations** in this manner.

If you want to locate an element that is not already present in the table, you can add a new Node Location by clicking the **Add** button below the table. In the **Locate XML Nodes** dialog, enter the name of the element, its namespace, and its position in the message using the **Element Name**, **Namespace**, and **Index** fields.

If you wish to select this node for encryption purposes, you must select an appropriate **Encryption Type**. For example, WS-Security Policy mandates that when encrypting the SOAP Body that only its contents are encrypted and not the SOAP Body element itself. This means that the `<xenc:EncryptedData>` is inserted as a direct child of the SOAP Body element. In this case, you should select the **Encrypt Node Content** radio button.

However, in most other cases, it is typically the entire node that gets encrypted. For example, when encrypting a `<wsse:UsernameToken>`, the entire node should be encrypted. In this case, the `<EncryptedData>` element replaces the `<UsernameToken>` element. To encrypt the entire node in this manner, select the **Encrypt Node** radio button.

XPath Expressions:
In cases where you want to select nodes that exist under a more complicated element hierarchy, it may be necessary to use an XPath expression to locate the required nodes. The **XPaths** table is pre-populated with a number of XPath expressions to locate SOAP elements and common security elements, including SAML Assertions and SAMLP Responses.

To select an existing XPath expression, you can select the checkbox next to the **Name** of the appropriate XPath expression. You can select any number of XPath expressions in this manner.

To add a new XPath expression, click the **Add** button. You must enter a name for the XPath expression in the **Name** field. You can then enter the XPath expression in the **XPath Expression** field. For more information on configuring this dialog, see the *Configuring XPath Expressions* topic.

If you wish to select this node for encryption purposes, you must select an appropriate **Encryption Type**. For example, WS-Security Policy mandates that when encrypting the SOAP Body that only its contents are encrypted and not the SOAP Body element itself. This means that the `<xenc:EncryptedData>` is inserted as a direct child of the SOAP Body element. In this case, you should select the **Encrypt Node Content** radio button.

However, in most other cases, it is typically the entire node that gets encrypted. For example, when encrypting a `<wsse:UsernameToken>`, the entire node should be encrypted. In this case, the `<EncryptedData>` element replaces the `<UsernameToken>` element. To encrypt the entire node in this manner, select the **Encrypt Node** radio button.

Message Attribute:
Finally, you can also retrieve nodes that have been previously stored in a named message attribute. In such cases, another filter extracts nodes from the message and stores them in a named message attribute (for example, `node.list`). The **Locate XML Nodes** filter can then extract these nodes and store them in the message attribute configured in the **Message Attribute Name** field below.

Extract nodes from Selector Expression:
Specify whether to extract nodes from a specified selector expression (for example, `${node.list}`). This setting is not selected by default. Using a selector enables settings to be evaluated and expanded at runtime based on metadata (for example, in a message attribute, Key Property Store (KPS), or environment variable). For more details, see *Selecting Configuration Values at Runtime*.

Message Attribute in which to place list of nodes:
At runtime, the **Locate XML Nodes** filter locates and extracts the selected nodes from the message. It then stores them in the specified message attribute. For example, if you wish to sign the selected nodes, it would make sense to store the nodes in a message attribute called `sign.nodeList`, which would then be specified in the **Sign Message** filter. Alternatively, if you wish to encrypt the selected nodes, you could store the nodes in the `encrypt.nodeList` message attribute, which would then be specified in the **XML Encryption Properties** filter. The **Message Attribute Name** setting defaults to the `node.list` attribute.

Finally, you must specify whether you want the selected nodes to **Overwrite** any nodes that may already exist in the specified attribute, or if you want to **Append** to any existing nodes. You can also decide to **Reset** the contents of the message attribute. Select the appropriate radio button depending on your requirements.

Pause Filter

Overview

The **Pause** filter is mainly used for testing purposes. A **Pause** filter causes a policy to sleep for a specified amount of time.

Configuration

Enter an appropriate name for the filter in the **Name** field. When the filter is executed in a policy, it sleeps for the time specified in the **Pause for** field. The sleep time is specified in milliseconds.

Policy Shortcut

Overview

The **Policy Shortcut** filter enables you to reuse the functionality of one policy in another policy. For example, you could create a policy called **Security Tokens** that inserts various security tokens into the message. You can then create a policy that calls this policy using a **Policy Shortcut** filter.

In this way, you can adopt a design pattern of building up reusable pieces of functionality in separate policies, and then bringing them together when required using a **Policy Shortcut** filter. For example, you can create modular reusable policies to perform specific tasks, such as authentication, content-filtering, or logging, and call them as required using a **Policy Shortcut** filter.

For details on how to create a sequence of policy shortcuts in a single policy, see the *Policy Shortcut Chain* filter.

Configuration

Complete the following fields to configure the **Policy Shortcut** filter:

Name:
Enter an appropriate name for the filter.

Policy Shortcut:
Select the policy that you want to reuse from the tree. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically. The policy in which this **Policy Shortcut** filter is configured calls the selected policy when it is executed.

Tip

Alternatively, to speed up policy shortcut configuration, you can drag a policy from the tree on the left of the Policy Studio and drop it on to the policy canvas on the right. This automatically configures the fields for the selected policy.

Policy Shortcut Chain

Overview

The **Policy Shortcut Chain** filter enables you to run a series of configured policies in sequence without needing to wire up a policy containing several **Policy Shortcut** filters. This enables you to adopt a design pattern of creating modular re-usable policies to perform specific tasks, such as authentication, content-filtering, or logging. You can then link these policies together into a single, coherent sequence using this filter.

Each policy in the **Policy Shortcut Chain** is evaluated in succession. The evaluation proceeds as each policy in the chain passes, until finally the filter exits with a pass status. If a policy in the chain fails, the entire **Policy Shortcut Chain** filter also fails at that point.

The **Policy Shortcut Chain** is available from the **Utility** category of filters. You can drag and drop this filter from the filter palette to the policy editor canvas in the Policy Studio.

General Configuration

Complete the following general setting:

Name:
Enter an intuitive name for the filter in this field. For example, the name might reflect the business logic of the policies that are chained together in this filter.

Add a Policy Shortcut

Click the **Add** button to display the **Policy Shortcut Editor** dialog, which enables you to add a policy shortcut to the chain. Complete the following settings in this dialog:

Shortcut Label:
Enter an appropriate name for this policy shortcut.

Evaluate this shortcut when executing the chain:
Select whether to evaluate this policy shortcut when executing a policy shortcut chain. When this option is selected, the policy shortcut has an **Active** status in the table view of the policy shortcut chain. This option is selected by default.

Choose a specific Policy to execute:
Select this option if you wish to choose a specific policy to execute. This option is selected by default.

Policy:
Click the browse button next to the **Policy** field, and select a policy to reuse from the tree (for example, **Health Check**). You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically. The policy in which this **Policy Shortcut Chain** filter is configured calls the selected policy when it is executed.

Choose a Policy to execute by label:
Select this option if you wish to choose a policy to execute based on a specific policy label. For example, this enables you to use the same policy on all requests or responses, and also enables you to update the assigned policy without needing to rewire any existing policies. For more details, see the *Configuring Global Policies* topic.

Policy Label:
Click the browse button next to the **Policy Label** field, and select a policy label to reuse from the tree (for example, **API Gateway request policy (Health Check)**). The policy in which this **Policy Shortcut Chain** filter is configured calls the selected policy label when it is executed.

Click **OK** when finished. You can click **Add** and repeat as necessary to add more policy shortcuts to the chain. You can alter the sequence in which the policies are executed by selecting a policy in the table and clicking the **Up** and **Down** buttons on the right. The policies are executed in the order in which they are listed in the table.

Edit a Policy Shortcut

Select an existing policy shortcut, and click the **Edit** button to display the **Policy Shortcut Editor** dialog. Complete the following settings in this dialog:

Shortcut Label:
Enter an appropriate name for this policy shortcut.

Evaluate this shortcut when executing the chain:
Select whether to evaluate this policy shortcut when executing a policy shortcut chain. When this option is selected, the policy shortcut has an **Active** status in the table view of the policy shortcut chain.

Policy / Policy Label:
Click the browse button next to the **Policy** or **Policy Label** field (depending on whether you chose a specific policy or a policy label when creating the policy shortcut). Select a policy or policy label to reuse from the tree (for example, **Health Check** or **API Gateway request policy (Health Check)**). The policy in which this **Policy Shortcut Chain** filter is configured calls the selected policy or policy label when it is executed.

Quote of the Day

Overview

The **Quote of the Day** filter is a useful test utility for returning a simple SOAP response to a client. The API Gateway wraps the quote in a SOAP response, which can then be returned to the client.

Configuration

Simply enter the quote in the **Quotes** text area. This quote can be returned in a SOAP response to the client by setting the **Reflect** filter to be the successor of this filter in the policy.

The **Quote of the Day** filter can also load a file containing a list of quotes at runtime. In this case, a random quote from the file will be returned to the client in the SOAP response. Each quote should be delimited by a `%` character on a new line. This is analogous to the *BSD fortune format*. The format of this file is shown in the following example:

```
Most powerful is he who has himself in his own power.
%
All science is either physics or stamp collecting.
%
A cynic is a man who knows the price of everything and the value of nothing.
%
Intellectuals solve problems; geniuses prevent them.
%
If you can't explain it simply, you don't understand it well enough.
```

The quotes can, of course, be simply entered in this format into the **Quotes** textarea to achieve the same goal.

The following example shows a SOAP response returned by the API Gateway to a client who requested the **Quote of the Day** service:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header/>
  <s:Body xmlns:oracle="www.oracle.com">
    <oracle:getQuoteResponse>
      Every cloud has a silver lining
    </oracle:getQuoteResponse>
  </s:Body>
</s:Envelope>
```

Reflect Message Filter

Overview

The **Reflect Message** filter echoes the HTTP request headers, body, and attachments back to the client.

Configuration

Enter a name for the filter in the **Name** field. Specify an HTTP status response code to return to the client in the **HTTP Response Code Status** field.

Reflect Message And Attributes Filter

Overview

The **Reflect Message and Attributes** filter echoes the HTTP request headers, body, and attachments back to the client. It also echoes back the message attributes that were stored in the message at the time when the message completed the policy.

Configuration

Enter a name for the filter in the **Name** field.

Remove Attribute

Overview

You can use this filter to remove a specified message attribute from a request message or a response message, depending on where the filter is placed in the policy.

Configuration

Name:
Enter a suitable name for this filter.

Attribute Name:
Select or enter the message attribute name to be removed from the message (for example, `authentication.subject.password`).

Set Response Status

Overview

The **Set Response Status** filter is used to explicitly set the response status of a call. This status is then recorded as a message metric for use in reporting.

This filter is primarily used in cases where the fault handler for a policy is a **Policy Shortcut** filter. If the **Policy Shortcut** passes, the overall `fail` status still exists. The **Set Response Status** filter can then be used to explicitly set the response status back to `pass`, if necessary.

Note

This filter should only be used under advice from the Oracle Support team.

Configuration

Name:
Enter an intuitive name for the filter in this field.

Response Status:
Select **Pass** or **Fail** to set the response status.

Set Attribute

Overview

The simple **Set Attribute** filter allows you to set the value of a message attribute.

Configuration

Complete the following fields to configure the **Set Attribute** filter.

- **Name:**
  Enter a name for the filter in the **Name** field.
- **Attribute Name:**
  Enter the name of the message attribute in which you want to store a value.
- **Attribute Value:**
  Enter the value of the message attribute specified above.

String Replace Filter

Overview

The **String Replace** filter enables you to replace all or part of the value of a specified message attribute. You can use this filter to replace any specified string or substring in a message attribute. For example, changing the `from` attribute in an email, or changing all or part of a URL.

Configuration

To configure the **String Replace** filter, specify the following fields:

Name	Name of the filter to be displayed in a policy. Defaults to **String Replace**. This field is required.
Message Attribute	Select the name of the message attribute to be replaced from the list. This is required. If this is not specified, a `MissingPropertyException` is thrown, which results in a `CircuitAbortException`.
Specify Destination Attribute	By default, the value of the specified **Message Attribute** is both the source and destination, and is therefore overwritten. If you wish to specify a different destination attribute, select this checkbox to enable the **Destination Attribute** field, and select a value from the drop-down list.
Replacement String	The string used to replace the value of the specified source attribute. You can specify this as a selector, which is expanded to the specified value at runtime (for example, `${http.request.uri}`). This is a required field if you specify the **Specify Destination Attribute**.
Straight	A match string used to search the value of the specified source attribute. You can specify this as a selector, which is expanded to the specified value at runtime. If a straight (exact) match is found, it is replaced with the specified **Replacement String**.
Regexp	A match string, specified as a regular expression, used to search the value of the specified source attribute. You can specify this as a selector, which is expanded to the specified attribute value at runtime. If a match is found, it is replaced with the specified **Replacement String**. For more details on selectors, see *Selecting Configuration Values at Runtime*.
First Match	If a match is found, only replace the first occurrence.
All Matches	If a match is found, replace all occurrences.

Note

The possible paths available through this filter are `True` (even if no replacement takes place), and `CircuitAbort`. Under certain circumstances, if the **Replacement String** contains a selector, a `MissingPropertyException` can occur, which results in a `CircuitAbortException`.

Switch on Attribute Value

Overview

The **Switch on Attribute Value** filter enables you to switch to a specific policy based on the value of a configured message attribute. You can specify various switch cases (for example, contains, is, ends with, matches regular expression, and so on). Specified switch cases are evaluated in succession until a switch case is found, and the policy specified for that case is then executed. You can also specify a default policy, which is executed when none of the switch cases specified in the filter are found.

The **Switch on Attribute Value** filter is available from the **Utility** category of filters. You can drag and drop this filter from the filter palette on to the policy editor canvas in the Policy Studio, and configure it to meet the requirements of your policy.

Configuration

Complete the following configuration settings in the **Switch on attribute value** screen:

Name:
Enter an intuitive name for the filter. For example, the name might reflect the business logic of a specified switch case.

Switch on selector expression:
Enter or select the name of the message attribute selector to switch on (for example, `${http.request.path}`). This filter examines the specified message attribute value, and switches to the specified policy if this value meets a configured switch case.

Case:
You can add, edit, and delete switch cases by clicking the appropriate button on the right. All configured switch cases are displayed in the table on this screen. For more details, see Adding a Switch Case.

Default:
This field specifies the default behavior of the filter when none of the specified switch cases are found in the configured message attribute value. Select one of the following options:

Return result of calling the following policy	Click the browse button, and select a default policy to execute from the dialog (for example, **XML Threat Policy**). The filter returns the result of the specified policy. This option is selected by default.
Return true	The filter returns true.
Return false	The filter returns false.

Adding a Switch Case

To add a switch case, click the **Add** button, and configure the following fields in the dialog:

Comparison Type:
Select the comparison type that you wish to perform with the configured message attribute. The available options include the following:

- `Contains`
- `Doesn't Contain`
- `Ends With`

- `Is`
- `Isn't`
- `Matches Regular Expression`
- `Starts With`

All of these options are case insensitive, except for `Matches Regular Expression`.

Compare with:
Enter the value to compare the configured message attribute value with. For example, if you select a **Comparison Type** of `Matches Regular Expression`, enter the regular expression in this field.

Policy:
Click the browse button next to the **Policy** field, and select the policy to execute from the dialog (for example, **Remove All Security Tokens**). You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically. The selected policy is executed when this switch case is found.

Click **OK** when finished. You can click **Add**, and repeat as necessary to add more switch cases to this filter. The switch cases are examined in the order in which they are listed in the table. You can alter the sequence in which the switch cases are evaluated by selecting a policy in the table and clicking the **Up** and **Down** buttons on the right.

Time Filter

Overview

The **Time Filter** enables you to block or allow messages on a specified time of day and/or day of week. You can input the time of day directly in the **Time Filter** screen, configure message attributes to supply this information using the Java `SimpleDateFormat`, or specify a cron expression.

You can use the **Time Filter** in any policy (for example, to block messages at specified times and/or days when a Web Service is not available, or has not been subscribed for by a consumer). In this way, this filter enables you to meter the availability of a Web Service and to enforce Service Level Agreements.

General Configuration

Configure the following general options:

Name:
Enter an appropriate name for this filter.

Block Messages:
Select this option if you wish to use this filter to block messages. This is the default option.

Allow Messages:
Select this option if you wish to use this filter to allow messages.

Basic Time Options

Select **Basic** if you wish to block or allow messages at specified times of the day. This is the default option. You can configure following settings:

User defined time:
Select this option to input the times to block or allow messages directly in this screen. This is the default option. Configure the following settings:

From	The time to start blocking or allowing messages from in hours, minutes, and seconds. Defaults to `9:00:00`.
To	The time to end blocking or allowing messages in hours, minutes, and seconds. Defaults to `17:00:00`.

Time from attribute:
Select this option to specify times to block or allow messages using configured message attributes. You can specify these attributes using selectors, which are replaced at runtime with the values of the specified message attributes set in previous filters or messages. For more details, see *Selecting Configuration Values at Runtime*. You must configure the following settings:

From	Message attribute that contains the time to start blocking or allowing messages from (for example, `$(message.starttime)`). Defaults to a time of `9:00:00`.
To	Message attribute that contains the time to end blocking or allowing messages (for example, `$(message.endtime)`). Defaults to a time of `17:00:00`.

Pattern	Message attribute that contains the time format based on the Java `SimpleDateFormat` class (for example,`$(message.pattern)`). This enables you to format and parse dates in a locale-sensitive manner. Day, month, years, and milliseconds are ignored. Defaults to a format of `HH:mm:ss`.

Days:
If you wish to block or allow messages on specific days of the week, select the checkboxes for those days. For example, you may wish to block messages on Saturday and Sunday.

Advanced Time Options

Select **Advanced** if you wish to block or allow messages at specified times based on a cron expression. Configure the following setting:

Cron Expression:
Enter a cron expression or a message attribute that contains a cron expression in this field. Alternatively, click the button next to this field to use a the **Cron Dialog** to guide you through the configuration steps. You can also use this dialog to test the cron expression. For details, see the topic on *Configuring Cron Expressions*.

For example, the following cron expression blocks all messages received on April 27 and 28 2012, at any time except those received between 10:00:01 and 10:59:59.

```
* * 0-9,11-23 27-28 APR ? 2012
```

The default value is `* * 9-17 * * ? *`, which specifies a time of 9:00:00 to 17:00:00 every day. For more details on cron expressions, see the *Policy Execution Scheduling* topic.

Trace Filter

Overview

The **Trace** filter outputs the current message attributes to the configured trace destination(s). By default, output is traced to the system console.

Configuration

Name:
Enter an appropriate name for the filter.

Include the following text in trace:
Enter an optional custom text message to include in the trace output.

Trace Level:
Select the level at which you wish to trace output from the drop-down list. `DATA` tracing is the most verbose level, while `FATAL` is the least verbose.

Include Attributes:
Select this option to trace all current message attributes to the configured trace destination.

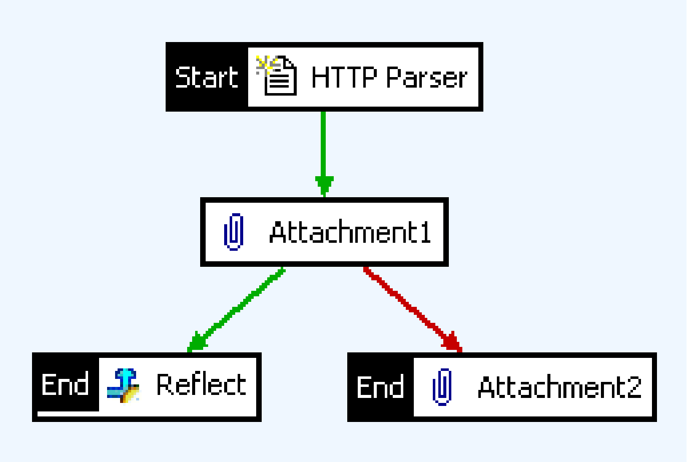
Include Body:
Select this option if you wish to trace the entire message body.

Indent XML:
If this option is selected, the XML message is pretty-printed (indented) before it is output to the trace destination.

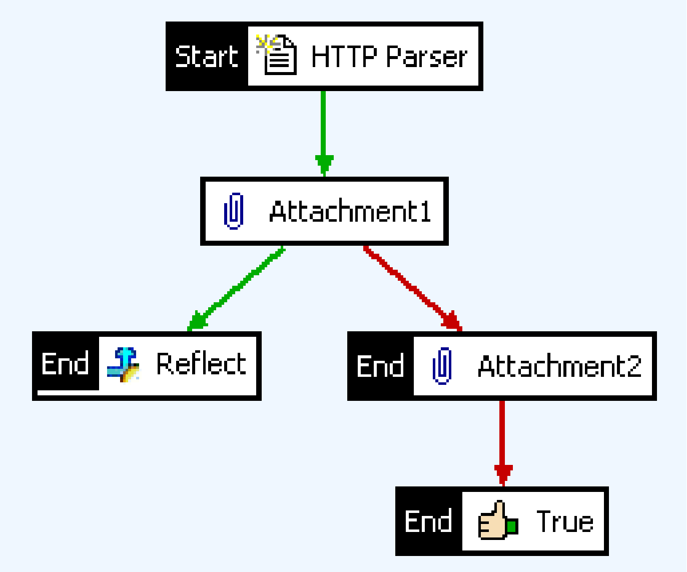
True Filter

Overview

You can use the **True** filter to force a path in a policy to return true. For example, this can be useful in cases where you want to prevent a path from ending on a false case and consequently throwing an exception. The following policy parses the HTTP request, and then runs **Attachment1** on the message. If **Attachment1** passes, the message is echoed back to the client by the **Reflect** filter. However, if **Attachment1** fails, the **Attachment2** filter is run on the message. Because this is an *end* node, if this filter fails, an exception is thrown.



By adding a **True** filter to the **Attachment2** filter, this path always ends on a true case, and so does not throw an exception if **Attachment2** fails.



Configuration

Enter an appropriate name for the filter in the **Name** field.

Web Service Filter

Overview

The **Web Service Filter** is used to control and validate requests to the Web Service and responses from the Web Service. Typically, this is automatically generated and populated as a **Service Handler** when a WSDL file is imported into the Web Services Repository. For example, if you import the WSDL file for a Web Service named `ExampleService`, a **Service Handler for ExampleService** filter is automatically generated. However, you can also configure a **Web Service Filter** manually.

In cases when the imported WSDL file contains WS-Policy assertions, a number of policies are automatically created to generate the filters required to generate the relevant security tokens (for example, SAML, WS-Security UsernameToken, and WS-Addressing headers). These policies perform the necessary cryptographic operations (for example, signing and encrypting) to meet the security constraints stipulated by the WS-Policy assertions.

General Settings

Name:
Enter an intuitive name for the filter in this field.

Web Service Context:
Click the button on the right, and select a WSDL file currently registered in the Web Service Repository from the tree to set the Web Service Context. To register a Web Service, right-click the default **Web Services** node, and select **Register Web Service**. For more details on adding services to the Web Services Repository, see the *Web Service Repository* tutorial.

Routing

When routing to a service, you can specify a direct connection to the Web Service endpoint by using the URL in the WSDL or you can override this URL by entering a URL in the field provided. Alternatively, in cases where the routing behavior is more complex, you can delegate to a custom routing policy, which takes care of the added complexity. The top-level radio buttons on the **Routing** tab allow for these alternative routing configurations.

Direct Connection to Service Endpoint:
Select this option to route to either the URL specified in the WSDL or a URL. The radio buttons in the **Routing Details** group enable you to choose between using the URL in the WSDL and providing an override. When providing an override, you can enter the new URL in the **URL** field. Alternatively, you can specify the URL as a selector so that the URL is built dynamically at runtime from the specified message attributes (for example `${host}:${port}`, or `${http.destination.protocol}://${http.destination.host}:${http.destination.port}`). For more details on selectors, see *Selecting Configuration Values at Runtime*.

In both cases, you can configure the connection details, such as SSL and other authentication schemes, for the direct connection using the fields in the **Connection Details** group. For more details, see the *Connection* tutorial.

Delegate to Routing Policy:
If you wish to use a dedicated routing policy to send messages on to the Web Service, you can select this radio button. For example, you may have configured a dedicated routing policy that uses the JMS-based **Messaging System** filter to route over JMS. Click the browse button next to the **Routing Policy** field. Select the policy that you want to use to route messages, and click **OK**. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically.

Validation

The WSDL for a Web Service contains information about the SOAP Action, SOAP Operation, and the data types of the message parts used in a particular SOAP operation. The API Gateway creates the following implicit validation incoming requests for the Web Service:

- **SOAPAction HTTP Header:**
  If a Web Service requires clients to send a certain SOAPAction HTTP header in all requests, the API Gateway can check the value of this header in the incoming request against the value specified in the WSDL.
- **SOAP Operation and Namespace:**
  The WSDL defines the SOAP Operation and namespace to be used in the SOAP request. The SOAP Operation is defined as the first child element of the SOAP Body element. The API Gateway can check the value of this element in an incoming SOAP request and its namespace against the values specified in the WSDL.
- **Relative Path:**
  The filter ensures that requests for this Web Service are received on the same URL as that specified in the `<service>` block of the WSDL.

It is also common for a WSDL document to contain an XML Schema that defines the format and types of the message parts in the request. This is usually the case for document/literal style SOAP requests, where a complete XML Schema is embedded or imported into the `<wsdl:types>` block of the WSDL.

When using a WSDL to import a service into the Web Services Repository, the Policy Studio can extract the XML Schema from the WSDL and configure the API Gateway to validate incoming requests against it. Select the **Use WSDL Schema** if you want to validate incoming requests against the Schema in the WSDL.

Alternatively, you can create a custom-built policy to validate the contents of incoming requests. To do this, select the **Delegate to Validation Policy** radio button, and the click the browse button next to the **Message Validation Policy** field. Select the policy that you want to use to validate requests, and click the **OK** button.

Message Interception Points

The configuration settings on the **Message Interception Points** tab determine how the request and response messages for the service are processed as they pass through the API Gateway. Several message interception points are exposed to enable you to hook into different stages of the API Gateway's request processing cycle.

At each of these interception points, it is possible to run policies that are specific to that stage of the request processing cycle. For example, you can configure a logging policy to run just before the request has been sent to the Web Service and then again just after the response has been received.

Typically, the configuration settings on this screen are automatically configured when importing a service into the Web Services Repository based on information contained in the WSDL. In cases where the WSDL contains WS-Policy assertions, a number of policies are automatically generated and hooked up to perform the relevant security operations on the message. For example, policies are created to insert SAML assertions, WS-Security Username Tokens, WS-Addressing headers, and WS-Security Timestamps into the message. Similarly, filters are created to sign and encrypt the outbound message, if necessary, and to decrypt and validate the signature on the response from the Web Service.

Order of Execution:
The order of execution of the message interception points is as follows:

- The interception points are executed in the following order:
  1. Request from Client
  2. User-defined Request Hooks
  3. Request to Service
  4. Response from Service
  5. User-defined Response Hooks
  6. Response to Client
- In steps 1, 3, 4, and 6, the execution order is as follows:
  A) Before Operation-specific Policy
  B) Operation-specific Policy Shortcuts
  C) After Operation-specific Policy
- The overall order of all the message interception points is given in the sequence below.

1. Request from Client:
This is the first message interception point, which enables you to run a policy against the request as it is received by the API Gateway. Typically, this is where authentication and authorization events should occur.

1A) Before Operation-specific Policy:
This is usually where authentication policies should be configured because it is the earliest point in the request cycle that you can hook into. To select a policy to run at this point, click the browse button, and select the checkbox next to a previously configured policy.

1B)Operation-specific Policy Shortcuts:
If you want to run policies that are specific to the different operations exposed by the Web Service, click the **Edit** button at the bottom of the table to set this up. For example, you may want to perform different validation on requests for the different operations.

On the **Policy Shortcut Editor** dialog, enter the **Operation Namespace** and **Operation Name** in the fields provided. Enter a regular expression used to match the value of the SOAPAction HTTP header in the **SOAPAction Regular Expression** field. Finally, select the policy to run requests for this operation by clicking the browse button next to the **Policy Shortcut** field. Select the checkbox next to the policy that you want to run.

1C) After Operation-specific Policy:
This enables you to run a policy on the request *after* all the operation-level policies have been executed on the request. Select the appropriate policy as described earlier by clicking the browse button.

2. User-defined Request Hooks:
Users should primarily use this interception point to hook in their own custom-built *request* processing policies.

User-defined Request Policy:
Browse to your custom-built request processing policy using the browse button as before.

3. Request to Service:
This enables you to alter the message before it is routed to the Web Service. For example, if the service requires the message to be signed and encrypted, you can configure the necessary policies here.

3A) Before Operation-specific Policy:
This enables run policies on the message *before* the operation-level policies are run. Select the policy to run as outlined in the previous sections.

3B) Operation-specific Policy Shortcuts:
Operation-level policies on the request to the Web Service can be run here. For example, if the input policy for a particular operation requires the body to be signed and encrypted, a **Locate XML Nodes** filter can be run here to mark the required nodes.

3C) After Operation-specific Policy:
This is the last interception point available before the message is routed on to the Web Service. For example, if certain operation-level policies have been run to mark parts of the message to be signed and encrypted, the signing and encrypting filters should be run here.

4. Response from Service:
This is executed on the response returned from the Web Service.

4A) Before Operation-specific Policy:
If the response from the Web Service is encrypted, this interception point enables you to decrypt the message *before* any of the operation-level policies are run on the decrypted message.

4B) Operation-specific Policy Shortcuts:
The policies configured at this point run on specific operation-level responses (for example, `getHelloResponse`) from the Web Service.

4C) After Operation-specific Policy:
This should be used to run policies *after* the operation-level policies have been run. For example, this is the appropriate point to place an XML Signature Verification filter.

5. User-defined Response Hooks:
You should primarily use this interception point to hook in custom-built *response* processing policies.

User-defined Response Policy:
Browse to your custom-built response processing policy using the browse button as before.

6. Response to Client:
This enables you to process the response before it is returned to the client.

6A) Before Operation-specific Policy:
As before, this enables you to process the message with a policy before the operation-level policies are run on the response.

6B) Operation-specific Policy Shortcuts:
The policies listed here are run on each operation response.

6C) After Operation-specific Policy:
This is the very last point at which you can run policies to process the response message before it is returned to the client. For example, if you are required to return a signed and encrypted response message to the client, the signing and encrypting should be done at this point.

WSDL

You can expose an imported WSDL file to clients of the API Gateway. A client can retrieve a WSDL for a service by appending the `WSDL` name to the query string of the relative path on which the service is accepting requests. For example, if the service is accepting requests at the URL `http://server:8080/services/getHello`, the client can retrieve the WSDL on the following URL:
`http://server:8080/services/getHello?WSDL`

Important

When the API Gateway returns the WSDL to the client, it dynamically modifies the service URL of the original WSDL to point to the machine on which the API Gateway is running. For example, the original WSDL contains the following service element, where `www.service.com` resolves to an internal IP address that is not accessible to the public Internet:

```
<wsdl:service name="GetHelloService">
  <wsdl:port name="GetHelloServiceSoap" binding="tns:ServiceSoap">
    <soap:address location="http://www.service.com/getHello"/>
  </wsdl:port>
</wsdl:service>
```

When the API Gateway returns this WSDL to the client, it dynamically modifies the value of the `location` attribute to point to the name of the machine hosting the API Gateway. In the following example, the `location` attribute has been modified to point to the API Gateway instance running on port 8080 on the `Oracle_SERVER` host:

```
<wsdl:service name="GetHelloService">
  <wsdl:port name="GetHelloServiceSoap" binding="tns:ServiceSoap">
    <soap:address location="http://Oracle_SERVER:8080/getHello"/>
  </wsdl:port>
</wsdl:service>
```

When the client receives the WSDL, it can automatically generate the SOAP request for the `getHello` service, which it then sends to the `Oracle_SERVER` machine on port 8080.

Complete the following fields if you wish to expose the WSDL for this service to clients.

Advertise WSDL to the Client:
Select this option if you wish to publish the WSDL for the selected Web Service.

Note

The exposed WSDL represents a *virtualized* view of the back-end Web Service. In this way, clients can retrieve the WSDL from the API Gateway, generate SOAP requests, and send these requests to the API Gateway. The API Gateway then routes the requests on to the Web Service.

WSDL Access Policy:
If you want to configure a policy to control or monitor access to the WSDL for this service, you can select the policy by clicking the browse button to the right of this field. Select the policy that you want to use to run on requests to retrieve the WSDL.

Monitoring

The fields on this tab enable you to configure whether this Web Service stores usage metrics data to a database. This information can then be used by API Gateway Analytics to produce reports showing how and who is calling this Web Service. The following fields are available on this tab:

- **Monitor service usage:**
  Select this option if you want to store message metrics for this Web Service.
- **Monitor service usage per client:**
  Select this option if you want to generate reports monitoring which authenticated clients are calling which Web Services.
- **Monitor client usage:**
  If you want to generate reports on authenticated clients, but are not interested in what Web Services they are calling, select this option, and deselect the **Monitoring service usage per client** checkbox.
- **Which attribute is used to identify the client?:**
  Enter the message attribute to use to identify authenticated clients. The default is `authentication.subject.id`, which stores the identifier of the authenticated user (for example, the username or user's X.509 Distinguished Name).
- **Composite Context:**
  This setting enables you to select a service context as a composite context in which multiple service contexts are monitored during the processing of a message. This setting is not selected by default.

  For example, the API Gateway receives a message, and sends it to `serviceA` first, and then to `serviceB`. Monitoring is performed separately for each service by default. However, you can set a composite service context before `serviceA` and `serviceB` that includes both services. This composite service passes if both services complete successfully, and monitoring is also performed on the composite service context.

Return WSDL

Overview

The **Return WSDL** filter returns a WSDL file from the Web Services Repository. This filter is configured automatically when auto-generating a policy from a WSDL file and is not normally manually configured.

For details on how to auto-generate a policy from a WSDL file, see the *Web Service Repository* topic. For details on how to identify services in the Web Service Repository, see the *Set Web Service Context* topic.

Configuration

Enter a name for the filter in the **Name** field.

Set Web Service Context

Overview

The **Set Web Service Context** filter is used in a policy to determine the service to obtain resources from in the Web Service Repository. For example, by pointing this filter at a pre-configured `getQuote` service in the Web Service Repository, the policy knows to return the WSDL for this particular service when a WSDL request is received. The **Return WSDL** filter is used in conjunction with this filter to achieve this.

The **Set Web Service Context** filter is configured automatically when auto-generating a policy from a WSDL file and is not normally manually configured. For a detailed example, see the *Web Service Repository* tutorial.

Configuration

Name:
Enter a name for the filter in the **Name:** field.

Service WSDL:
Click the button on the right, and select a service definition (WSDL file) currently registered in the Web Service Repository from the tree. To register a Web Service, right-click the default **Web Services** node, and select **Register Web Service**. For more details on adding services to the Web Services Repository, see the *Web Service Repository* tutorial.

Monitoring:
The fields on this tab enable you to configure whether this Web Service stores usage metrics data to a database. This information can be used by API Gateway Analytics to produce reports showing how and who is calling this Web Service. The following monitoring options are available on this tab:

- **Monitor service usage:**
  Select this option if you want to store message metrics for this Web Service.
- **Monitor service usage per client:**
  Select this option if you want to generate reports monitoring which authenticated clients are calling which Web Services.
- **Monitor client usage:**
  If you want to generate reports on authenticated clients, but are not interested in what Web Services they are calling, select this option and deselect the **Monitoring service usage per client** checkbox.
- **Which attribute is used to identify the client?:**
  Enter the message attribute to use to identify authenticated clients. The default is `authentication.subject.id`, which stores the identifier of the authenticated user (for example, the username or user's X.509 Distinguished Name).
- **Composite Context:**
  This setting enables you to select a service context as a composite context in which multiple service contexts are monitored during the processing of a message. This setting is not selected by default.

  For example, the API Gateway receives a message, and sends it to `serviceA` first, and then to `serviceB`. Monitoring is performed separately for each service by default. However, you can set a composite service context before `serviceA` and `serviceB` that includes both services. This composite service passes if both services complete successfully, and monitoring is also performed on the composite service context.

Authentication Repository

Overview

The API Gateway supports a wide range of common authentication schemes, including SSL, XML Signatures, WS-Security Username tokens, and HTTP Authentication. With SSL, the client authenticates to the API Gateway using a client certificate. With XML Signatures, the client is authenticated by validating the signature contained within the XML message. However, when the API Gateway attempts to authenticate a client using a username and password (for example, WS-Security Username tokens and HTTP Authentication), it must compare the username and password presented by the client to those stored in the Oracle **Authentication Repository**.

The **Authentication Repository** acts as a repository for **Users**. **Users** serve many roles in the API Gateway. For example, clients whose username and password combinations are stored in the **Authentication Repository** can authenticate to the API Gateway using that username and password combination. For more information on **Users**, see the *API Gateway Users* tutorial.

The **Authentication Repository** can be maintained in the API Gateway's local configuration store, in an LDAP directory, or in a range of third-party Identity Management products and services. When a user has been successfully authenticated against one of these repositories, the API Gateway can use any one of that user's stored attributes (for example, DName, email address, username) to authorize that same user in a subsequent Authorization Filter.

For example, this *credential mapping* is useful in cases where your client-base uses username-password combinations for authentication (*authentication attributes*), yet their access rights must be looked up in an authorization server using the client's DName (*authorization attribute*). In this way, the client possesses a single *virtual identity* within the API Gateway. The client can use one identity for authentication, and another for authorization, yet the API Gateway sees both identities as representing the same client.

You can add a new repository under the **External Connections** node in the Policy Studio tree by right-clicking the appropriate node (for example, **Database Repositories**), and selecting **Add a new Repository**. Similarly, you can edit an existing repository by right-clicking the repository node (for example, the default **Local User Store**), and selecting **Edit Repository**. Repositories added under the **External Connections** node are available for reuse by multiple filters.

Local Repositories

The **Authentication Repository** can be maintained in the same database that the API Gateway uses to store all its configuration information. To edit the default user store, select **Local Repositories** -> **Local User Store** -> **Edit Repository**. Alternatively, to create a new user store, select **Local Repositories** -> **Add a new Repository**.

You can enter an appropriate name for the repository in the **Repository Name** field. The **Authorization Attribute Format** field enables administrators to specify whether to use the client's **X.509 Distinguished Name** or **User Name** in subsequent Authorization Filters. If **User Name** is selected, the user name used by the client to authenticate to the API Gateway is used in any configured Authorization filters. If **X.509 Distinguished Name** is selected, the X.509 DName stored by the API Gateway for that user is used for subsequent authorization.

For example, if the administrator selects **User Name** from the **Authorization Attribute Format** drop-down list, `admin` (the **User Name** field) is used for authorization. Alternatively, if **X.509 Distinguished Name** is selected, the X.509 DName is used for authorization (for example, `O=Company, OU=comp, EMAIL=emp@company.com, CN=emp`).

For more information on adding and configuring users to the **Authentication Repository**, see the *API Gateway Users* tutorial.

LDAP Repositories

In cases where an organization stores user profiles in an LDAP directory, it does not make sense to re-enter these profiles into the default API Gateway user store. Instead, the API Gateway can leverage an existing LDAP directory by querying it for user profile data. If a user's profile can be retrieved, and you can bind to the LDAP directory as that user, the user is authenticated.

Authenticating with LDAP

The following steps occur when a filter is configured to authenticate a user against an LDAP repository using a username and password combination:

1. A pooled LDAP connection to the repository selected in the **LDAP Directory** field is retrieved.
2. A search filter is run using the retrieved connection (for example, `(&(objectClass={User})(sAMAccountName={c05vc}))`). Attributes configured in the **Login Authentication Attribute** and **Authorization Attribute** fields are retrieved in this search.
   For example, if you select `Distinguished Name` from the drop-down list, the user's DName is retrieved from the LDAP directory. This uniquely identifies the user in the LDAP directory, and is used to bind to the directory so the user's password can be verified. The attribute specified in **Login Authentication Attribute** is used when you bind as any user. The value of the attribute specified in **Authorization Attribute** is stored in `authentication.subject.id`, and can be used by subsequent filters in the policy (for example, Authorization filters that authorize the authenticated user).
3. If no results are returned from the search, the user is not found in the directory. It is important that the administrator user configured on the **Configure LDAP Server** screen has the ability to see the user that you are attempting to authenticate.
4. If multiple users are returned from the search, an attempt is made to bind to the directory using each **Login Authentication Attribute** value retrieved from the search, together with the password from the message.
5. If more than one user is authenticated correctly, an error is returned because you only want to authenticate a single user.
6. If no user is authenticated, an error is returned.
7. If a single user's **Login Authentication Attribute** value and password binds successfully to the directory, authentication has succeeded.
8. Any successful bind is immediately closed.

Creating an LDAP Repository

To create a new LDAP repository, right-click **LDAP Repositories**, and select **Add a new Repository**. The details entered on the **Authentication Repository** dialog depend on the type of LDAP directory that you are using. The Policy Studio has default entries for some of the more common LDAP directories, which are available from the drop-down lists. However, you can also connect to alternative LDAP directories.

The following subsections demonstrate how to configure this screen for typical user searches on three common LDAP servers:

- Oracle Directory Server
- Microsoft Active Directory Server
- IBM Directory Server

Oracle Directory Server

To configure the **Authentication Repository** dialog for Oracle Directory Server (formerly iPlanet and Sun Directory Server), use the following settings:

- **Repository Name:**
  Enter a suitable name for this user store.
- **Directory Name:**
  Click **Add/Edit** to add details of your Oracle Directory Server. For more details, see the topic on *Configuring LDAP Directories*.

The **User Search Conditions** section instructs the API Gateway to search the LDAP tree according to the following conditions:

- **Base Criteria:**
  Enter where the API Gateway should begin searching the LDAP directory (for example, `cn=Users, dc=qa, dc=vordel, dc=com`).
- **User Class:**
  Enter or select the name given by the particular LDAP directory to the *User* class. For Oracle Directory Server, select `'inetorgperson' LDAP Class`.
- **User Search Attribute:**
  The value entered depends on the type of LDAP directory to which you are connecting. When a user is stored in an LDAP directory, a number of user *attributes* are stored with that user. One of these attributes corresponds to the user name presented by the client for authentication. However, different LDAP directories use different names for that user attribute. For Oracle Directory Server, select `cn` from the drop-down list.
- **Allow Blank Passwords:**
  Select this to allow the use of blank passwords.

In the next section, you must specify the following:

- **Login Authentication Attribute:**
  In an LDAP directory tree, there must be one user attribute that uniquely distinguishes any one user from all the others. In Oracle Directory Server, the Distinguished name is referred to as the *entrydn* or Entry Domain Name. Select `Entry Domain Name` to uniquely identify the client authenticating to the API Gateway.
- **Authorization Attribute:**
  When the client has been successfully authenticated, you can use any one of that user's stored attributes in a subsequent Authorization filter. In this case, you want to use the user's Entry Domain Name (Distinguished Name) for an Authorization filter, so enter `entrydn` in the text box. However, you can enter any user attribute as long as the subsequent Authorization filter supports it. The value of the LDAP attribute specified is stored in the `authentication.subject.id` message attribute.
- **Authorization Attribute Format:**
  Because any user attribute can be specified in the **Authorization Attribute** above, you must inform the API Gateway of the type of this attribute. This information is used internally by the API Gateway in subsequent Authorization filters. Select `X.509 Distinguished Name` from the drop-down list.

Microsoft Active Directory Server
This subsection describes how to configure the **Authentication Repository** dialog for Microsoft Active Directory Server. The values enter here differ from those entered when interfacing to the Oracle Directory Server:

- **Repository Name:**
  Enter a suitable name for this search.
- **LDAP Directory:**
  Click **Add/Edit** to add details of your Active Directory Server.

The **User Search Conditions** instruct the API Gateway to search the LDAP tree according to certain criteria. The values specified are different from those selected for Oracle Directory Server, because MS Active Directory Server uses different attributes and classes to Oracle Directory Server:

- **Base Criteria:**
  The base criteria specify the base object under which to search for the user's profile (for example, `cn=Users, dc=qa, dc=vordel, dc=com`.
- **User Class:**
  In Active Directory Server, the user class is called *User*, so select `'User' LDAP Class`.
- **User Search Attribute:**
  This specifies the name of the user attribute whose value corresponds to the user name entered by the client during a successful authentication process. With Active Directory Server, this attribute is called *givenName*, which repres-

ents the name of the user. Enter `givenName` in this field.

- **Login Authentication Attribute:**
  Enter the name of the user attribute that uniquely identifies the user in the LDAP directory. This attribute is the Distinguished Name and is called *distinguishedName* in Active Directory Server. Select `Distinguished Name` from the drop-down list to uniquely identify the user. The API Gateway authenticates the username and password presented by the client against the values stored for the user identified in this field.
- **Authorization Attribute:**
  When the client has been successfully authenticated, the API Gateway can use any of that user's stored attributes in subsequent Authorization filters. Because most Authorization filters require a Distinguished Name, enter `Distinguished Name` in the text box. However, any user attribute could be entered here, as long as the subsequent Authorization filter supports it.
- **Authorization Attribute Format:**
  The API Gateway needs to know the format of the **Authorization Attribute**. Select `X.509 Distinguished Name` from the drop-down list.

IBM Directory Server

The configuration details for IBM Directory Server provide an example of a directory server that does not return a full Distinguished Name (DName) as the result of a standard LDAP user search. Instead, it returns a *contextualized DName*, which is relative to the specified **Base Criteria**. In such cases, the API Gateway can build up the full DName by combining the **Base Criteria** and the returned name. The following example shows how this works in practice.

If `C=IE` is specified as the **Base Criteria**, the IBM Directory Server returns `CN=niall, OU=Dev`, instead of the full DName, which is `C=IE, CN=niall, OU=Dev`. To enable the API Gateway to do this, leave the **Login Authentication Attribute** field blank. The API Gateway then automatically concatenates the specified **Base Criteria** (`C=IE`) with the contextualized DName returned from the directory server (`CN=niall, OU=Dev`) to obtain the fully qualified DName (`C=IE, CN=niall, OU=Dev`).

You can also leave the **Authorization Attribute** field blank, which enables the API Gateway to automatically use the fully qualified DName for subsequent Authorization Filters. You should select `X.509 Distinguished Name` from the **Authorization Attribute Format** drop-down list.

CA SiteMinder Repositories

In cases where user profiles have been stored in an existing SiteMinder server, the API Gateway can query SiteMinder to authenticate users.

To authenticate users against a CA SiteMinder repository, right-click **CA SiteMinder Repositories**, and select **Add a new Repository**. Complete the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter a suitable name for this repository.

Agent Name:
Select a previously configured SiteMinder Agent name from the drop-down list. Click **Add** to register a new agent. Complete the following fields in the **SiteMinder Connection Details** dialog:

- **Agent Name:**
  Enter the name of the agent to connect to SiteMinder in the **Agent Name** field. This name **must** correspond with the name of an agent that was previously configured in the **Policy Server**.
- **Agent Configuration Object:**
  The name entered must match the name of the Agent Configuration Object (ACO) configured in the Policy Server. The API Gateway currently does not support any of the features represented by the ACO parameters except for the `PersistentIPCheck` setting. For example, the API Gateway disregards the `DefaultAgent` parameter and uses the agent value it collects separately during agent registration.
  When the `PersistentIPCheck` ACO parameter is set to `yes`, it instructs the API Gateway to compare the IP address from the last request (stored in a persistent cookie) with the IP address in the current request to see if they

match. If the IP addresses do not match, the API Gateway rejects the request. If this parameter is set to no, this check is disabled.

- **Connection Details:**
  For more information on configuring this section, please refer to the instructions in the **SiteMinder Connection Details** section of the *SiteMinder Certificate Authentication* help page.

Resource:
Enter the name of the protected resource for which the user must be authenticated.

Alternatively, you can enter a selector representing a message attribute, which is looked up and expanded to a value at runtime. Message attribute selectors have the following format:

```
${message.attribute}
```

For example, to specify the original path on which the request was received by the API Gateway as the resource, enter the following selector:

```
${http.request.uri}
```

Action:
The user must be authenticated for a specific action on the protected resource. By default, this action is taken from the HTTP verb used in the incoming request. You can use the following selector to get the HTTP verb:

```
${http.request.verb}
```

Alternatively, you can enter any user-specified value. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Create Single Sign-On Token:
When this option is selected, SiteMinder generates a single sign-on token as part of the authentication event and returns it to the API Gateway. This is then inserted into the downstream message for re-use later, either by another instance of the API Gateway running the **SiteMinder Session Validation** filter, or by another SiteMinder-aware agent.

Put Token in Message Attribute:
Enter the name of the message attribute where you wish to store the single sign-on token. By default, the token is stored in the `siteminder.session` attribute. Please refer to the Message Attribute Reference for a complete list of available message attributes.

Database Repositories

The API Gateway can store its **Authentication Repository** in an external database. This option makes sense when an organization already has a silo of user profiles stored in the database and does not want to duplicate this store within the API Gateway's local configuration storage.

To authenticate users against a database repository, right-click **Database Repositories**, and select **Add a new Repository**. Complete the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for the database in the **Repository Name** field.

Database:
There are two basic configuration items required to retrieve a user's profile from the database:

- **Database Location:**
  You can configure connection details for the database by clicking **Add**, and completing the **Database Connection**

dialog. For details on configuring the fields on this dialog, see the *Database Connection* topic. You can edit or remove previously configured database connections by selecting them in the drop-down list and clicking **Edit** or **Delete**.

- **Database Query:**
  The **Database Query** retrieves a specific user's profile from the database to enable the API Gateway to authenticate them. Having successfully authenticated the user, you can select an attribute of this user to use for the authorization filter later in the policy. The **Database Query** can take the form of an SQL statement, stored procedure, or function call. For details on how to configure the **Database Query**, see the *Database Query* topic.

Format Password Received From Client:
If the user sends up a clear-text password to the API Gateway, but that user's password is stored in a hashed format in the database, it is the API Gateway must hash the password before performing the authentication step.

- **Hash Client Password:**
  Depending on whether you wish to hash the user's submitted password, select the appropriate radio button.
- **Hash Format:**
  If you have selected to hash the client's password, the API Gateway needs to know the format of the hashed password. The most typical formats are available from the drop-down list, however, you can also enter another format. Formats should be entered in terms of message attribute selectors. The following formats are available from the **Hash Format** drop-down list.

```
${authentication.subject.id}:${authentication.subject.realm}:${authentication.
subject.password}
${authentication.subject.password}
```

  The first option combines the username, authentication realm, and password respectively. This combination is then hashed. The second option simply creates a hash of the user's password.
- **Hash Algorithm:**
  Select either **MD5** or **SHA1** to use as the digest algorithm to use when creating the hash.

For more details on selectors, see *Selecting Configuration Values at Runtime*.

Query Result Processing:
This section enables you to provide the API Gateway with some meta information about the result returned by the **Database Query** configured earlier on this screen. It enables allows you to identify the name of the database table column or row that contains the user's password, and also the name of the column or row that contains the attribute that is to be used for the authorization filter.

- **Password Column:**
  Specify the name of the database table column that contains the user's password. The contents of this column are compared to the password submitted by the user.
- **Password Type:**
  Depending on how the user's password has been stored in the database, select either **Clear Password** or **Digest Password** from the drop-down list.
- **Authorization Attribute Column:**
  By running the **Database Query**, all of the user's attributes are returned. Only the user's username and password are used for the authentication event. You can also use one of the other user's attributes for authorization at a later stage in the policy. The additional authorization attribute should be either a username or an X.509 distinguished name (DName). You should enter the name of the column containing the username or the DName here, but only if this value is required for authorization purposes.
- **Authorization Attribute Format:**
  The API Gateway's authorization filters all operate on the basis of a username or DName. They all evaluate whether a user identified by a username or DName is allowed to access a specific resource. Select the appropriate format

from the drop-down list depending on what type of user credential is stored in the database table column entered above.

Entrust GetAccess Repositories

Entrust GetAccess provides Identity Management and access control services for Web resources. It centrally manages access to Web applications, enabling users to benefit from a single sign-on capability when accessing the applications that they are authorized to use.

You can configure the API Gateway to connect to a group of GetAccess servers in a round-robin fashion. This provides the necessary failover capability when one or more GetAccess servers are not available. When the API Gateway successfully authenticates to a GetAccess server, it obtains authorization information about the end-user from the GetAccess SAML PDP. The authorization details are returned in a SAML authorization assertion, which is then validated by the API Gateway to determine whether the request should be denied.

To authenticate users against an Entrust GetAccess repository, right-click **Entrust GetAccess Repositories**, and select **Add a new Repository**. Configure the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for this repository

Request:
Configure the following request settings:

- **URL Group:**
  Select a URL group from the drop-down list. This group consists of a number of GetAccess Servers to which the API Gateway round-robins connection attempts. You can add URL groups under the **External Connections** tree node in Policy Studio. Expand the **URL Connection Sets** node, right-click **Entrust GetAccess URL Sets**, and select **Add a URL Set**. For more details on adding and editing URL groups, see the *Configuring URL Groups* topic.
- **WS-Trust Attribute Field Name:**
  Specify the field name for the `Id` field in the WS-Trust request. The default is `Id`.

Response:
Configure the following response settings:

- **SOAP Actor/Role:**
  To add the SAML authorization assertion to the response message, select a SOAP actor/role to indicate the WS-Security block where the assertion is added. By leaving this field blank, the assertion is not added to the message.

  Drift Time:
  The specified time is used to allow for the possible difference between the time on the GetAccess SAML PDP and the time on the machine hosting the API Gateway. This comes into effect when validating the SAML authorization assertion.

Further Information
For details on using a filter to integrate the API Gateway with Entrust GetAccess, see the *Entrust GetAccess Authorization* topic.

Oracle Access Manager Repositories

You can authorize an authenticated user for a particular resource against an Oracle Access Manager (OAM) repository. After successful authentication, OAM issues a Single Sign On (SSO) token, which can then be used instead of the user name and password.

To authenticate users against an Oracle Access Manager repository, right-click **Oracle Access Manager Repositories**, and select **Add a new Repository**. Configure the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for this repository.

Resource Request:
Configure the following settings for the resource request:

- **Resource Type:**
  Enter the type of the resource for which you are requesting access. For example, for access to a Web-based URL, enter `http`.
- **Resource Name:**
  Enter the name of the resource for which the user is requesting access. By default, this field is set to `//hostname${http.request.uri}`, which contains the original path requested by the client.
- **Operation:**
  In most access management products, users are authorized for a limited set of actions on the requested resource. For example, users with management roles may be permitted to write (`HTTP POST`) to a certain Web Service, but users with junior roles might only have read access (`HTTP GET`) to the same service. Use this field to specify the operation to which you want to grant the user access on the specified resource. By default, this is set to the `http.request.verb` message attribute, which contains the HTTP verb used by the client to send the message to the API Gateway (for example, `HTTP POST`).
- **Include query string:**
  Select whether the query string parameters are used by the OAM server to determine the policy that protects this resource. This setting is optional if the policies configured do not rely on the query string parameters.
- **Client location:**
  If the client location must be passed to OAM for it to make its decision, you can enter a valid DNS name or IP address to specify this location.
- **Optional Parameters:**
  You can add optional additional parameters to be used in the authentication decision. The available optional parameters include the following:

`ip`	IP address, in dotted decimal notation, of the client accessing the resource.
`operation`	Operation attempted on the resource (for HTTP resources, one of `GET`, `POST`, `PUT`, `HEAD`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, or `OTHER`).
`resource`	The requested resource identifier (for HTTP resources, the full URL).
`targethost`	The host (`host:port`) to which resource request is sent.

Note

One or more of these optional parameters may be required by certain authentication schemes, modules, or plugins configured in the OAM server. To determine which parameters to add, see your OAM server configuration and documentation.

Single Sign On:
Configure the following settings for single sign on:

- **Create SSO Token:**
  Select whether to create an SSO token. This is selected by default.

- **Store SSO Token in User Attribute:**
  Enter the name of the message attribute that contains the user's SSO token. This attribute is populated when authenticating to Oracle Access Manager using the *HTTP Basic Authentication* or *HTTP Digest Authentication* filter. By default, the SSO token is stored in the `oracle.sso.token` message attribute.
- **Add SSO Token to User Attributes:**
  Select whether to add the SSO Token to user message attributes. This is selected by default.

OAM Access Server SDK Directory:
Enter the path to your OAM Access Server SDK directory. For more details on the OAM Access Server SDK, see your Oracle Access Manager documentation.

Further Information
For details on using filters to integrate the API Gateway with Oracle Access Manager, see Chapter 23, Oracle Access Manager.

Oracle Entitlements Server 10g Repositories

You can authenticate and authorize a user for a particular resource against an Oracle Entitlements Server (OES) 10g repository.

For example, the API Gateway can extract credentials from the message sent by the client, and delegate authentication to OES 10g. When the client has been authenticated, the API Gateway queries OES 10g to see if the client is permitted to access the Web Service resource. When authentication and authorization have passed, the message is trusted and forwarded to the target Web Service.

To authenticate and authorize users against an OES 10g repository, right-click **Oracle Entitlements Server 10g Repositories**, and select **Add a new Repository**. Configure the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for this repository.

Oracle SSM Settings:
Click **Configure** to launch the **Oracle Security Service Module Settings** dialog. For details on configuring these settings, see the *Oracle Security Service Module Settings (10g)* topic.

RADIUS Repositories

You can configure the API Gateway to authenticate users in a Remote Authentication Dial In User Service (RADIUS) repository. RADIUS is a client-server network protocol that provides centralized authentication and authorization for clients connecting to remote services.

To authenticate users against a RADIUS repository, perform the following steps:

1. Right-click **RADIUS Repositories**, and select **Add a new Repository**.
2. In the **Authentication Repository** dialog, enter the RADIUS **Repository Name**.
3. On the **Client** tab, select the RADIUS clients that you wish to authenticate to the repository. For details on how to add clients to this list, see the *RADIUS Clients* topic.
4. On the **Attributes** tab, click **Add** to add a RADIUS attribute. This is a name-value pair used to determine how access is granted. Examples include `User-Name`, `User-Password`, `NAS-IP-Address`, or `NAS-Port`).
5. In the **RADIUS Attributes** dialog, specify a **Name** (for example, `User-Name`). You can select standard RADIUS attributes from the drop-down list, or enter a custom attribute.
6. Enter a **Value**, and click **OK**.
7. Click **OK**.

Repeat steps 4-6 to add multiple attributes. You can edit or delete attributes using the buttons provided.

RSA Access Manager Repositories

RSA Access Manager (formerly known as RSA ClearTrust) provides Identity Management and access control services for Web applications. It centrally manages access to Web applications, ensuring that only authorized users are allowed access to resources. Integration with RSA Access Manager requires RSA ClearTrust SDK version 6.0.

To authenticate users against an RSA Access Manager repository, right-click **RSA Access Manager Repositories**, and select **Add a new Repository**. Configure the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for this repository.

Connection Details:
The API Gateway can connect to a group of Access Manager *Authorization Servers* or *Dispatcher Servers*. When multiple Access Manager Authorization Servers are deployed for load-balancing purposes, the API Gateway first connects to a Dispatcher Server, which returns a list of active Authorization Servers. An attempt is made to connect to one of these Authorization Servers using round-robin DNS. If the first Dispatcher Server in the Connection Group is not available, the API Gateway attempts to connect to the Dispatcher Server with the next highest priority in the group, and so on.

If a Dispatcher Server has not been deployed, the API Gateway can connect directly to an Authorization Server. If the Authorization Server with the highest priority in the Connection Group is not available, the API Gateway attempts to connect to the Authorization Server with the next highest priority, and so on. You can select the type of the Connection Group using the **Authorization Server** or **Dispatcher Server** radio button. All servers in the group must be of the same type.

Connection Group:
Select the **Connection Group** to use for authenticating clients. You can add Connection Groups under the **External Connections** tree node in the Policy Studio. Expand the **Connection Sets** node, right-click **RSA ClearTrust Connection Sets**, and select **Add a Connection Set**. For more details on adding and editing Connection Groups, see the *Configuring Connection Groups* topic.

Authentication Type:
Select one of the following authentication types for the connection:

- HTTP Basic
- Windows NT
- RSA SecureID
- LDAP
- Certificate Distinguished Name

Further Information
For more details on prerequisites and on using a filter to integrate the API Gateway with RSA Access Manager, see the *RSA Access Manager Authorization* topic.

Tivoli Repositories

The API Gateway can integrate with Tivoli Access Manager to authenticate users. To authenticate users against a Tivoli repository, right-click **Tivoli Repositories**, and select **Add a new Repository**.

For more information on how to configure API Gateway to communicate with a Tivoli server, see the *Tivoli Integration* topic.

Certificate Chain Check

Overview

Whenever the API Gateway receives a client's X.509 certificate, either in an XML Signature or as part of an SSL handshake, it needs to determine whether that certificate can be trusted. For example, it is a trivial task for a user to generate a structurally sound X.509 certificate. This certificate can then be used to negotiate mutually authenticated connections to publicly available services.

Clearly, this scenario represents a security nightmare for IT administrators. You cannot allow any user to generate their own certificate and use it on the Internet. The server must be able to *trust* the authenticity of the client certificate. Furthermore, it must be able to verify that the certificate originated from a trusted source. To do this, a server can perform a *certificate chain check* on the client certificate.

The main purpose of certificate chain validation is to ensure that a certificate has been issued by a trusted source. Typically, in a Public-Key Infrastructure (PKI), a Certificate Authority (CA) is responsible for issuing and distributing certificates. This infrastructure is based on the premise of *transitive trust*: If everybody trusts the CA, everybody transitively trusts the certificates issued by that CA. If entities only trust certificates that have been issued by the CA, they can then reject certificates which have been self-generated by clients.

When a CA issues a certificate, it digitally signs the certificate and inserts a copy of its own certificate into it. This is called a certificate chain. Whenever an application (such as the API Gateway) receives a client certificate, it can extract the issuing CA's certificate from it, and run a certificate chain check to determine whether it should trust the CA. If it trusts the CA, it also trusts the client certificate.

The question then arises how does the API Gateway *trust* a Certificate Authority? The API Gateway maintains a repository of both trusted CA certificates, and trusted server certificates for use in SSL communications. To trust a certain CA, that CA's certificate must be imported into the **Certificate Store**.

Configuration

The **Policy Studio** provides an easy-to-use interface for configuring certificate chain validation. This interface allows you to amalgamate CA and server certificates into groups such that if an incoming client certificate has been issued by any of the CAs in the group, the API Gateway trusts the certificate. Enter a name for the group in the **Group Name** field. To populate the new group, click the **Add/Edit** button.

By selecting a group from this list, the members of this group are displayed in the **Certificate Alias** table. To add and/or remove members from the selected group, click the **Add/Edit** button. Certificates can be added to and removed from new or existing groups using the **Configure Trusted Certificate Groups** dialog which is displayed on clicking the **Add/Edit** button.

The **Configure Trusted Certificate Groups** dialog consists of 2 main tables. The first lists all certificates currently in the **Certificate Store** (those trusted by the API Gateway). The second lists the members of the group selected in the **Group Name** field. To add a certificate to a trusted group, select it from the **Certificate Store** table, and click **Add**. The certificate appears in the group certificates table. Similarly, to remove a certificate from the group, select it from the group certificates table, and click **<- Remove**. The certificate is removed from the group table.

You can also add, remove, and view certificates in the **Certificate Store** using this dialog. To add a certificate to the **Certificate Store**, click **Add**, which displays the **Import Certificate** dialog. Browse to the location of the CA certificate file, and enter an **Alias** for the certificate. This is used to uniquely identify the certificate in the API Gateway. You can remove a certificate by selecting it in the **Trusted Store** table, and clicking **Remove**. The certificate is removed from the table, and is no longer be trusted by the API Gateway.

Finally, you can also examine the details of any one of the certificates in the **Certificate Store**. To do this, again select a certificate from the **Trusted Certificate** table, and click the **View** button.

Certificate Validation

Overview

Whenever the API Gateway receives an X.509 certificate, either as part of the SSL handshake or as part of the XML message itself, it is important to be able to determine whether that certificate is legitimate or not. Certificates can be revoked by their issuers if it becomes apparent that the certificate is being used maliciously. Such certificates should never be trusted, and so it is very important that the API Gateway can perform certificate validation.

The API Gateway uses the following methods/protocols to validate certificates:

OCSP - Online Certificate Status Protocol
OCSP is an automated certificate checking network protocol. The API Gateway can query the OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

CRL - Certificate Revocation Lists
A CRL is a signed list indicating a set of certificates that are no longer considered valid (i.e. revoked certificates) by the certificate issuer. The API Gateway can query a CRL to find out if a given certificate has been revoked - if the certificate is present in the CRL, it should not be trusted.

XKMS - XML Key Management Services
XKMS is an XML-based protocol for (amongst other things) establishing the trustworthiness of a certificate over the Internet. The API Gateway can query an XKMS responder to determine whether or not a given certificate can be trusted or not.

Configuration

The API Gateway can check that the validity of a client certificate using any of the following methods:

1. OCSP - Online Certificate Status Protocol
2. CRL - Certificate Revocation Lists
3. XKMS - XML Key Management Services

Note

To validate a certificate using either an or CRL lookup, the issuing CA's certificate should be trusted by the API Gateway. This is because for a CRL lookup, the CA's public key is needed to verify the signature on the CRL, and for an OCSP request, the protocol stipulates that the CA's public key must be submitted as part of the request. The issuing CA's public key is not always present in issued certificates, so it is necessary to retrieve it from the API Gateway's certificate store instead.

OCSP - Online Certificate Status Protocol

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *OCSP* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Select a group of OCSP Responders from the **URL Group** field. The API Gateway will attempt to connect to the Responders in the selected group in a round-robin fashion. It will attempt to connect to the Responders with the highest priority first, before connecting to Responders with a lower priority. URL Groups can be added, edited, and removed by selecting the **Add**, **Edit**, and **Remove** buttons respectively.
   Take a look at the Configuring URL Groups section below for more information on adding and editing URL groups.
5. Enter the user name of a User whose key will be used to sign status requests sent to the OCSP responder in the **User Name** field.

6. Enter the corresponding password for this user in the **Password** field.
7. If the OCSP Responder signs the OCSP response, and you wish to validate this signature, select the **Validate Response** checkbox.

CRL - Certificate Revocation Lists

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *CRL* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Select a previously configured LDAP directory from the **LDAP directory** dropdown list, or add a new one using the **Add** button.

XKMS - XML Key Management Services

1. Enter or select a name for the validation rule in the **Name** field.
2. Select *XKMS* from the **Type** dropdown.
3. Optionally enter a description of the rule in the **Description** field.
4. Enter the URL of the XKMS Responder in the **URL** field.
5. Enter the user name of a User whose key will be used to sign status requests sent to the XKMS responder in the **User Name** field.
6. Enter the corresponding password for this user in the **Password** field.

Configuring URL Groups

The API Gateway can make connections on a round-robin basis to the URLs listed in a URL group, thus enabling a high degree of failover to external servers. URL groups can be configured by selecting the **Add** and/or **Edit** buttons.

The API Gateway will attempt to connect to the listed servers according to the priorities assigned to them. So, for example, let's assume there are two "High" priority URLs, one "Medium" URL, and a single "Low" URL configured. Assuming the API Gateway can successfully connect to the two "High" priority URLs, it will alternate requests between these two URLs only in a round-robin fashion. The other group URLs will not be used at all. If, however, both of the "High" priority URLs become unavailable, the API Gateway will then try to use the "Medium" priority URL, and only if this fails will the "Low" priority URL be used.

So, in general, the API Gateway will attempt to round-robin requests over URLs of the same priority, but will use higher priority URLs before lower priority ones. When a new URL is added to the group it is automatically given the highest priority. Priorities can then be changed by selecting the URL and clicking the **Up** and **Down** buttons.

Individual URLs can be added and edited by selecting the URL from the table and clicking on the **Add** and **Edit** buttons respectively.

The following fields should be completed:

- **URL:**
  Enter the full URL of the external server.
- **Timeout:**
  Specify the timeout in seconds for connections to the specified server.
- **Time:**
  Whenever the server becomes unavailable for whatever reason (maintenance, for example), no attempt will be made to connect to that server until the time specified here has elapsed. In other words, once a connection failure has been detected, the next connection to that URL will be made after this amount of time.
- **Username:**
  If the specified server requires clients to authenticate to it over 2-way SSL, a **User** must be selected here for authen-

tication.

- **Password:**
  Enter the password for this user.
- **Host/IP:**
  If the specified server sits behind a proxy server, the host name or IP address of the proxy server must be entered here.
- **Port:**
  Enter the port on which the proxy is listening.

Compressed Content Encoding

Overview

The API Gateway supports HTTP content encoding for the `gzip` and `deflate` compressed content encodings. This enables the API Gateway to compress files and deliver them to clients (for example, web browsers) and to back-end servers. For example, HTML, text, and XML files can be compressed to approximately 20% to 30% of their original size, thereby saving on server traffic. Compressing files causes a slightly higher load on the server, but this is compensated by a significant decrease in client connection times to the server. For example, a client that takes six seconds to download an uncompressed XML file might only take two seconds for a compressed version of the same file.

In the Policy Studio, an **Input Encodings** list specifies the content encodings that the API Gateway can accept from peers, and an **Output Encodings** list specifies the content encodings that the API Gateway can apply to outgoing messages. You can configure these settings globally, per remote host, or per listening interface.

Encoding of HTTP Responses

Content encoding of HTTP responses is negotiated using the `Accept-Encoding` HTTP request header. This enables a client to indicate its willingness to receive a particular encoding in this header. The server can then choose to encode the response with one of these client-supported encodings, and indicate this with the `Content-Encoding` header.

When the API Gateway is acting as a client communicating with a server, it uses the currently configured **Input Encodings** list to format the `Accept-Encoding` header sent to the server, thereby requesting the server to apply one of these encodings to it. If the server decides to apply one of these encodings, the API Gateway automatically inflates the compressed response when it is received.

When acting as a server, the API Gateway selects an output encoding from the intersection of what the client specified in its `Accept-Encoding` header, and the currently configured **Output Encodings**, and applies that encoding to the response.

Encoding of HTTP Requests

Because a request arrives unsolicited from a client to a server, there is not normally a chance to negotiate the server's ability to process any optional features, so the automatic negotiation provided by the `Accept-Encoding` header is not available.

When acting as a client, the API Gateway selects the first configured encoding from the **Output Encodings** list to encode its request to the server. If the server fails to accept this encoding, it most likely responds with an HTTP 415 error, and the API Gateway treats this as a general HTTP error. Therefore, if the server is unable to accept content encodings, the API Gateway must be configured not to send them to that particular server.

By default, the API Gateway always accepts any supported encoding from clients, regardless of settings. For example, if a client sends gzipped content, the API Gateway inflates it regardless of configured settings.

Delimiting the End of an HTTP Message

HTTP sessions can encode a number of request/response pairs. The rules for delimiting the end of each message and the start of the next one are well defined, but complex due to requirements for historical compatibility, and poor support from HTTP entities.

HTTP Requests
There are two ways to delimit the end of an HTTP request:

- A `Content-Length` header in the request indicates to the server the exact length of the payload entity following the HTTP headers, and can be used by the receiving server to locate the end of that entity.
- Alternatively, an HTTP/1.1 server should accept chunked transfer encoding, which precedes each chunk of the en-

tity with a length, until a zero-length chunk indicates the end.

The benefit of using chunked transfer encoding is that the client does not need to know the length of the transmitted entity when it sends HTTP request headers (unlike when inserting a `Content-Length` header). Because the API Gateway compresses the requests on the fly, it is prohibitively expensive to calculate the content length before compressing the body. As a result, outbound content encoding is only supported when talking to HTTP/1.1 servers that support chunked transfer encoding.

Note

All HTTP/1.1 servers are required to support chunked transfer encoding, but unfortunately some do not, so you can use **Remote Host** settings to configure whether a destination is capable of supporting the chunked encoding in HTTP/1.1, regardless of its advertised HTTP protocol version. For more details, see the topic on *Remote Host Settings*.

HTTP Responses
For HTTP responses, the server has three options for delimiting the end of the entity. The two mentioned above, and also the ability to close the HTTP connection after the response is transmitted. The receiving client can then infer that the entire message has been received due to the normal end of the TCP/IP session. When content encoding responses, the API Gateway avoids using `Content-Length` headers in the response, and uses chunked encoding or TCP/IP connection closure to indicate the end of the response. This means that content encoding of responses is supported for HTTP/1.0 or HTTP/1.1 clients.

Configuring Content Encoding

In the Policy Studio, you can configure HTTP content encodings in the **Content Encodings** dialog. You can configure the following settings globally, per remote host, or per listening interface:

Input Encodings	Specifies the content encodings that the API Gateway can accept from peers.
Output Encodings	Specifies the content encodings that the API Gateway can apply to outgoing messages.

The available content encodings include `gzip` and `deflate`. By default, the content encodings configured in the **Default Settings** are used. You can override this setting at the HTTP interface and Remote Host levels.

Configuring Content Encodings
To configure content encodings, perform the following steps in the **Content Encodings** dialog:

1. Deselect the **Use Default** setting.
2. Select the content encoding(s) that you wish to configure in the **Available Content Encodings** list on the left.
3. Click the **>** or **>>** button to move the content encoding(s) to the **Content Encodings** list on the right. You can also double-click a content encoding to move it to the right or left.
4. Click **OK**. This displays the configured encoding(s) in the **Input Encodings** or **Output Encodings** field (for example, `gzip, deflate`).

Configuring No Content Encodings
Alternatively, to configure no content encodings, deselect the **Use Default** setting, and click **OK**. This displays `None` in the **Input Encodings** or **Output Encodings** field.

Note

You can select the **Use Default** setting to switch to the **Default Settings** without losing your original content encoding selection.

Further Information

For more details on the different levels at which you can configure content encodings in the Policy Studio, see the following topics:

- *Default Settings*
- *Remote Host Settings*
- *Configuring HTTP Services*

For more details on HTTP content encoding, see HTTP RFC 2616:
`http://www.w3.org/Protocols/rfc2616/rfc2616.html` [http://www.w3.org/Protocols/rfc2616/rfc2616.html]

Configuring Connection Groups

Overview

A **Connection Group** consists of a number of external servers that the API Gateway connects to (for example, RSA Access Manager servers for authorization). The API Gateway attempts to connect to all the servers in the group in a round-robin fashion, therefore providing a high degree of failover. If one or more servers are unavailable, the API Gateway can still connect to an alternative server.

The API Gateway attempts to connect to the listed servers according to the priorities assigned to them. For example, assume there are two High priority servers, one Medium priority server, and one Low priority server configured. Assuming the API Gateway can successfully connect to the two High priority servers, it alternates requests between these two servers only in a round-robin fashion. The other group servers are not used. However, if both High priority servers become unavailable, the API Gateway then tries to use the Medium priority server, and only if this fails is the Low priority server used.

Connection Groups are available in Policy Studio tree under the **External Connections** node according to the filter from which they are available. For example, Connection Sets under the **RSA ClearTrust Connection Sets** node are available in the RSA Access Manager filter. For more details, see the *RSA Access Manager Authorization* topic.

Configuring a Connection Group

You can configure a Connection Group using the **Connection Group** dialog. The external servers are listed in order of priority in the table on the **Connection Group** dialog. The API Gateway attempts to connect to the server at the top of the list first. If this server is not available, a connection attempt is made to the second server, and so on until an available server is contacted. If none of the listed servers are available, the client is not authorized and a SOAP fault is returned to the client.

You can increase or decrease the priorities of the listed external servers using the **Up** and **Down** buttons. You can add, edit, and delete Access Manager servers using the **Add**, **Edit**, and **Remove** buttons.

Configuring a Connection

You can configure a single connection using the **Connection Configuration** dialog. To configure a single **Access Manager Connection**, perform the following steps:

1. Enter the name or IP address of the machine hosting the selected Access Manager server in the **Location** field.
2. Enter the **Port** on which the specified Access Manager server is listening.
3. Select a suitable **Timeout** in seconds for connections to this server.
4. Select the appropriate **Connection Type** for the API Gateway to use when connecting to the specified Access Manager server. Connections between the API Gateway and the Access Manager server can be made in the clear, over Anonymous SSL, or Mutual SSL Authentication (two-way SSL).
5. If **SSL Authentication** is selected, you must select an **SSL mutual authentication certificate**. This certificate is then used to authenticate to the Access Manager server.

Configuring Cron Expressions

Overview

The **Cron Dialog** enables you to create a cron expression used to trigger regularly occurring events (for example, generate a report, or block or allow messages at specified times). You can use the time tabs in this dialog to guide you through the configuration steps. Alternatively, enter the cron expression value directly in the text boxes. When you have created the cron expression, you can click the **Test Cron** button to test the value of the cron expression and see when it is next due to fire.

For background information and details on cron expression syntax, see the section on Cron Expressions in the *Policy Execution Scheduling* topic.

Creating a Cron Expression using the Time Tabs

Using the time tabs in the dialog to guide you through the configuration steps is the default option. You can create a cron expression to trigger at the specified times using the following settings:

Seconds:
Select one of the following options:

Every Second of the Minute	Fires every second of the minute. This is the default setting.
Just on Second	Fires only on the specified second of the minute.
Range from Second	Fires over a range of seconds. For example, if the first value is 10 and the second value is 25, the trigger starts firing on second 10 of the minute and continues to fire for 15 seconds.
Start on Second	Fires on the specified second of the minute and repeats every specified number of seconds. For example, if the first number is 15 and the second number is 30, the trigger fires at 15 seconds and repeats every 30 seconds until stopped.
On Multiple Seconds	Fires on the specified seconds of each minute. Enter a comma separated list of seconds (values of `0-59` inclusive). For example, `10,20,30`.

Minutes:
Select one of the following options:

Every Minute of the Hour	Fires every minute of the hour. This is the default setting.
Just on Minute	Fires only on the specified minute of the hour.
Range from Minute	Fires over a range of minutes. For example, if the first value is 5 and the second value is 15, the trigger starts firing on minute 5 of the hour and continues to fire for 10 minutes.
Start on Minute	Fires on the specified minute of the hour and repeats every specified number of minutes. For example, if the first number is 10 and the second number is 20, the trigger fires at 10 minutes and repeats every 20 minutes until stopped.

On Multiple Minutes	Fires on the specified minute of each hour. Enter a comma-separated list of minutes (values of `0-59` inclusive). For example, `5,15,30`.

Hours:
Select one of the following options:

Every Hour of the Day	Fires every hour of the day. This is the default setting.
Just on Hour	Fires only on the specified hour of the day.
Range from Hour	Fires over a range of hours. For example, if the first value is 9 and the second value is 17, the trigger starts firing on hour 9 of the day and continues to fire for 8 hours.
Start on Hour	Fires on the specified hour of the day and repeats every specified number of hours. For example, if the first number is 6 and the second number is 2, the trigger fires at hour 6 and repeats every 2 hours until stopped.
On Multiple Hours	Fires on the specified hours of each day. Enter a comma-separated list of hours (values of `0-23` inclusive). For example, `6,12,18`.
Multiple Ranges	Fires on the specified ranges of hours of each day. Enter comma separated ranges of hours (values of `0-23` inclusive). For example, `9-1,14-17`.

Day:
You must first select **Day of Week** or **Day of Month** from the drop-down list (using both of these fields in not supported). **Day of Month** is selected by default.

Day of Month
Select one of the following options:

Every Day of the Month	Fires every day of the month. This is the default setting.
Just on Day	Fires only on the specified day of the month.
Range from Day	Fires over a range of days. For example, if the first value is 7 and the second value is 14, the trigger starts firing on day 7 of the month and continues to fire for 9 days.
Start on Day	Fires on the specified day of the month and repeats every specified number of days. For example, if the first day is 2 and the second number is 5, the trigger fires at day 2 and repeats every 5 days until stopped.
On Multiple Days	Fires on the specified days of each month. Enter a comma separated list of days (values of `1-32` inclusive). For example, `1,14,21,28`.
Last Day of the Month	Fires on the last day of each month (for example, 31 January, or 28 February in non-leap years).
Last Week Day of the Month	Fires on the last week day of each month (Monday-Friday inclusive only).

Day of Week
Select one of the following options:

Every Day of the Week	Fires every day of the week. This is the default setting.
Just on *Day*	Fires only on the specified day of the week. Defaults to `SUN`.
Range from *Day*	Fires over a range of days. For example, if the first value is `MON` and the second value is `FRI`, the trigger starts firing on `MON` and continues to fire until `FRI`.
Start on *Day*	Fires on the specified day of the week and repeats every specified number of days. For example, if the first day is `TUES` and the number is 3, the trigger fires on `TUES` and repeats every 3 days until stopped.
On Multiple *Days*	Fires on the specified days of each week. Enter a comma separated list of days. For example, `MON,WED,FRI`.
Last Day of the Week	Fires on the last day of each week (`SAT`).
On the *Nth Day*	Fires on the Nth day of the week of each month (for example, the second `FRI` of each month).

Month:
Select one of the following options:

Every Month of the Year	Fires every month of the year. This is the default setting.
Just on *Month*	Fires only on the specified month of the year. Defaults to `JAN`.
Range from *Month*	Fires over a range of months. For example, if the first value is `MAY` and the second value is `JUL`, the trigger starts firing on `MAY` and continues to fire until `JUL`.
Start on *Month*	Fires on the specified month of the year and repeats every specified number of months. For example, if the first month is `FEB` and the number is 2, the trigger fires in `FEB` and repeats every 2 months until stopped.
On Multiple *Months*	Fires on the specified months of each year. Enter a comma-separated list of months (values of `JAN-DEC` or `1-12` inclusive). For example, `MAR,JUN,SEPT`.

Year:
Select one of the following options:

Every Year	Fires every year. This is the default setting.
No Specific Year	Fires no specific year.
Just on *Year*	Fires only on the specified year. Defaults to current year.
Range from *Year*	Fires over a range of years. For example, if the first value is `2012` and the second value is `2015`, the trigger starts firing on `2012` and continues to fire until `2015`.

Start on *Year*	Fires on the specified year and repeats every specified number of years. For example, if the first year is `2012` and the number is 2, the trigger fires in `2012` and repeats every 2 years until stopped.
On Multiple *Years*	Fires on the specified Year. Enter a comma-separated list of years (for example, `2012,2013,2015`).

Entering a Cron Expression

To enter the cron expression value directly in the dialog, click **Create cron expression using edit boxes**, and enter the values in the appropriate boxes. For example, the following cron expression fires on April 27 and 28, at any time except those received between 10:00:01 and 10:59:59:

```
* * 0-9,11-23 27-28 APR ?
```

For details on cron expression syntax and special characters, see the section on Cron Expressions in the *Policy Execution Scheduling* topic.

Testing the Cron Expression

When you have configured the cron expression using either approach, click the **Test Cron** button to test the syntax of the cron expression value and view when it is next due to fire. If the configured cron expression is invalid, a warning dialog is displayed.

Results:
The test results include the following output:

Expression	Displays the configured cron expression. For example, the default is: `* * 9-17 * * ? *`
Next Fire Times	Displays when cron expression is next due to fire. For example, `Next fire event: Fri Jul 22 10:26:09 EST 2012`.

Further Information

For details on using the **Cron Dialog** to create cron expressions that trigger regularly occurring events (for example, generate reports, or block or allow messages at specified times), see the following topics:

- *Time Filter*
- *Scheduled Reports*

Database Connection

Overview

The details entered on the **Configure Database Connection** dialog specify how the API Gateway connects to the database. The API Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented using *Apache Commons DBCP (Database Connection Pools)*. The settings in the **Advanced - Connection pool** section of this screen configure the database connection pool. For details on how the fields on this screen correspond to specific DBCP configuration settings, see the Database Connection Pool Settings table.

Configuring the Database Connection

Configure the following fields on this screen:

Name:
Enter a name for the database connection in the **Name** field.

URL:
Enter the fully qualified URL of the location of the database. The following table shows examples of database connection URLs, where `reports` is the name of the database and `DB_HOST` is the IP address or host name of the machine on which the database is running:

Database	***Example Connection URL***
Oracle	`jdbc:oracle:thin:@DB_HOST:1521:reports`
Microsoft SQL Server	`jdbc:sqlserver://DB_HOST:1433;DatabaseName=reports;integratedSecurity=false;`
MySQL	`jdbc:mysql://DB_HOST:3306/reports`
IBM DB2	`jdbc:db2://DB_HOST:50000/reports`

User Name:
The username to use to access the database.

Password:
The password for the user specified in the **User Name** field.

Initial pool size:
The initial size of the DBCP pool when it is first created.

Maximum number of active connections:
The maximum number of active connections that can be allocated from the JDBC pool at the same time. The default maximum is 8 active connections.

Maximum number of idle connections:
The maximum number of active connections that can remain idle in the pool without extra connections being released. The default maximum is 8 connections.

Minimum number of idle connections:
The minimum number of active connections that can remain idle in the pool without extra connections being created. The default minimum is 8 connections.

Maximum wait time:
The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely. The default time is 10000ms, and a value of -1 indicates an indefinite time to wait.

Time between eviction:
The number of milliseconds to sleep between runs of the thread that evicts unused connections from the JDBC pool.

Number of tests:
The number of connection objects to examine from the pool during each run of the evictor thread. The default number of objects is 3.

Minimum idle time:
The minimum amount of time an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).

Database Connection Pool Settings

The table below shows the correspondence between the fields in the **Advanced - Connection pool** section of the screen and the Apache Commons DBCP configuration properties:

Field Name	***DBCP Configuration Property***
URL	`url`
User Name	`username`
Password	`password`
Initial pool size	`initialSize`
Maximum number of active connections	`maxActive`
Maximum number of idle connections	`maxIdle`
Minimum number of idle connections	`minIdle`
Maximum wait time	`maxWait`
Time between eviction	`timeBetweenEvictionRunsMillis`
Number of tests	`numTestsPerEvictionRun`
Minimum idle time	`minEvictableIdleTimeMillis`

Connection Validation

By default, when the API Gateway makes a connection, it performs a simple connection validation query. This enables the API Gateway to test the database connection before use, and to recover if the database goes down (for example, if there is a network failure, or if the database server reboots). The API Gateway validates connections by running a simple SQL query (for example, a `SELECT 1` query with MySQL). If it detects a broken connection, it creates a new connection to replace it.

Test the Connection

When you have specified all the database connection details, you can click the **Test Connection** button to verify that the connection to the database is configured successfully. This enables you to detect any configuration errors at design time instead at runtime.

Database Query

Overview

The **Database Statement** dialog enables you to enter an SQL query, stored procedure, or function call that the API Gateway runs to return a specific user's profile from a database.

Configuration

The following fields should be completed on this screen:

Name:
Enter a name for this database query here.

Database Query:
Enter the actual SQL query, stored procedure, or function call in the text area provided. When executed, the query should return a single user's profile. The following are examples of SQL statements and stored procedures:

```
select * from users where username=${authentication.subject.id}

{ call load_user (${authentication.subject.id}, ${out.param}) }

{ call ${out.param.cursor} := p_test.f_load_user(${authentication.subject.id}) }
```

These examples show that you can use selectors in the query. The selector that is most commonly used in this context is `${authentication.subject.id}`, which specifies the message attribute that holds the identity of the authenticated user. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Statement Type:
The database can take the form of an SQL query, stored procedure, or function call, as shown in the above examples. Select the appropriate radio button depending on whether the database query is an SQL **Query** or a **Stored procedure/ function call**

Table Structure:
To process the result set that is returned by the database query, the API Gateway needs to know whether the user's attributes are structured as rows or columns in the database table.

The following example of a database table shows the user's attributes (Role, Dept, and Email) structured as table columns:

Username	***Role***	***Dept***	***Email***
Admin	Administrator	Engineering	admin@org.com
Tester	Testing	QA	tester@org.com
Dev	Developer	Engineering	dev@org.com

In the following table, the user's attributes have been structured as name-value pairs in table rows:

Username	***Attribute Name***	***Attribute Value***
Admin	Role	Administrator

Username	***Attribute Name***	***Attribute Value***
Admin	Dept	Engineering
Admin	Email	admin@org.com
Tester	Role	Testing
Tester	Dept	QA
Tester	Email	tester@org.com
Dev	Role	Developer
Dev	Dept	Engineering
Dev	Email	dev@org.com

If the user's attributes are structured as column names in the database table, select the **attributes as column names** radio button. If the attributes are structured as name-value pair in table rows, select the **attribute name-value pairs in rows** option.

Configuring ICAP Servers

Overview

The Internet Content Adaptation Protocol (ICAP) is a lightweight HTTP-based protocol used to optimize proxy servers, which frees up resources and standardizes how features are implemented. For example, ICAP is typically used to implement features such as virus scanning, content filtering, ad insertion, or language translation in the HTTP proxy server cache. *Content Adaptation* refers to performing a specific value-added service (for example, virus scanning) for a specific client request and/or response.

You can configure ICAP Servers under the **External Connections** tree node, which you can then specify in an **ICAP** filter later. To configure an ICAP Server, right-click the **ICAP Servers** node, and select **Add an ICAP Server** to display the **ICAP Server Settings** dialog.

General Settings

Configure the following general setting:

Name:
Enter an appropriate name for the ICAP server.

Server Settings

Configure the following settings on the **Server** tab:

Host	The machine name or IP address of the remote ICAP host. Defaults to the localhost (`127.0.0.1`).
Port	The port on which the ICAP server is listening. Defaults to `1344`.
Request Service	The path to the service (exposed by the ICAP server) that handles Request Modification (REQMOD) requests. The default value is `/request`.
Response Service	The path to the service exposed by the ICAP server that handles Response Modification (RESPMOD) requests. The default value is `/response`.
Options Service	The path to the service (exposed by the ICAP server) that handles OPTIONS requests. OPTIONS requests enable server capabilities to be queried. The default value is `/options`.

Security Settings

The following settings on the **Security** tab enable you to secure the connection to the ICAP server:

Trusted Certificates
When the API Gateway connects to the ICAP server over SSL, it must decide whether to trust the ICAP server's SSL certificate. You can select a list of CA or server certificates on the **Trusted Certificates** tab that are considered trusted by API Gateway when connecting to the ICAP server. The table displayed on the **Trusted Certificates** tab lists all certificates imported into the API Gateway Certificate Store. To *trust* a certificate for this particular connection, select the box next to the certificate in the table.

Client SSL Authentication

In cases where the destination ICAP server requires clients to authenticate to it using an SSL certificate, you must select a client certificate on the **Client SSL Authentication** tab. Select the checkbox next to the client certificate that you want to use to authenticate to the ICAP server.

Advanced

The **Ciphers** field enables you to specify the ciphers that API Gateway supports. The API Gateway sends this list of supported ciphers to the destination ICAP server, which then selects the highest strength common cipher as part of the SSL handshake. The selected cipher is then used to encrypt the data when it is sent over the secure channel.

Advanced Settings

Select one of the following ICAP server modes on the **Advanced** tab:

Request Modification Mode (REQMOD)	Specifies that the **ICAP** filter in the API Gateway sends a Request Modification (REQMOD) request to the ICAP server. The ICAP Server returns a modified version of the request, an HTTP response, or a 204 response code indicating that no modification is required. This mode is selected by default.
Response Modification Mode (RESPMOD)	Specifies that the **ICAP** filter in the API Gateway sends a Response Modification (RESPMOD) request to the ICAP server. For example, the API Gateway sends an origin server's HTTP response to the ICAP server. The response from the ICAP server can be an adapted HTTP response, an error, or a 204 response code indicating that no adaptation is required.

Further Information

For more details, see the *ICAP Filter* topic. This topic includes example policies that show an **ICAP** filter configured in REQMOD and RESPMOD modes.

Configuring LDAP Directories

Overview

A filter that uses an LDAP directory to authenticate a user or retrieve attributes for a user must have an LDAP directory associated with it. You can use the **Configure LDAP Server** dialog to configure connection details of the LDAP directory. Both LDAP and LDAPS (LDAP over SSL) are supported.

When a filter that uses an LDAP directory is run for the first time after a server refresh/restart, the server binds to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually, the connection details include the username and password of an administrator user who has read access to all users in the LDAP directory for whom you wish to retrieve attributes or authenticate.

General Configuration

Configure the following general LDAP connection settings:

Name:
Enter or select a name for the LDAP filter in the drop-down list.

URL:
Enter the URL location of the LDAP directory. The URL is a combination of the protocol (LDAP or LDAPS), the IP address of the host machine, and the port number for the LDAP service. By default, port `389` is reserved for LDAP connections, while port `636` is reserved for LDAPS connections. For example, the following are valid LDAP directory URLs:

```
ldap://192.168.0.45:389
ldaps://145.123.0.28:636
```

Cache Timeout:
Specifies the timeout for cached LDAP connections. Any cached connection that is not used in this time period is discarded. Defaults to 300000 milliseconds (5 minutes). A cache timeout of 0 means that the LDAP connection is cached indefinitely and never times out.

Cache Size:
Specifies the number of cached LDAP connections. Defaults to 8 connections. A cache size of 0 means that no caching is performed.

Authentication Configuration

If the configured LDAP directory requires clients to authenticate to it, you must select the appropriate authentication method in the **Authentication Type** field. When the API Gateway connects to the LDAP directory, it is authenticated using the selected method. Choose one of the following authentication methods:

- None
- Simple
- Digest-MD5
- External

Important

If any of the following authentication methods connect to the LDAP server over SSL, that server's SSL certificate must be imported into the API Gateway **Certificate Store**.

None:

No authentication credentials need to be submitted to the LDAP server for this method. In other words, the client connects anonymously to the server. Typically, a client is only allowed to perform read operations when connected anonymously to the LDAP server. It is not necessary to enter any details for this authentication method.

Simple:
Simple authentication involves sending a user name and corresponding password in clear text to the LDAP server. Because the password is passed in clear text to the LDAP server, it is recommended to connect to the server over an encrypted channel (for example, over SSL).

It is not necessary to specify a **Realm** for the Simple authentication method. The realm is only used when a hash of the password is supplied (for Digest-MD5). However, in cases where the LDAP server contains multiple realms, and the specified user name is present in more than one of these realms, it is at the discretion of the specific LDAP server as to which user name *binds* to it.

Click the **SSL Enabled** checkbox to force the API Gateway to connect to the LDAP directory over SSL. To successfully establish SSL connections with the LDAP directory, you must import the directory's certificate into the API Gateway's certificate store. You can do this using the global *Certificates and Keys* screen. For LDAPS (LDAP over SSL) connections, the LDAP server's certificate must be imported into the Policy's Studio's JRE trusted store. For more details, see Testing the Connection.

Digest-MD5:
With *Digest-MD5* authentication, the server generates some data and sends it to the client. The client encrypts this data with its password according to the MD5 algorithm. The LDAP server then uses the client's stored password to decrypt the data and hence authenticate the user.

The **Realm** field is optional, but may be necessary in cases where the LDAP server contains multiple realms. If a realm is specified, the LDAP server attempts to authenticate the user for the specified realm only.

External:
External authentication enables you to use client certificate-based authentication when connecting to an LDAP directory. When this option is selected, you must select a client certificate from the API Gateway certificate store. The **SSL Enabled** checkbox is selected automatically. This means that you must specify the **URL** field using LDAPS (for example, `ldaps://145.123.0.28:636`). The username, password, and realm fields are not required for external authentication.

Testing the LDAP Connection

When you have specified all the LDAP connection details, you can click the **Test Connection** button to verify that the connection to the LDAP directory is configured successfully. This enables you to detect any configuration errors at design time, rather than at runtime.

Important

For LDAPS (LDAP over SSL) connections, the LDAP server's certificate must be imported into the Policy's Studio's JRE trusted store. You can do this by performing the following steps in the Policy Studio:

1. Select the **Certificates and Keys** -> **Certificates** node in the Policy Studio tree.
2. In the **Certificates** panel on the right, click **Create/Import**, and click **Import Certificate**.
3. Browse to the LDAP server's certificate file, and click **Open**.
4. Click **Use Subject** on the right of the **Alias Name** field, and click **OK**. The LDAP server's certificate is now imported into the **Certificate Store**, and must be added to the Java keystore.
5. In the **Certificates** panel, select the certificates that you wish the JRE to trust.
6. Click **Export to Keystore**, and browse to the `cacerts` file in the following directory: `*Policy_Studio_Install*\Win32\jre\lib\security\cacerts`
7. Select the `cacerts` file.
8. Click **Save**.

9. You are prompted for a password. The default password for the JRE is `changeit`.
10. Click **OK**.
11. Restart the Policy Studio.
12. You can now click **Test Connection** to test the connection to the LDAP directory server over SSL.

Additional JNDI Properties

You can also specify optional JNDI properties as simple name-value pairs. Click the **Add** button to specify properties in the dialog.

RADIUS Clients

Overview

The API Gateway provides support for integration with remote systems over the Remote Authentication Dial In User Service (RADIUS) protocol. RADIUS is a client-server network protocol that provides centralized authentication and authorization for clients connecting to remote services. For more details, see the RADIUS specification [http://tools.ietf.org/html/rfc2865].

To configure a client connection to a remote server over the RADIUS protocol, under the **External Connections** tree node in the Policy Studio, select **RADIUS Clients** -> **Add a RADIUS Client**. This topic explains how to configure the settings the **RADIUS Client** dialog.

For details on how to configure a RADIUS Authentication Repository, see the *Authentication Repository* topic.

Configuration

Configure the following fields in the **RADIUS Client** dialog:

- **Name:**
  Enter an appropriate name for the RADIUS client to display in the Policy Studio.
- **Host name:**
  Enter the host name used by the RADIUS client.
- **Client port:**
  Enter the port number used by the RADIUS client.
- **RADIUS servers:**
  This field displays a list of configured RADIUS servers. To add a server to the list, click **Add**, and complete the following fields:

Name	Name of the RADIUS server.
Port	Port number used by the RADIUS server.
Secret	Shared secret used to access the RADIUS server.
Response timeout (sec)	Response timeout in seconds before the connection to the server is closed.
Retransmit count	Number of times retransmission is attempted before the connection to the server fails.

SAML PDP Response XML-Signature Verification

Overview

A SAML assertion contains identity information about an end-user. Depending on its type, the assertion can convey proof of an authentication event, details of user attributes, or authorization information about the end-user. Typically such assertions are issued by a SAML PDP (Policy Decision Point) after a client authenticates to it or requests access to a specified resource.

Clients use the SAML Protocol (SAMLP) to obtain SAML assertions from SAML PDPs. The SAMLP request usually contains identity information about the end-user. The PDP then uses this information when generating the assertion, which is then returned in a SAMLP response.

The PDP will usually *sign* the assertion and/or the SAMLP response as proof that only it could have issued the assertion/ response, and also to guarantee the integrity of the assertion. When the API Gateway receives the SAMLP response from the PDP, it can validate the signature on the assertion or on the response.

Configuration

Configure the following fields to validate the XML Signature over the SAMLP response:

Signature Location:
Because there may be multiple signatures contained within the SAMLP response, it is necessary to specify which signature the API Gateway should validate. The signature can be extracted from one of three places:

- From the SOAP header
- Using WS-Security Actors
- Using XPath

Select the appropriate option from the dropdown. Take a look at the *Signature Location* tutorial for more information on configuring this section.

What Must Be Signed:
This section defines the content that must be signed in order for the signature on the SAMLP response to be considered valid. This ensures that the client has signed something meaningful (i.e. part of the SAMLP response) as opposed to some arbitrary data that would pass a "blind" signature validation.

An XPath expression is used to identify the nodeset that should be signed. To specify that nodeset, select either an existing XPath expression from the **XPath Expression** dropdown list, or add a new one using the **Add** button. XPath expressions can also be edited or removed with the **Edit** and **Delete** buttons respectively.

Signer's Public Key/Certificate
Select the **Certificate in Message** radio button in order to use the certificate from the XML-Signature specified in the **Signature Location** section. The certificate will be extracted from the `<KeyInfo>` block.

```
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="Sample">
  ...
  <dsig:KeyInfo>
    <dsig:X509Data>
      <dsig:X509SubjectName>CN=Sample User...</dsig:X509SubjectName>
      <dsig:X509Certificate>
        MIIE ....... EQgJ
      </dsig:X509Certificate>
    </dsig:X509Data>
    <dsig:KeyValue>
      <dsig:RSAKeyValue>
        <dsig:Modulus>
          AMfb2tT53GmMiD
          ...
          NmrNht7iy18=
        </dsig:Modulus>
        <dsig:Exponent>AQAB</dsig:Exponent>
       </dsig:RSAKeyValue>
    </dsig:KeyValue>
  </dsig:KeyInfo>
</dsig:Signature>
```

Clients may not always want to include their public keys in their signatures. In such cases, the public key must be retrieved from a certificate stored either in a specified LDAP directory or in the the API Gateway's global Certificate Store.

For example, the following signed XML message does not include the signatory's certificate. Instead only the *Common Name* of the signatory's certificate is included. In this case, the API Gateway must obtain the certificate from an LDAP directory or the Certificate Store to validate the signature on the assertion.

```
<?xml version="1.0" encoding="UTF-8"?>
 <soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="User">
    <dsig:SignedInfo>
     <dsig:CanonicalizationMethod
                   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     <dsig:SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <dsig:Reference URI="">
      <dsig:Transforms>
       <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <dsig:XPath>ancestor-or-self::soap-env:Body</dsig:XPath>
       </dsig:Transform>
       <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
      </dsig:Transforms>
      <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <dsig:DigestValue>rvJMkZ1RDo3pNfqCUBa4Qhs8i+M=</dsig:DigestValue>
     </dsig:Reference>
    </dsig:SignedInfo>
    <dsig:SignatureValue>
      AXL2gKhqqKwcKujVPftVoztySvtCdARGf97Cjt6Bbpf0w8QFiNuLJncQVnKB
      cQ+91KvudYZ/Sk8u7tXhoEiLvNwg76B2STPh+ypEWO+J7OSPedlUdnfVRRvW
      vjYLwJVjGNZ+mMTxvfO1wwcIb2Hg94n1BOaeBrNJ+2uO4i87W5TyufAGI+V8
      S6oSpPc5KQeHLXoyHS2+fXyqReSiwdhOeli4D4xT+HbjRgYJIwIikXn2k1Fr
      D/hnd1/xVf/LjrOwoY9id8W3IcZAzMIRh5SBZjWHYOQzk79xy4YDpzNVYIOB
      laAFqzg9G+Z4VYj+RdgrIVHhOXt+mq+fGZV6VheWGQ==
    </dsig:SignatureValue>
    <dsig:KeyInfo>
     <dsig:KeyName>
       CN=User,OU=R&D,O=Company Ltd.,L=Dublin 4,ST=Dublin,C=IE
     </dsig:KeyName>
    </dsig:KeyInfo>
   </dsig:Signature>
  </soap-env:Header>
 <soap-env:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </soap-env:Body>
</soap-env:Envelope>
```

To retrieve a client certificate from an LDAP directory, select a pre-configured one from the **LDAP Source** dropdown, or add/edit a new/existing LDAP directory by clicking the **Add/Edit** button.

Alternatively, select a certificate from the Certificate Store by selecting the **Certificate in Store** radio button and clicking on the **Select** button. This certificate will then be associated with the incoming message so that all subsequent certificate-based filters will use this user's certificate.

Signature Location

Overview

A given XML message can contain several XML Signatures. Consider an XML document (for example, a company policy approval form) that must be digitally signed by a number of users (for example, department managers) before being submitted to the ultimate Web Service (for example, a company policy approval Web Service). Such a message will contain several XML Signatures by the time it is ready to be submitted to the Web Service.

In such cases, where multiple signatures will be present within a given XML message, it is necessary to specify which signature the API Gateway should use in the validation process.

Configuration

The API Gateway can extract the signature from an XML message using several different methods. The signature can be extracted:

- Using WS-Security Actors
- From the SOAP header
- Using XPath

Select the most appropriate method from the **Signature Location** dropdown. Your selection will depend on the types of SOAP messages that you expect to receive. For example, if incoming SOAP messages will contain an XML Signature within a WS-Security block, you should choose this option from the dropdown.

Using WS-Security Actors:
If the signature is present in a WS-Security block:

1. Select *WS-Security block* from the **Signature Location** dropdown list.
2. Select a SOAP Actor from the **Select Actor/Role(s)** dropdown. Each Actor uniquely identifies a separate WS-Security block. By selecting *Current actor only* from the dropdown, the WS-Security block with no Actor will be taken.
3. In cases where there may be multiple signatures within the WS-Security block, it is necessary to extract one using the **Signature Position** field.

The following is a skeleton version of a message where the XML Signature is contained in the *sample* WS-Security block, (`soap-env:actor="sample"`):

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
       s:actor="sample">
   <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
    ....
   </dsig:Signature>
  </wsse:Security>
 </s:Header>
 <s:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </s:Body>
</s:Envelope>
```

SOAP Header:
If the signature is present in the SOAP Header:

1. Select *SOAP message header* from the **Signature Location** dropdown list.
2. If there is more than one signature in the SOAP Header, then it is necessary to specify which signature the API Gateway should use. Specify the appropriate signature by setting the **Signature Position** field.

The following is an example of an XML message where the XML Signature is contained within the SOAP header:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
      ....
  </dsig:Signature>
 </s:Header>
 <s:Body>
  <ns1:getTime xmlns:ns1="urn:timeservice">
  </ns1:getTime>
 </s:Body>
</s:Envelope>
```

Using XPath:
Finally, an XPath expression can be used to locate the signature.

1. Select *Advanced (XPath)* from the **Signature Location** dropdown list.
2. Select an existing XPath expression from the dropdown, or add a new one by clicking on the **Add** button. XPath expressions can also be edited or removed with the **Edit** and **Remove** buttons respectively.

The default *First Signature* XPath expression takes the first signature from the SOAP Header. The expression is as follows:

XPath Expression:	
//s:Envelope/s:Header/dsig:Signature[1]	

To edit this expression, click the **Edit** button to display the **Enter XPath Expression** dialog.

An example of a SOAP message containing an XML Signature in the SOAP header is provided below. The following XPath expression instructs the API Gateway to extract the first signature from the SOAP header:

XPath Expression:	
//s:Envelope/s:Header/dsig:Signature[1]	

Because the elements referenced in the expression (`Envelope` and `Signature`) are *prefixed* elements, you must define the namespace mappings for each of these elements as follows:

Prefix	***URI***
s	http://schemas.xmlsoap.org/soap/envelope/
dsig	http://www.w3.org/2000/09/xmldsig#

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
    ....
  </dsig:Signature>
 </s:Header>
 <s:Body>
  <product xmlns="http://www.oracle.com">
   <name>SOA Product*</name>
   <company>Company</company>
   <description>Web Services Security</description>
  </product>
 </s:Body>
</s:Envelope>
```

When adding your own XPath expressions, you must be careful to define any namespace mappings in a manner similar to that outlined above. This avoids any potential clashes that might occur where elements of the same name, but belonging to different namespaces are present in an XML message.

SMTP Servers

Overview

You can configure the API Gateway to use a specified Simple Mail Transfer Protocol (SMTP) server to relay messages to an email recipient. You can configure the SMTP server as a global configuration item under the **External Connections** node. The SMTP server is then available for selection in the **SMTP** filter in the **Routing** category.

To configure an SMTP server, right-click the **External Connections** -> **SMTP Servers** node, and select **Add an SMTP Server**. Alternatively, in the **SMTP** filter screen, click the button beside the **SMTP Server Settings** field, right-click the **SMTP Servers** node, and select **Add an SMTP Server**.

Configuration

Configure the following fields in the **SMTP Server settings** dialog:

Name:
Enter an appropriate name for the SMTP server.

SMTP Server Settings
Specify the following fields:

- **SMTP Server Hostname:**
  Enter the hostname or IP address of the SMTP server.
- **Port:**
  Enter the SMTP port on the specified server hostname. By default, SMTP servers run on port 25.
- **Connection Security:**
  Select the security required for the connection to the SMTP server (`SSL`, `TLS`, or `NONE`). Defaults to `NONE`.

Log on using
If you are required to authenticate to the SMTP server, specify the following fields:

- **User Name:**
  Enter the user name of a registered user that can access the SMTP server.
- **Password:**
  Enter the password for the user name specified.

Note

The SMTP server connection supports a simple username/password type of authentication. Microsoft Windows NT LAN Manager (NTLM) authentication is not supported.

Configuring a Transparent Proxy

Overview

On Linux systems with the `TPROXY` kernel option enabled, you can configure the API Gateway as a *transparent proxy*. This enables the API Gateway to present itself as having the server's IP address from the point of view of the client, and/or having the client's IP address from the point of view of the server. This can be useful for administrative or network infrastructure purposes (for example, to keep using existing client/server IP addresses, and for load-balancing).

You can configure transparent proxy mode both for inbound and outbound API Gateway connections:

- Incoming interfaces can listen on IP addresses that are not assigned to any interface on the local host.
- Outbound calls can present the originating client's IP address to the destination server.

Both of these options act independently of each other.

Configuring Transparent Proxy Mode for Incoming Interfaces

To enable transparent proxy mode on an incoming interface, perform the following steps:

1. In the Policy Studio tree view, expand the **Listeners** -> **Oracle API Gateway** nodes.
2. Right-click your service, and select **Add Interface** -> **HTTP** or **HTTPS** to display the appropriate dialog (for example, **Configure HTTP Interface**).
3. Select the checkbox labeled **Transparent Proxy (allow bind to foreign address)**. When selected, the value in the **Address** field can specify any IP address, and incoming traffic for the configured address/port combinations is handled by the API Gateway.

For more details on configuring interfaces, see *Configuring HTTP Services*.

Configuring Transparent Proxy Mode for Outgoing Calls

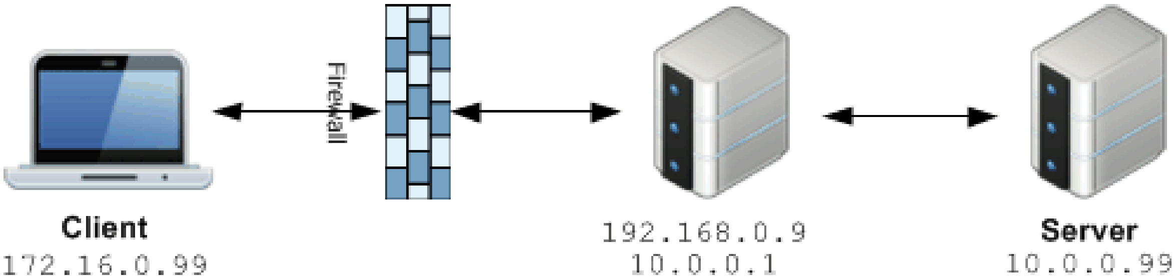
Transparent proxy mode for outgoing calls must be enabled at the level of a connection filter in a policy. To enable transparent proxy mode for outbound calls, perform the following steps:

1. Ensure that your policy contains a connection filter (for example, **Connect to URL** or **Connection**, available from the **Routing** category in the filter palette).
2. In your connection filter, select the **Advanced** tab.
3. Select the checkbox labeled **Transparent Proxy (present client's IP address to server)**. When selected, the IP address of the original client connection that caused the policy to be invoked is used as the local address of the TCP connection to the destination server.

For more details on configuring connection filters, see *Connection* and *Connect to URL* .

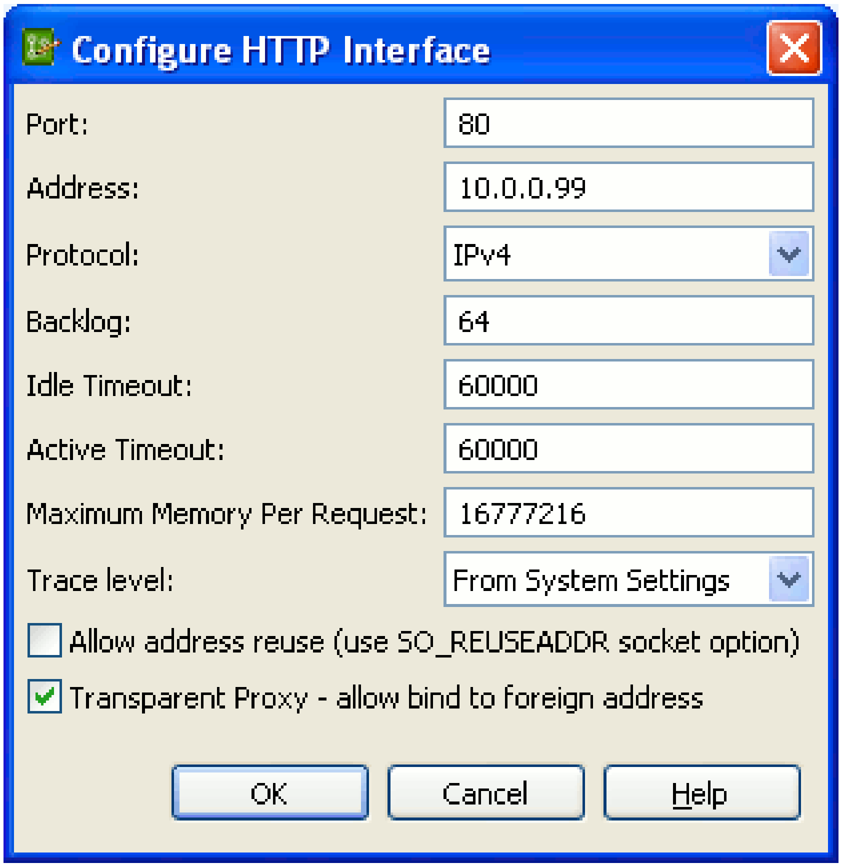
Configuration Example

A typical configuration example of transparent proxy mode is shown as follows:



In this example, the remote client's address is `172.16.0.99`, and it is attempting to connect to the server at `10.0.0.99`, port `80`. The front-facing firewall is configured to route traffic for `10.0.0.99` through the API Gateway at address `192.168.0.9`. The server is configured to use the API Gateway at address `10.0.0.1` as its default IP router.

The API Gateway is multi-homed, and sits on both the `192.168.0.0/24` and `10.0.0.0/24` networks. It is configured with a listening interface at address `10.0.0.99:80`, with transparent proxy mode switched on, as shown in the following **Configure HTTP Interface** dialog:



Port:	80
Address:	10.0.0.99
Protocol:	IPv4
Backlog:	64
Idle Timeout:	60000
Active Timeout:	60000
Maximum Memory Per Request:	16777216
Trace level:	From System Settings

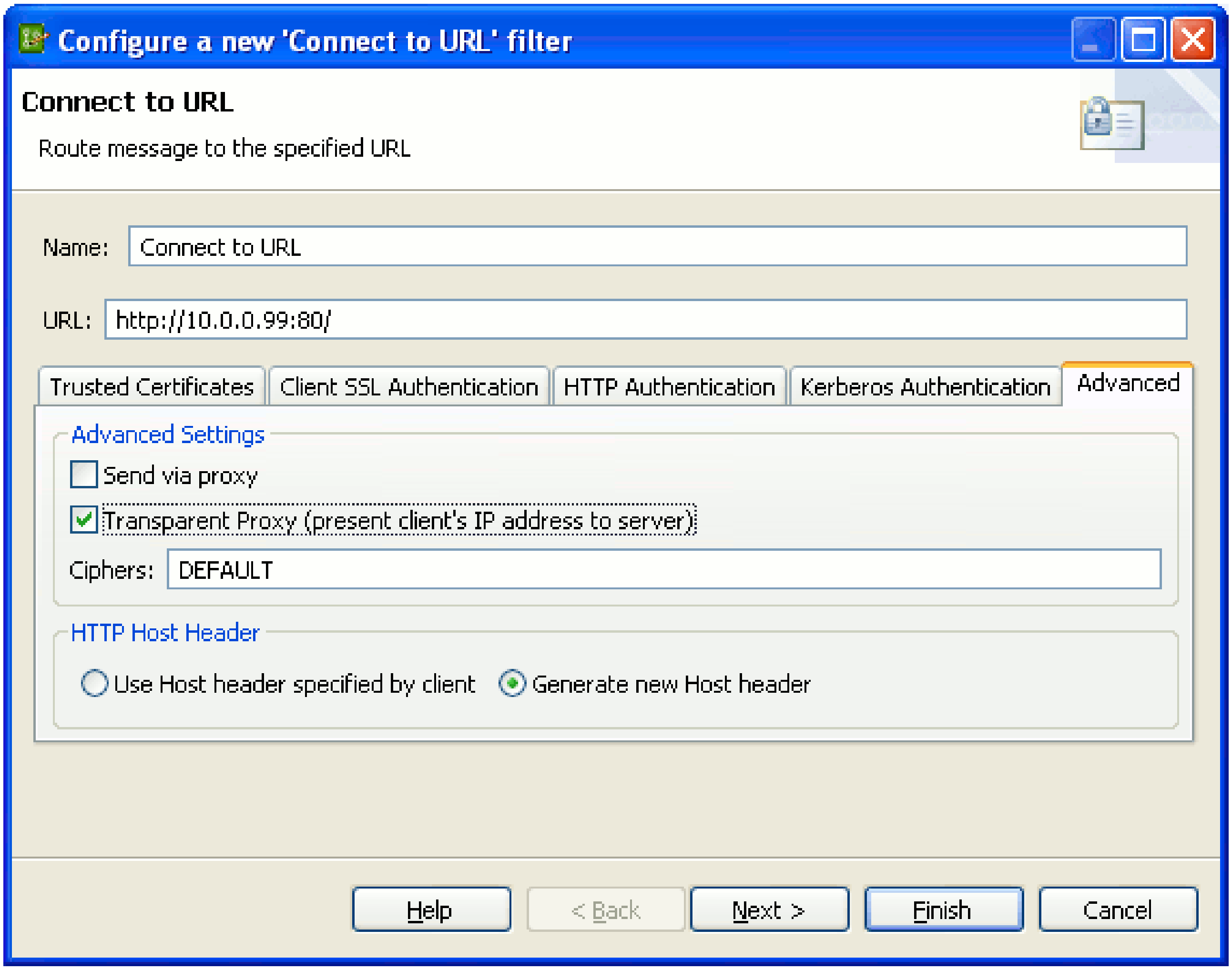
□ Allow address reuse (use SO_REUSEADDR socket option)

☑ Transparent Proxy - allow bind to foreign address

The API Gateway accepts the incoming call from the client, and processes it locally. However, there is no communication with the server yet. The API Gateway can process the call to completion and respond to the client—it is masquerading as the server.

If the API Gateway invokes a connection filter when processing this call (with transparent proxying enabled), the connection filter consults the originating address of the client, and binds the local address of the new outbound connection to that address before connecting. The server then sees the incoming call on the API Gateway originating from the client

`172.16.0.99`), rather than either of the API Gateway's IP addresses. The following dialog shows the example configuration for the **Connect to URL** filter:



Connect to URL

Route message to the specified URL

Name: Connect to URL

URL: http://10.0.0.99:80/

Advanced Settings

□ Send via proxy

☑ Transparent Proxy (present client's IP address to server)

Ciphers: DEFAULT

HTTP Host Header

○ Use Host header specified by client ● Generate new Host header

The result is a transparent proxy, where the client sees itself as connecting directly to the server, and the server sees an incoming call directly from the client. The API Gateway processes two separate TCP connections, one to the client, one to the server, with both masquerading as the other on each connection.

Note

Either side of the transparent proxy is optional. By configuring the appropriate settings for the incoming interface or the connection filter, you can masquerade only to the server, or only to the client.

Retrieving WSDL Files from a UDDI Registry

Overview

A Web Services Description Language (WSDL) file defines the interface to a Web Service, or list of services. It lists the operations exposed by the service, the wire format for operation requests, and the data types in the request body. It also specifies the location of the Web Service as a URL that clients can access to use the exposed operations. For example, the Policy Studio can extract this information from the WSDL file to generate the following filters, which can be incorporated into policies:

- **Relative Path Resolver**
- **SOAPAction Resolver**
- **SOAP Operation Resolver**
- **Connection Handler**
- **Static Router**
- **Schema Validation**

You can use WSDL files to register Web Services in the **Web Services Repository** and to add XML Schemas to the global **Schema Cache**. The Policy Studio can retrieve a WSDL file from the file system, from a URL, or from a UDDI registry. This topic explains how to retrieve a WSDL file from a UDDI registry. For details on how to register WSDL files, see *Web Service Repository*. For details on how to publish WSDL files, see *Publishing WSDL Files to a UDDI Registry*.

You can also browse a UDDI registry in the Policy Studio directly without registering a WSDL file. Under the **Business Services** node in the tree, right-click the **Web Services Repository** node, and select **Browse UDDI Registry**.

UDDI: A Brief Introduction

Universal Description, Discovery and Integration (UDDI) is an OASIS-led initiative that enables businesses to publish and discover Web Services on the Internet. A business publishes services that it provides to a public XML-based registry so that other businesses can dynamically look up the registry and discover these services. Enough information is published to the registry to enable other businesses to find services and communicate with them. In addition, businesses can also publish services to a private or semi-private registry for internal use.

A business registration in a UDDI registry includes the following components:

- **Green Pages:**
  Contains technical information about the services exposed by the business
- **Yellow Pages:**
  Categorizes the services according to standard taxonomies and categorization systems
- **White Pages:**
  Gives general information about the business, such as name, address, and contact information

You can search the UDDI registry according to a whole range of search criteria, which is of key importance to the Policy Studio. You can search the registry to retrieve the WSDL file for a particular service. The Policy Studio can then use this WSDL file to create a policy for the service, or to extract a schema from the WSDL to check the format of messages attempting to use the operations exposed by the Web Service.

For a more detailed description of UDDI, see the UDDI specification. In the meantime, the next section gives high-level definitions of some of the terms displayed in the Policy Studio interface.

UDDI Definitions

Because UDDI terminology is used in Policy Studio screens such as the **Import WSDL** wizard, the following list of definitions explains some common UDDI terms. For more detailed explanations, see the UDDI specification.

businessEntity
This represents all known information about a particular business (for example, name, description, and contact information). A businessEntity can contain a number of businessService entities. A businessEntity may have an identifierBag, which is a list of name-value pairs for identifiers, such as Data Universal Numbering System (DUNS) numbers or taxonomy identifiers. A businessEntity may also have a categoryBag, which is a list of name-value pairs used to tag the businessEntity with classification information such as industry, product, or geographic codes. There is no mapping for a WSDL item to a businessEntity. When a WSDL file is published, you must specify a businessEntity for the businessService.

businessService
A businessService represents a logical service classification, and is used to describe a Web Service provided by a business. It contains descriptive information in business terms outlining the type of technical services found in each businessService element. A businessService may have a categoryBag, and may contain a number of bindingTemplate entities. In the WSDL to UDDI mapping, a businessService represents a wsdl:service. A businessService has a businessEntity as its parent in the UDDI registry.

bindingTemplate
A bindingTemplate contains pointers to the technical descriptions and the access point URL of the Web Service, but does not contain the details of the service specification. A bindingTemplate may contain references to a number of tModel entities, which do include details of the service specification. In the WSDL to UDDI mapping, a bindingTemplate represents a wsdl:port.

tModel
A tModel is a Web service type definition, which is used to categorize a service type. A tModel consists of a key, a name, a description, and a URL. tModels are referred to by other entities in the registry. The primary role of the tModel is to represent a technical specification (for example, WSDL file). A specification designer can establish a unique technical identity in a UDDI registry by registering information about the specification in a tModel. Other parties can express the availability of Web services that are compliant with a specification by including a reference to the tModel in their bindingTemplate data.

This approach facilitates searching for registered Web services that are compatible with a particular specification. tModels are also used in identifierBag and categoryBag structures to define organizational identity and various classifications. In this way, a tModel reference represents a relationship between the keyed name-value pairs to the supername, or namespace in which the name-value pairs are meaningful. A tModel may have an identifierBag and a categoryBag. In the WSDL to UDDI mapping, a tModel represents a wsdl:binding or wsdl:portType.

Identifier
The purpose of identifiers in a UDDI registry is to enable others to find the published information using more formal identifiers such as DUNS numbers, Global Location Numbers (GLN), tax identifiers, or any other kind of organizational identifiers, regardless of whether these are private or shared.

The following are identification systems used commonly in UDDI registries:

Identification System	*Name*	*tModel Key*
Dun and Bradstreet D-U-N-S Number	dnb-com:D-U-N-S	uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823
Thomas Registry Suppliers	thomasregister-com:supplierID	uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039

Category
Entities in the registry may be categorized according to categorization system defined in a tModel (for example, geographical region). The businessEntity, businessService, and tModel types have an optional categoryBag. This is a collection of categories, each of which has a name, value, and tModel key.

The following are categorization systems used commonly in UDDI registries:

Categorization System:	*Name:*	*tModel Key:*
UDDI Type Taxonomy	`uddi-org:types`	`uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4`
North American Industry Classification System (NAICS) 1997 Release	`ntis-gov:naics:1997`	`uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2`

Example tModel Mapping for WSDL portType
The following shows an example `tModel` mapped for a WSDL `portType`:

```
<tModel tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3" >
    <name>
       StockQuotePortType
    </name>
    <overviewDoc>
         <overviewURL>
             http://location/sample.wsdl
         <overviewURL>
    <overviewDoc>

    <categoryBag>
         <keyedReference
             tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
         keyName="portType namespace"
             keyValue="http://example.com/stockquote/" />
         <keyedReference
             tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
         keyName="WSDL type"
             keyValue="portType" />
    </categoryBag>
</tModel>
```

In this example, the `tModel` name is the same as the local name of the WSDL `portType` (in this case, `StockQuotePortType`), and the `overviewURL` links to the WSDL file. The `categoryBag` specifies the WSDL namespace, and shows that the `tModel` is for a `portType`.

Configuring a Registry Connection

You first need to select the UDDI registry that you want to search for WSDL files. Complete the following fields to select or add a UDDI registry:

Select Registry:
Select an existing UDDI registry to browse for WSDL files from the **Registry** drop-down list. To configure the location of a new UDDI registry, click **Add**. Similarly, to edit an existing UDDI registry location, click **Edit**. Then configure the fields in the **Registry Connection Details** dialog. For more details, see *Connecting to a UDDI Registry*.

Select Locale:
You can select an optional language locale from this list. The default is `No Locale`.

WSDL Search

When you have configured a UDDI registry connection, you can search the registry using a variety of different search options on the **Search** tab. **WSDL Search** is the default option. This enables you to search for the UDDI entries that the

WSDL file is mapped to. You can also do this using the **Advanced Search** option. The following different types of WSDL searches are available:

WSDL portType (UDDI tModel):
Searches for a `uddi:tModel` that corresponds to a `wsdl:portType`. You can enter optional search criteria for specific categories in the `uddi:tModel` (for example, **Namespace of portType**).

WSDL binding (UDDI tModel):
Searches for a `uddi:tModel` that corresponds to a `wsdl:binding`. You can enter optional search criteria for specific categories in the `uddi:tModel` (for example, **Name of binding**, or **Binding Transport Type**).

WSDL service (UDDI businessService):
Searches for a `uddi:businessService` that corresponds to a `wsdl:service`. You can enter optional search criteria for specific categories in the `uddi:businessService` (for example, **Namespace of service**).

WSDL port (UDDI bindingTemplate):
Searches for a `uddi:bindingTemplate` that corresponds to a `wsdl:port`. This search is more complex because a `serviceKey` is required to find a `uddi:bindingTemplate`. This means that at least two queries are carried out, first to find the `uddi:businessService`, and another to find the `uddi:bindingTemplate`.

For example, a `bindingTemplate` contains a reference to the `tModel` for the `wsdl:portType`. You can use the `tModel` key to find all implementations (`bindingTemplates`) for that `wsdl:portType`. The search looks for `businessServices` that have `bindingTemplates` that refer to the `tModel` for the `wsdl:portType`. Then with the `serviceKey`, you can find the `bindingTemplate` that refers to the `tModel` for the `wsdl:portType`.

In all cases, click **Next** to start the WSDL search. The **Search Results** tree shows the `tModel` URIs as top-level nodes. These URIs are all WSDL URIs, and you can use these to generate policies on import by selecting the URI, and clicking the **Finish** button.

You can click any of the nodes in the tree to display detailed properties about that node in the table below the **Search Results** tree. The properties listed depend on the type of the node that is selected. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Quick Search

The **Quick Search** option enables you to search the UDDI registry for a specific `tModel` name or category.

tModel Name:
You can enter a **tModel Name** for a fine-grained search. This is a partial or full name pattern with wildcard searching as specified by the *SQL-92 LIKE* specification. The wildcard characters are percent `%`, and underscore `_`, where an underscore matches any single character and a percent matches zero or more characters.

Categories:
You can select one of the following options to search by:

wsdlSpec	Search for `tModels` classified as `wsdlSpec` (`uddi-org:types` category set to `wsdlSpec`). This is the default.
Reusable WS-Policy Expressions	Search for `tModels` classified as reusable WS-Policy Expressions.
All	Search for all `tModels`.

Click **Next** to start the search. The **Search Results** tree shows the `tModel` URIs as top-level nodes. These URIs are all WSDL URIs, and you can use these to generate policies on import by selecting the URI, and clicking the **Finish** button.

You can click any of the nodes in the tree to display detailed properties about that node in the table below the **Search**

Results tree. The properties listed depend on the type of the node that is selected. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Name Search

The **Name Search** option enables you to search for a `businessEntity`, `businessService` or `tModel` by name. In the **Select Registry Data Type**, select one of the following UDDI entity levels to search for:

- **businessEntity**
- **businessService**
- **tModel**

You can enter a name in the **Name** field to narrow the search. You can also use wildcards in the name. The name applies to a `businessEntity`, `businessService`, or `tModel`, depending on which registry entity type has been selected. If no name is entered, all entities of the selected type are retrieved.

Click the **Search** button to start the search. The search results display the matching entities in the tree. For example, if you enter `MyTestBusiness` for **Name**, and select **businessEntity**, this searches for a `businessEntity` with the name `MyTestBusiness`. Child nodes of the matching `businessEntity` nodes are also shown. `tModels` are displayed in the results if any child nodes of the `businessEntity` refer to `tModels`. Only referred to `tModels` are displayed. The same applies if you search for a `businessService`. If you select `tModel`, and search for `tModels`, only `tModels` are displayed.

Important

The `tModel` URIs shown in the resulting tree may not all be categorized as `wsdlSpec` according to the `uddi-org:types` categorization system. You can choose to load any of these URIs as a WSDL file, but you are warned if it is not categorized as `wsdlSpec`.

As before, you can click any node in the results tree to display properties about that node in the table. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

UDDI v3 Name Searches
By default, a UDDI v3 name search is an exact match. To perform a search using wildcards (for example, `%`, `_`, and so on), you must select the **approximateMatch** find qualifier in the **Advanced Options** tab. This applies to anywhere you enter a name for search purposes (for example, **Name Search**, **Quick Search**, and **Advanced Search**).

Advanced Search

The **Advanced Search** option enables you to search the UDDI registry using any combination of **Names**, **Keys**, **tModels**, **Discovery URLs**, **Categories**, and **Identifiers**. You can also specify the entity level to search for in the tree. All of these options combine to provide a very powerful search facility. You can specify search criteria for any of the categories listed above by right-clicking the folder node in the **Enter Search Criteria** tree, and selecting the **Add** menu option. You can enter more than one search criteria of the same type (for example, two **Key** search criteria).

Important

The `tModel` URIs shown in the resulting tree may not all be categorized as `wsdlSpec` according to the `uddi-org:types` categorization system. You can choose to load any of these URIs as a WSDL file, but you are warned if it is not categorized as `wsdlSpec`.

The following options enable you to add a search criteria for each of the types listed in the **Enter Search Criteria** tree. All search criteria are configured by right-clicking the folder node, and selecting the **Add** menu option.

Names:
Enter a name to be used in the search in the **Name** field in the **Name Search Criterion** dialog. For example, the name could be the **businessEntity** name. The name is a partial or full name pattern with wildcards allowed as specified by the *SQL-92 LIKE* specification. The wildcard characters are percent `%`, and underscore `_`, where an underscore matches any single character and a percent matches zero or more characters. A name search criterion can be used for `businessEntity`, `businessService`, and `tModel` level searches.

Keys:
In the **Key Search Criterion** dialog, you can specify a key to search the registry for in the **Key** field. The key value is a Universally Unique Identifier (UUID) value for a registry object. You can use the **Key Search Criterion** on all levels of searches. If one or more keys are specified with no other search criteria, the keys are interpreted as the keys of the selected type of registry object and used for a direct lookup, instead of a find/search operation. For example, if you enter `key1` and `key2`, and select the `businessService` entity type, the search retrieves the `businessService` object with key `key1`, and another `businessService` with key `key2`.

If you enter a key with other search criteria, a key criterion is interpreted as follows:

- For a businessService entity lookup, the key is the `businessKey` of the services.
- For a `bindingTemplate` entity lookup, the key is the `serviceKey` of the binding templates.
- Not applicable for any other object type.

tModels:
You can enter a key in the **tModel Key** field on the **tModel Search Criterion** screen. The key entered should correspond to the UUID of the `tModel` associated with the type of object you are searching for. A `tModel` search criterion may be used for `businessEntity`, `businessService`, and `bindingTemplate` level searches.

Discovery URLs:
Enter a URL in the **Discovery URL** field on the **Discovery URL Search Criterion** dialog. The **Use Type** field is optional, but can be used to further fine-grain the search by type. You can use a Discovery URL search criterion for `businessEntity` level searches only.

Categories:
Select a previously configured categorization system from the **Type** drop-down list in the **Category Search Criterion** dialog. This pre-populated with a list of common categorization systems. You can add a new categorization system using the **Add** button.

In the **Add/Edit Category** dialog, enter a **Name**, **Description**, and **UUID** for the new category type in the fields provided. When the categorization system has been selected or added, enter a value to search for in the **Value** field. The **Name** field is optional.

Identifiers:
Select a previously configured identification system from the **Type** drop-down list in the **Identifier Search Criterion** dialog. This is pre-populated with well-known identification systems. To add a new identification system, click the **Add** button.

In the **Add/Edit Identifier** dialog, enter a **Name**, **Description**, and **UUID** for the new identifier in the fields provided.

Select Registry Data Type:
Select one of the following UDDI entity levels to search for:

- **businessEntity**
- **businessService**
- **bindingTemplate**
- **tModel**

The search also displays child nodes of the matching nodes. `tModels` are also returned if they are referred to.

Advanced Options

This tab enables you to configure various aspects of the search conditions specified on the previous tabs. The following options are available:

UDDI Find Qualifier:	*Description:*
andAllKeys	By default, identifier search criteria are ORed together. This setting ensures that they are ANDed instead. This is already the default for `categoryBag` and `tModelBag`.
approximateMatch (v3)	This applies to a UDDI v3 registry only. Specifies wildcard searching as defined by the `uddi-org:approximatematch:SQL99 tModel`, which means approximate matching where percent sign (%) indicates any number of characters, and underscore (_) indicates any single character. The backslash character (\) is an escape character for the percent sign, underscore and backslash characters. This option adjusts the matching behavior for `name`, `keyValue` and `keyName` (where applicable).
binarySort (v3)	This applies to a UDDI v3 registry only. Enables greater speed in sorting, and causes a binary sort by name, as represented in Unicode codepoints.
bindingSubset (v3)	This applies to a UDDI v3 registry only. Specifies that the search uses only `categoryBag` elements from contained `bindingTemplate` elements in the registered data, and ignores any entries found in the `categoryBag` that are not direct descendents of registered `businessEntity` or `businessService` elements.
caseInsensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that that the matching for `name`, `keyValue` and `keyName` (where applicable) should be performed without regard to case.
caseInsensitiveSort (v3)	This applies to a UDDI v3 registry only. Specifies that the result set should be sorted without regard to case. This overrides the default case sensitive sorting behavior.
caseSensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that that the matching for `name`, `keyValue` and `keyName` (where applicable) should be case sensitive. This is the default behavior.
caseSensitiveSort (v3)	This applies to a UDDI v3 registry only. Specifies that the result set should be sorted with regard to case. This is the default behavior.
combineCategoryBags	Makes the `categoryBag` entries of a `businessEntity` behave as if all `categoryBags` found at the `businessEntity` level and in all contained or referenced `businessServices` are combined. Searching for a category yields a positive match on a registered business if any of the `categoryBags` contained in a `businessEntity` (including the `categoryBags` in contained or referenced `businessServices`) contain the filter criteria.
diacriticInsensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that matching for `name`, `keyValue` and `keyName` (where applicable) should be performed without regard to diacritics.

	Support for this qualifier by nodes is optional.
diacriticSensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that matching for `name`, `keyValue` and `keyName` (where applicable) should be performed with regard to diacritics. This is the default behavior.
exactMatch (v3)	This applies to a UDDI v3 registry only. Specifies that only entries with `name`, `keyValue` and `keyName` (where applicable) that exactly match the name argument passed in, after normalization, are returned. This qualifier is sensitive to case and diacritics where applicable. This is the default behavior.
exactNameMatch (v2)	This applies to a UDDI v2 registry only. Specifies that the name entered as part of the search criteria must exactly match the name specified in the UDDI registry.
orAllKeys	By default, `tModel` and category search criteria are ANDed. This setting ORs these criteria instead.
orLikeKeys	When a bag container contains multiple `keyedReference` elements (`categoryBag` or `identifierBag`), any `keyedReference` filters from the same namespace (for example, with the same `tModelKey` value) are ORed together rather than ANDed. For example, this enables you to search for `any of these four values from this namespace, and any of these two values from this namespace`.
serviceSubset	Causes the component of the search that involves categorization to use only the `categoryBags` from directly contained or referenced `businessServices` in the registered data. The search results return only those businesses that match based on this modified behavior, in conjunction with any other search arguments provided.
signaturePresent (v3)	This applies to a UDDI v3 registry only. This restricts the result to entities that contain, or are contained in, an XML Digital `Signature` element. The `Signature` element should be verified by the client. This option, or the presence of a `Signature` element, should only be used to refine a search result, and should not be used as a verification mechanism by UDDI clients.
sortByDateAsc (v3)	This applies to a UDDI v3 registry only. Sorts the results alphabetically in order of ascending date.
sortByDateDsc (v3)	This applies to a UDDI v3 registry only. Sorts the results alphabetically in order of descending date.
sortByNameAsc	Sorts the results alphabetically in order of ascending name.
sortByNameDsc	Sorts the results alphabetically in order of descending name.
suppressProjectedServices (v3)	This applies to a UDDI v3 registry only. Specifies that service projections must not be returned when searching for services or businesses. This option is enabled by default when searching for a service without a `businessKey`.
UTS-10 (v3)	This applies to a UDDI v3 registry only. Specifies sorting of results based on the Unicode Collation Algorithm on elements normalized according to Unicode Normalization

	Form C. A sort is performed according to the Unicode Collation Element Table in conjunction with the Unicode Collation Algorithm on the name field, and normalized using Unicode Normalization Form C. Support for this qualifier by nodes is optional.

Publish

Click the **Publish** radio button to view the **Published UDDI Entities Tree View**. This enables you to manually publish UDDI entities to the specified UDDI registry (for example, `businessEntity`, `businessService`, `bindingTemplate`, and `tModel` entities). You must already have the appropriate permissions to write to the UDDI registry. **Adding a businessEntity**
To add a business, perform the following steps:

1. Right-click the tree view, and select **Add businessEntity**.
2. In the **Business** dialog, enter a **Name** and **Description** for the business.
3. Click **OK**.
4. You can right-click the new `businessEntity` node to add child UDDI entities in the tree (for example, `businessService`, `Category`, and `Identifier` entities).

Adding a tModel
To add a `tModel`, perform the following steps:

1. Right-click the tree view, and select **Add tModel**.
2. In the **tModel** dialog, enter a **Name**, **Description**, and **Overview URL** for the `tModel`. For example, you can use the **Overview URL** to specify the location of a WSDL file.
3. Click **OK**.
4. You can right-click the new `tModel` node to add child UDDI entities in the tree (for example, `Category` and `Identifier` entities).

As before, you can click any node in the results tree to display properties about that node in the table. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node). At any stage, you can click the **Clear** button on the right to clear the entire contents of the tree. This does not delete the contents of the registry.

For more details on UDDI entities such as `businessEntity` and `tModel`, see the UDDI Definitions section. For details on how to publish Web Services automatically using a wizard, see *Publishing WSDL Files to a UDDI Registry*.

For more details on UDDI entities such as `businessEntity` and `tModel`, see the UDDI Definitions section.

Connecting to a UDDI Registry

Overview

This topic explains how to configure a connection to a UDDI registry in the **Registry Connection Details** dialog. It explains how to configure connections to UDDI v2 and UDDI v3 registries, also how to secure a connection over SSL.

Configuring a Registry Connection

Configure the following fields in the **Registry Connection Details** dialog:

Registry Name:
Enter the display name for the UDDI registry.

UDDI v2:
Select this option to use UDDI v2.

UDDI v3:
Select this option to use UDDI v3.

Inquiry URL:
Enter the URL on which to search the UDDI registry (for example, `http://HOSTNAME:PORT/uddi/inquiry`).

Publish URL:
Enter the URL on which to publish to the UDDI registry, if required (for example, `http://HOSTNAME:PORT/uddi/publishing`).

Security URL (UDDI v3):
For UDDI v3 only, enter the URL for the security service, if required (for example, `http://HOSTNAME:PORT/uddi/security.wsdl`).

Important

For UDDI v3, the **Inquiry URL**, **Publish URL**, and **Security URL** specify the URLs of the WSDL for the inquiry, publishing, and security Web services that the registry exposes. These fields can use the same URL if the WSDL for each service is at the same URL.

For example, a WSDL file at `http://HOSTNAME:PORT/uddi/uddi_v3_registry.wsdl` can contain three URLs (`http://HOSTNAME:PORT/uddi/inquiry`, `http://HOSTNAME:PORT/uddi/publishing`, and `http://HOSTNAME:PORT/uddi/security`). These are the service endpoint URLs that the Policy Studio uses to browse and publish to the registry. These URLs are not set in the connection dialog, but discovered using the WSDL. However, for UDDI v2, WSDL is *not* used to discover the service endpoints, so you must enter these URLs directly in the connection dialog.

Max Rows:
Enter the maximum number of entries returned by a search. Defaults to `20`.

Registry Authentication:
The registry authentication settings are as follows:

Type	This optional field applies to UDDI v2 only. The only supported authentication type is `UDDI_GET_AUTHTOKEN`.
Username	Enter the username required to authenticate to the registry, if required.
Password	Enter the password for this user, if required.

The username and password apply to UDDI v2 and v3. These are generally required for publishing, but depend on the configuration on the registry side.

HTTP Proxy:
The HTTP proxy settings apply to UDDI v2 and v3:

Proxy Host	If the UDDI registry location entered above requires a connection to be made through an HTTP proxy, enter the host name of the proxy.
Proxy Port	If a proxy is required, enter the port on which the proxy server is listening.
Username	If the proxy has been configured to only accept authenticated requests, the Policy Studio sends this username and password to the proxy using HTTP Basic authentication.
Password	Enter the password to use with the username specified in the field above.

HTTPS Proxy:
The HTTPS proxy settings apply to UDDI v2 and v3:

SSL Proxy Host	If the **Inquiry URL** or **Publish URL** uses the HTTPS protocol, the SSL proxy host entered is used instead of the HTTP proxy entered above. In this case, the HTTP proxy settings are not used.
Proxy Port	Enter the port that the SSL proxy is listening on.

Securing a Connection to a UDDI Registry

You may wish to communicate with the UDDI registry over SSL. All communication may not need to be over SSL (for example, you may wish publish over SSL, and perform inquiry calls without SSL). For UDDI v2 and v3, you can use a mix of `http` and `https` URLs for WSDL and service address locations.

You can specify some or all of the **Inquiry URL**, **Publish URL**, and **Security URL** settings as `https` URLs. For example, with UDDI v3, you could use a single URL like the following:

```
https://HOSTNAME:PORT/uddi/wsdl/uddi_v3_registry.wsdl
```

If any URLs (WSDL or service address location) use `https`, you must configure the Policy Studio so that it trusts the registry SSL certificate.

Configuring the Policy Studio to Trust a Registry Certificate
For an SSL connection, you must configure the registry server certificate as a trusted certificate. Assuming mutual authentication is not required, the simplest way to configure an SSL connection between the Policy Studio and UDDI registry is to add the registry certificate to the Policy Studio default truststore (the `cacerts` file). You can do this by performing the following steps in the Policy Studio:

1. Select the **Certificates and Keys** -> **Certificates** node in the Policy Studio tree.
2. Click **Create/Import**, and click **Import Certificate**.
3. Browse to the UDDI registry SSL certificate file, and click **Open**.

4. Click **Use Subject** on the right of the **Alias Name** field, and click **OK**. The registry SSL Certificate is now imported into the **Certificate Store**, and must be added to the Java keystore.
5. Click **Keystore** on the **Certificate** screen.
6. Click **Browse** next to the **Keystore** field.
7. Browse to the following file: `INSTALL_DIR/policystudio/jre/lib/security/cacerts`
8. Click **Open**, and enter the **Keystore password**. The default password is: `changeit`.
9. Click **Add to Keystore**.
10. Browse to the registry SSL certificate imported earlier, select it, and click **OK**.
11. Restart the Policy Studio. You should now be able to connect to the registry over SSL.

Configuring Mutual SSL Authentication

If mutual SSL authentication is required (if the Policy Studio must authenticate to the registry), the Policy Studio must have an SSL private key and certificate. In this case, you should create a keystore containing the Policy Studio key and certificate. You must configure the Policy Studio to load this file. For example, edit the `INSTALL_DIR/policystudio/policystudio.ini` file, and add the following arguments:

```
-Djavax.net.ssl.keyStore=/home/oracle/osr-client.jks
-Djavax.net.ssl.keyStorePassword=changeit
```

This example shows an `osr-client.jks` keystore file used with Oracle Service Registry (OSR), which is the UDDI registry provided by Oracle.

Note

You can also use the Policy Studio to create a new keystore (`.jks`) file. Click **New keystore** instead of browsing to the `cacerts` file as described earlier.

Publishing WSDL Files to a UDDI Registry

Overview

You can register Web Services in the **Web Services Repository** using Web Services Description Language (WSDL) files. The Policy Studio can retrieve a WSDL file from the file system, from a URL, or from a UDDI registry. When you have registered a WSDL file in the **Web Services Repository**, you can use the **Publish WSDL Wizard** to publish the WSDL file to a UDDI registry. You can also use the **Find WSDL Wizard** to search for the selected WSDL file in a UDDI registry. This topic explains how to perform both of these tasks.

For background information and an introduction to general UDDI concepts, see *Retrieving WSDL Files from a UDDI Registry*. For details on how to register WSDL files, see *Web Service Repository*.

Finding WSDL Files

You can search a UDDI registry to determine if a Web Service is already published in the registry. To search for a selected WSDL file in a specified UDDI registry, perform the following steps:

1. In the Policy Studio tree, expand the **Business Services** -> **Web Services Repository** node.
2. Right-click a WSDL node (for example, `http://HOSTNAME/TestService/StockQuote.svc?WSDL`, where `HOSTNAME` is the endpoint host of the Web service).
3. Select **Find in UDDI Registry** to launch the **Find WSDL Wizard**.
4. In the **Find WSDL** screen, select a UDDI **Registry** from the list. You can add or edit a registry connection using the buttons provided. For details on configuring a registry connection, see *Connecting to a UDDI Registry*.
5. You can select an optional language **Locale** from the list. The default is `No Locale`.
6. Click **Next**. The **WSDL Found in UDDI Registry** screen displays the result of the search in a tree. The **Node Counts** field shows the total numbers of each UDDI entity type returned from the search (`businessEntity`, `businessService`, `bindingTemplate`, and `tModel`).
7. You can right-click to edit a UDDI entity node in the tree, if necessary (for example, add a description, add a category or identifier node, or delete a duplicate node).
8. Click the **Refresh** button to run the search again.
9. Click **Finish**.

The **Find WSDL Wizard** provides is a quick and easy way of finding a selected WSDL file published in a UDDI registry. For more fine-grained ways of searching a UDDI registry (for example, for specific WSDL or UDDI entities), see *Retrieving WSDL Files from a UDDI Registry*.

Publishing WSDL Files

To publish a WSDL file registered in the **Web Services Repository** to a UDDI registry, perform the following steps:

1. Expand the **Business Services** -> **Web Services Repository** tree node.
2. Right-click a WSDL node (for example, `http://HOSTNAME/TestService/StockQuote.svc?WSDL`, where `HOSTNAME` is the host from which the Web service is registered).
3. Select **Publish WSDL to UDDI Registry** to launch the **Publish WSDL Wizard**.
4. Perform the steps in the wizard screens described in the next sections.

Step 1: Enter Virtualized Service Address and WSDL URL for Publishing in UDDI Registry

When you register a WSDL file in the Web Services Repository, the API Gateway exposes a *virtualized* version of the Web Service. The host and port for the Web Service are changed dynamically to point to the machine running the API Gateway. The client can then retrieve the WSDL for the virtualized Web Service from the API Gateway, without knowing its real location.

This screen enables you to optionally override the service address locations in the WSDL file with the virtualized addresses exposed by the API Gateway. You can also override the WSDL URL published to the UDDI registry. Complete the following fields:

Mapping of Service Addresses to Virtualized Service Addresses
You can enter multiple virtual service address mappings for each service address specified in the selected WSDL file. If you do not enter a mapping, the original address location in the WSDL file is published to the UDDI registry. If one or more mappings are provided, corresponding UDDI `bindingTemplates` are published in the UDDI registry, each with a different access point (virtual service address). This enables you to publish the access points of a service when it is exposed on different ports/schemes using the API Gateway.

When you launch the wizard, the mapping table is populated with a row for each `wsdl:service`, `wsdl:port`, `soap:address`, `soap12:address`, or `http:address` in the selected WSDL file. To modify an existing entry, select a row in the table, and click **Edit**. Alternatively, click **Add** to add an entry. In the **Virtualize Service Address** dialog, enter the virtualized service address. For example, if the API Gateway is running on a machine named `roadrunner`, the new URL on which the Web service is available to clients is: `http://roadrunner:8080/TestServices/StockQuote.svc`.

WSDL URL:
You can enter a WSDL URL to be published to the UDDI registry in the corresponding `tModel overviewURL` fields. If you do not enter a URL, the WSDL URL in the **Original WSDL URL** field is used. For example, an endpoint service is at `http://coyote.qa.acmecorp.com/TestService/StockQuote.svc`. Assume this service is virtualized in the API Gateway and exposed at `http://HOST:8080/TestService/StockQuote.svc`, where `HOST` is the machine on which the API Gateway is running. The `http://HOST:8080/TestService/StockQuote.svc` URL is entered as the virtual service address, and `http://HOST:8080/TestService/StockQuote.svc?WSDL` is entered as the WSDL URL to Publish.

Note

If incorrect URLs are published, you can edit these in the UDDI tree in later steps in this wizard, or when browsing the registry.

Click **Next** when finished.

Step 2: View WSDL to UDDI Mapping Result

You can use this screen to view the unpublished mapping of the WSDL file to a UDDI registry structure. You can also edit a specific mapping in the tree view. This screen includes the following fields:

Mapping of WSDL to a UDDI Registry Structure:
The unpublished mappings from the WSDL file to the UDDI registry are displayed in the table. For example, this includes the relevant `businessService`, `bindingTemplate`, `tModel`, `Identifier`, `Category` mappings. You can select a tree node to display its values in the table below.

You can optionally edit the values for a specific mapping in the table (for example, update a value, or add a key or description for the selected UDDI entity). You can also right-click a tree node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Retrieve service address from WSDL instead of bindingTemplate access point:
When selected, this ensures that the `bindingTemplate` access point does not contain the service port address, and is set to `WSDL` instead. This means that you must retrieve the WSDL to get the service access point. When selected, the `bindingTemplate` contains an additional `tModelInstanceInfo` that points to the `uddi:uddi.org.wsdl:address tModel`. This option is not selected by default.

Include WS-Policy as:

When selected, you can choose one of the following options to specify how WS-Policy statements in the WSDL file are included in the registry:

Remote Policy Expressions	Each WS-Policy URL in the WSDL that is associated with a mapped UDDI entity is accessed remotely. For example, a `businessService` is categorized with the `uddi:w3.org:ws-policy:v1.5:attachment:remotepolicyreference tModel` where the `keyValue` holds the remote WS-Policy URL. This is the default option.
Reusable Policy Expressions	Each WS-Policy URL in the WSDL that is associated with a mapped UDDI entity has a separate `tModel` published for it. Other UDDI entities (for example, `businessService`) can then refer to these `tModels`. These are reusable because UDDI entities published in the future can also use these `tModels`. You can do this in **Step 4: Select a duplicate publishing approach** by selecting the **Reuse duplicate tModels** option.

Click **Next** when finished.

Step 3: Select a Registry for Publishing

Use this screen to select a UDDI registry in which to publish the WSDL to UDDI mapping. Complete the following fields:

Select Registry
Select an existing UDDI registry to browse for WSDL files from the **Registry** drop-down list. To configure the location of a new UDDI registry, click **Add**. Similarly, to edit an existing UDDI registry location, click **Edit**. For details on how to configure a UDDI connection, see *Connecting to a UDDI Registry*.

Select Locale:
You can select an optional language locale from this list. The default is `No Locale`.

Click **Next** when finished.

Step 4: Select a Duplicate Publishing Approach

This screen is displayed only if mapped WSDL entities already exist in the UDDI registry. Otherwise, the wizard skips to step 5. This screen includes the following fields:

Select Duplicate Mappings
The **Mapped WSDL to publish** pane on the left displays the unpublished WSDL mappings from Step 2. The **Duplicates for WSDL mappings in UDDI registry** pane on the right displays the nodes already published in the registry. The **Node List** at the bottom right shows a breakdown of the duplicate nodes.

Edit Duplicate Mappings
You can eliminate duplicate mappings by right-clicking a tree node in the right or left pane, and selecting edit to update a specific mapping in the dialog. Select the **Refresh** button on the right to run the search again, and view the updated **Node List**. Alternatively, you can configure the options in the next field.

Select Publishing Approach for Duplicate Entries:
Select one of the following options:

Reuse duplicate tModels	Publishes the selected entries from the tree on the left, and reuses the selected duplicate entries in the tree on the right. This is the default option. Some or all duplicate

	`tModels` (for example, for `portType`, `binding`, and reusable WS-Policy expressions) that already exist in the registry can be reused. This means that a new `businessService` that points to existing `tModels` is published. Any entries selected on the left are published, and any referred to `tModels` on the left now point to selected duplicate `tModels` on the right. By default, this option selects all `businessServices` on the left, and all duplicate `tModels` on the right. If there is more than one duplicate `tModels`, only the first is selected.
Overwrite duplicates	Publishes the selected entries from the tree on the left, and overwrites the selected duplicate entries in the tree on the right. When a UDDI entity is overwritten, its UUID key stays the same, but all the data associated with it is overwritten. This is not just a transfer of additions or differences. You can also overwrite some duplicates and create some new entries. By default, this option selects all `businessServices` and `tModels` on the left and all duplicate `businessServices` and `tModels` on the right. If there is more than one duplicate, only the first is selected. The default overwrites all selected duplicates and does not create any new UDDI entries, unless there is a new referred to `tModel` (for example, for a reusable WS-Policy expression).
Ignore duplicates	Publishes the selected entries from the tree on the left, and ignores all duplicates. You can proceed to publish the mapped WSDL to UDDI data. New UDDI entries are created for each item that is selected in the tree on the left.

Click **Next** when finished.

Note

If you select duplicate `businessServices` in the tree, and select **Overwrite duplicates**, the wizard skips to Step 6 when you click **Next**.

Step 5: Create or Search for Business

Use this screen to specify a `businessEntity` for the Web Service. You can create a new `businessEntity` or search for an existing one in the UDDI registry. Complete the following fields:

Creating a New businessEntity
This is the default option. Enter a **Name** and **Description** for the `businessEntity`, and click **Publish**.

Searching for an Existing businessEntity
To search for an existing `businessEntity` name, perform the following steps:

1. Select the **Search for an existing businessEntity in the UDDI registry** option.
2. In the **Search** tab, ensure the **Name Search** option is selected.
3. Enter a **Name** option (for example, `Acme Corporation`).

Alternatively, you can select the **Advanced Search** option to search by different criteria such as **Keys**, **Categories**, or

tModels. For more details, see *Retrieving WSDL Files from a UDDI Registry*.

Advanced Options
You can also select a range of search options on the **Advanced** tab (for example, **Exact match**, **Case sensitive**, or **Service subset**). For more details, see *Retrieving WSDL Files from a UDDI Registry*.

The **Node Counts** field shows the total numbers of each UDDI entity type returned from the search (`businessEntity`, `businessService`, `bindingTemplate`, and `tModel`).

Click **Next** when finished.

Step 6: Publish WSDL

Use this screen to publish the WSDL to the UDDI registry.

Selected businessEntity for Publishing:
This field displays the name and `tModel` key of the `businessEntity` to be published. Click the **Publish WSDL** button on the right.

Published WSDL:
This field displays the tree of the UDDI mapping for the WSDL file. You can right-click to edit or delete any nodes in the tree if necessary, and click **Refresh** to run the search again. Click **Publish WSDL** to publish your updates.

Click **Finish**.

LDAP User Search

Configure Directory Search

The **User Search** dialog is used to search a given LDAP directory for a unique user according to the criteria configured in the fields on this dialog.

Base Criteria:
The value entered here tells the API Gateway where it should begin searching the LDAP directory. For example, it may be appropriate to search for a given user under the `C=IE` tree in the LDAP hierarchy.

Query Search Filter:
The value entered here is what the API Gateway uses to determine whether it has obtained a successful match. In this case, because you are searching for a specific user, you can use the username of an authenticated user (the value of the `authentication.subject.id` message attribute to lookup in the LDAP directory. You must also specify the object class that defines users for the particular type of LDAP directory that you are searching against. For example, object classes representing users amongst common LDAP directories are `inetOrgPerson`, `givenName`, and `User`.

For example, to search for an authenticated user against Microsoft's Active Directory, you might specify the following as the **Query Search Filter**:

```
(objectclass=User)(cn=${authentication.subject.id})
```

This example uses a selector to obtain the ID of the authenticated subject at runtime. For more details on selectors, see *Selecting Configuration Values at Runtime*.

Search Scope:
These settings specify the depth of the LDAP tree that you wish to search. The settings selected here depends largely on the structure of your LDAP directory.

Configuring URL Groups

Overview

The API Gateway can make connections on a round-robin basis to the URLs listed in a URL group, thus enabling a high degree of failover to external servers (for example, Entrust GetAccess, OCSP, SAML PDP, or XKMS).

The API Gateway attempts to connect to the listed servers according to the priorities assigned to them. For example, assume there are two High priority URLs, one Medium URL, and one Low URL configured. Assuming the API Gateway can successfully connect to the two High priority URLs, it alternates requests between these two URLs only in a round-robin fashion. The other group URLs are not used. However, if both of the High priority URLs become unavailable, the API Gateway then tries to use the Medium priority URL, and only if this fails is the Low priority URL used.

In general, the API Gateway attempts to round-robin requests over URLs of the same priority, but uses higher priority URLs before lower priority ones. When a new URL is added to the group, it is automatically given the highest priority. You can then change priorities by selecting the URL, and clicking the **Up** and **Down** buttons.

You can add and edit URLs by selecting the URL from the table, and clicking on the **Add** and **Edit** buttons.

Configuration

Configure the following fields in the **URL Configuration** dialog:

- **URL:**
  Enter the full URL of the external server.
- **Timeout:**
  Specify the timeout in seconds for connections to the specified server.
- **Retry After:**
  Whenever the server becomes unavailable for whatever reason (for example, maintenance), no attempt is made to connect to that server until the time specified here has elapsed. In other words, when a connection failure is detected, the next connection to that URL is after this amount of time.
- **SSL Certificate:**
  If the specified server requires clients to authenticate to it over two-way SSL, you must select an **SSL Certificate** from the Certificate Store for authentication.
- **Host/IP:**
  If the specified server sits behind a proxy server, you must enter the host name or IP address of the proxy server.
- **Port:**
  Enter the port on which the proxy is listening.

What To Sign

Overview

The **What To Sign** section enables the administrator to define the exact content that must be signed for a SOAP message to pass the corresponding filter. The purpose of this configuration section is to ensure that the client has signed something meaningful (part of the SOAP message) instead of some arbitrary data that would pass a blind signature validation.

This prevents clients from simply pasting technically correct, but unrelated signatures into messages in the hope that they pass any blind signature verification. For example, the user may be able to generate a valid XML Signature over any arbitrary XML document. Then by including the signature and XML portion into a malicious SOAP message, the signature passes a blind signature validation, and the harmful XML is allowed to reach the Web Service.

The **What To Sign** section ensures that clients must sign a part of the SOAP message, and therefore prevents them from pasting arbitrary XML Signatures into the message. This section enables you to use any combination of **Node Locations**, **XPath Expressions**, **XPath Predicates**, and/or **Message Attribute** to specify message content that must be signed. This topic describes how to configure each of the corresponding tabs displayed in this section.

ID Configuration

With WSU IDs, an ID attribute is inserted into the root element of the nodeset that is to be signed. The XML Signature then references this ID to indicate to verifiers of the signature the nodes that were signed. The use of WSU IDs is the default option because they are WS-I compliant.

Alternatively, a generic ID attribute (that is not bound to the WSU namespace) can be used to dereference the data. The ID attribute is inserted into the top-level element of the nodeset to be signed. The generated XML Signature can then reference this ID to indicate what nodes were signed.

You can also use `AssertionID` attributes when signing SAML assertions. The following options provide more details and examples of the different styles of IDs that are available.

Use WSU IDs:
Select this option to reference the signed data using a `wsu:Id` attribute. In this case, a `wsu:Id` attribute is inserted into the root node of the nodeset that is signed. This id is then referenced in the generated XML Signature as an indication of what nodes were signed. The following example shows the correlation:

```
<s:Envelope xmlns:s="...">
  <s:Header>
  <wsse:Security xmlns:wsse="...">
    <dsig:Signature xmlns:dsig="..." Id="Id-00000112e2c98df8-0000000000000004">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-00000112e2c98df8-0000000000000003">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>xChPoiWJJrrPZkbXN8FPB8S4U7w=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>KG4N .... /9dw==</dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-00000112e2c98df8-0000000000000005">
        <dsig:X509Data>
```

```
          <dsig:X509Certificate>
            MIID ... ZiBQ==
          </dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
  </wsse:Security>
  </s:Header>
   <s:Body xmlns:wsu="..." wsu:Id="Id-00000112e2c98df8-0000000000000003">
  <vs:getProductInfo xmlns:vs="http://ww.oracle.com">
   <vs:Name>API Gateway</vs:Name>
   <vs:Version>11.1.2.1.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

In the above example, a `wsu:Id` attribute has been inserted into the `<s:Body>` element. This `wsu:Id` attribute is then referenced by the `URI` attribute of the `<dsig:Reference>` element in the actual Signature.

When the Signature is being verified, the value of the `URI` attribute can be used to locate the nodes that have been signed.

Use IDs:
Select this option to use generic IDs (that are not bound to the WSU namespace) to dereference the signed data. Under this schema, the `URI` attribute of the `<Reference>` points at an ID attribute, which is inserted into the top-level node of the nodeset that is signed. Take a look at the following example, noting how the ID specified in the Signature matches the ID attribute that has been inserted into the `<Body>` element, indicating that the Signature applies to the entire contents of the SOAP Body.

```
lt;soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
                Id="Id-0000011a101b167c-0000000000000013">
      <dsig:SignedInfo>
        <dsig:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <dsig:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <dsig:Reference URI="#Id-0000011a101b167c-0000000000000012">
          <dsig:Transforms>
            <dsig:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </dsig:Transforms>
          <dsig:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <dsig:DigestValue>JCy0JoyhVZYzmrLrl92nxfr1+zQ=</dsig:DigestValue>
        </dsig:Reference>
      </dsig:SignedInfo>
      <dsig:SignatureValue>......<dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-0000011a101b167c-0000000000000014">
        <dsig:X509Data>
          <dsig:X509Certificate>......</dsig:X509Certificate>
        </dsig:X509Data>
      </dsig:KeyInfo>
    </dsig:Signature>
  </soap:Header>
 <soap:Body Id="Id-0000011a101b167c-0000000000000012">
  <product version="11.1.2.1.0">
    <name>API Gateway</name>
    <company>Oracle</company>
    <description>SOA Security and Management</description>
  </product>
 </soap:Body>
</soap:Envelope>
```

Use SAML IDs for SAML Elements:
This ID option is specifically intended for use where a SAML assertion is to be signed. When this option is selected, an `AssertionID` attribute is inserted into a SAML 1.1 assertion, or a more generic ID attribute is used for a SAML 2.0 assertion.

Node Locations

Node Locations are perhaps the simplest way to configure the message content that must be signed. The table on this screen is pre-populated with a number of common SOAP security headers, including the SOAP Body, WS-Security block, SAML assertion, WS-Security UsernameToken and Timestamp, and the WS-Addressing headers. For each of these headers, there are several namespace options available. For example, you can sign both a SOAP 1.1 and/or a SOAP 1.2 block by distinguishing between their namespaces.

On the **Node Locations** tab, you can select one or more nodesets to sign from the default list. You can also add more default nodesets by clicking the **Add** button. Enter the **Element Name**, **Namespace**, and **Index** of the nodeset in the fields provided. The **Index** field is used to distinguish between two elements of the same name that occur in the same message.

XPath Configuration

You can use an XPath expression to identify the nodeset (the series of elements) that must be signed. To specify that nodeset, select an existing XPath expression from the table, which contains several XPath expressions that can be used to locate nodesets representing common SOAP security headers, including SAML assertions. Alternatively, you can add a new XPath expression using the **Add** button. XPath expressions can also be edited and removed with the **Edit** and **Remove** buttons.

An example of a SOAP message is provided below. The following XPath expression indicates that all the contents of the SOAP body, including the `Body` element itself, should be signed:

```
/soap:Envelope/soap:Body/descendant-or-self::node()
```

You must also supply the namespace mapping for the `soap` prefix, for example:

Prefix	***URI***
soap	http://schemas.xmlsoap.org/soap/envelope/

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <product xmlns="http://www.oracle.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

XPath Predicates

Select this option if you wish to use an XPath transform to reference the signed content. You must select an **XPath Predicate** from the table to do this. The table is pre-populated with several XPath predicates that can be used to identify common security headers that occur in SOAP messages, including SAML assertions.

To illustrate the use of XPath predicates, the following example shows how the SOAP message is signed when the default *Sign SOAP Body* predicate is selected:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <vs:getProductInfo xmlns:vs="http://www.oracle.com">
   <vs:Name>API Gateway</vs:Name>
   <vs:Version>11.1.2.1.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

The default XPath expression (Sign SOAP Body) identifies the contents of the SOAP `Body` element, including the `Body` element itself. The following is the XML Signature produced when this XPath predicate is used:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <dsig:Signature id="Sample" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
   <dsig:SignedInfo>
     ...
    <dsig:Reference URI="">
     <dsig:Transforms>
      <dsig:Transform
          Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
       <dsig:XPath xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
         ancestor-or-self::soap:Body
       </dsig:XPath>
      </dsig:Transform>
      <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n"/>
     </dsig:Transforms>
       ...
    </dsig:Reference>
   </dsig:SignedInfo>
    ...
  </dsig:Signature>
 </s:Header>
 <s:Body>
  <vs:getProductInfo xmlns:vs="http://ww.oracle.com">
   <vs:Name>API Gateway</vs:Name>
   <vs:Version>11.1.2.1.0</vs:Version>
  </vs:getProductInfo>
 </s:Body>
</s:Envelope>
```

This XML Signature includes an extra `Transform` element, which has a child `XPath` element. This element specifies the XPath predicate that validating applications must use to identify the signed content.

Message Attribute

Finally, you can use the contents of a message attribute to determine what must be signed in the message. For example, you can configure a *Locate XML Nodes* filter to extract certain content from the message and store it in a particular message attribute. You can then specify this message attribute on the **Message Attribute** tab.

To do this, select the **Extract nodes from message attribute** checkbox, and enter the name of the attribute that contains the nodes in the field provided.

Configuring XPath Expressions

Overview

The API Gateway uses XPath expressions in a number of ways, for example, to locate an XML Signature in a SOAP message, to determine what elements of an XML message to validate against an XML Schema, to check the content of a particular element within an XML message, amongst many more uses.

There are two ways to configure XPath expressions on this screen:

- Manual Configuration
- XPath Wizard

Manual Configuration

If you are already familiar with XPath and wish to configure the expression manually, complete the following fields, using the examples below if necessary:

1. Enter or select a name for the XPath expression in the **Name** drop-down list.
2. Enter the XPath expression to use in the **XPath Expression** field.
3. In order to resolve any prefixes within the XPath expression, the namespace mappings (**Prefix**, **URI**) should be entered in the table.

Consider the following example SOAP message: >

```
 <?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
      ...............
      ...............
      ...............
      ...............
  </dsig:Signature>
</soap:Header>
  <soap:Body>
    <prod:product xmlns:prod="http://www.oracle.com">
      <prod:name>SOA Product*</prod:name>
      <prod:company>Company</prod:company>
      <prod:description>WebServices Security</prod:description>
    </prod:product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the <*name*> element contains the value *API Gateway*:
XPath Expression: `//prod:name[text()='API Gateway']`

In this case, it is necessary to define a mapping for the *prod* namespace as follows:

Prefix	***URI***
`prod`	`http://www.oracle.com`

In another example, the element to be examined belongs to a default namespace. Consider the following SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
  <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sample">
      ...............
      ...............
      ...............
      ...............
  </dsig:Signature>
</soap:Header>
  <soap:Body>
    <product xmlns="http://www.company.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>WebServices Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the *<company>* element contains the value *Company*:
XPath Expression: `//ns:company[text()='Company']`

The *<company>* element actually belongs to the default (`xmlns`) namespace (`http://www.company.com`. This means that it is necessary to make up an arbitrary prefix, `ns`, for use in the XPath expression and assign it to `http://www.company.com`. This is necessary to distinguish between potentially several default namespaces, which may exist throughout the XML message. The following mapping illustrates this:

Prefix	***URI***
`ns`	`http://www.oracle.com`

Returning a NodeSet:
Both of the examples above dealt with cases where the XPath expression evaluated to a Boolean value. For example, the expression in the above example asks does the `<company>` element in the `http://www.oracle.com` namespace contain a text node with the value `oracle`?.

It is sometimes necessary to use the XPath expression to return a subset of the XML message. For example, when using an XPath expression to determine what nodes should be signed in a signed XML message, or when retrieving the nodeset to validate against an XML Schema.

The API Gateway ships with such an XPath expression: one that returns `All Elements inside SOAP Body` To view this expression, select it from the **Name** field. It appears as follows:
XPath Expression: `/soap:Envelope/soap:Body//*`

This XPath expression simply returns all child elements of the SOAP `<Body>` element. To construct and test more complicateD expressions, administrators are advised to use the **XPath Wizard**.

XPath Wizard

The **XPath Wizard** assists administrators in creating correct and accurate XPath expressions. The wizard allows administrators to load an XML message and then run an XPath expression on it to determine what nodes are returned. To launch the **XPath Wizard**, click the **XPath Wizard Button** on the **XPath Expression** dialog.

To use the XPath Wizard, simply enter (or browse to) the location of an XML file in the **File** field. The contents of the XML file will appear in the main window of the wizard. Enter an XPath expression in the **XPath** field and click the **Evaluate** button to run the XPath against the contents of the file. If the XPath expression returns any elements (or returns true), those elements will be highlighted in the main window.

If you are not sure how to write the XPath expression, you can select an element in the main window. An XPath expression to isolate this element is automatically generated and displayed in the **Selected** field. If you wish to use this expression, select the **Use this path** button, and click **OK**.

Message Attribute Reference

attribute.lookup.list

Name	`attribute.lookup.list`
Description	User attributes can be retrieved from a variety of sources, including LDAP directories, databases, Oracle Entity Store, SAML attribute assertions, and so on. All retrieved attributes are stored in the `attribute.lookup.list` attribute, where they can be looked up at a later stage in the policy.
Type	`java.util.HashMap`
Generated By	**Extract Certificate Attributes** **Retrieve Attributes from Database** **Retrieve Attributes from Directory Server** **Retrieve Attributes from SAML Attribute Assertion** **Retrieve Attributes from SAML PDP** **Retrieve Attributes from User Store** **SiteMinder Authorization**
Consumed By	
Required By	**Attribute Authorization** **Insert SAML Attribute Assertion**

attribute.subject.format

Name	`attribute.subject.format`
Description	The format of the subject ID that is used to lookup attributes (for example, X.509 DName or username).
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes** **Retrieve Attributes from Directory Server** **Retrieve Attributes from SAML Attribute Assertion** **SiteMinder Authorization**
Consumed By	
Required By	**Insert SAML Attribute Assertion**

attribute.subject.id

Name	`attribute.subject.id`
Description	The ID of the subject that is used to look up user attributes. This can either be an X.509 Distinguished Name (DName) or a username.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes** **Retrieve Attributes from Directory Server** **Retrieve Attributes from SAML Attribute Assertion** **SiteMinder Authorization**

Consumed By	
Required By	**Insert SAML Attribute Assertion**

authentication.cert

Name	`authentication.cert`
Description	The certificate that was used to authenticate the client.
Type	`java.security.cert.X509Certificate`
Generated By	**HTTP Header Authentication**
Consumed By	
Required By	

authentication.issuer.format

Name	`authentication.issuer.format`
Description	The format of the authentication.issuer.id attribute.
Type	`java.lang.String`
Generated By	**HTTP Header Authentication** **SSL Authentication**
Consumed By	
Required By	

authentication.issuer.id

Name	`authentication.issuer.id`
Description	Contains the ID of the issuer of the authenticated client's certificate. This is usually either the X.509 DName or the username of the issuer of the subject's certificate.
Type	`java.lang.String`
Generated By	**HTTP Header Authentication** **SSL Authentication** **SAML Authentication XML-Signature Verification**
Consumed By	
Required By	

authentication.issuer.orig.format

Name	`authentication.issuer.orig.format`
Description	Format of the authentication.issuer.orig.id attribute. This is the format of the issuer's ID before any credential mapping

	was done to the identifier, e.g. DName to username.
Type	`java.lang.String`
Generated By	**HTTP Header Authentication**
Consumed By	
Required By	

authentication.issuer.orig.id

Name	`authentication.issuer.orig.id`
Description	The ID of the issuer of the subject's credential before any credential mapping took place. An example of credential mapping would involve mapping an issuer's username to a DName.
Type	`java.lang.String`
Generated By	**HTTP Header Authentication**
Consumed By	
Required By	

authentication.method

Name	`authentication.method`
Description	The method used by the client to authenticate to API Gateway.
Type	`java.lang.String`
Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication** **SAML Authentication** **SSL Authentication** **SiteMinder Certificate Authentication** **SiteMinder Session Validation** **XML-Signature Authentication**
Consumed By	
Required By	**SAML Authentication** **Retrieve Attributes from SAML PDP** **SAML PDP Authorization**

authentication.subject.format

Name	`authentication.subject.format`
Description	The format of the subject's ID, e.g. X.509 DName or username.
Type	`java.lang.String`

Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication** **Retrieve Attributes from SAML Attribute Assertion** **Retrieve Attributes from SAML PDP** **Retrieve Attributes from Database** **Retrieve Attributes from Directory Server** **Retrieve Attributes from User Store** **SAML Authentication** **SSL Authentication** **SiteMinder Certificate Authentication** **SiteMinder Session Validation** **XML-Signature Authentication**
Consumed By	
Required By	**Retrieve Attributes from Database** **Retrieve Attributes from Directory Server** **Retrieve Attributes from SAML Attribute Assertion** **Retrieve Attributes from SAML PDP** **Retrieve Attributes from User Store** **SAML PDP Authorization** **SAML Authorization** **Tivoli Authorization**

authentication.subject.id

Name	`authentication.subject.id`
Description	Contains the ID of the authenticated subject (for example, the username supplied by the client).
Type	`java.lang.String`
Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication** **Retrieve Attributes from SAML Attribute Assertion** **Retrieve Attributes from SAML PDP** **Retrieve Attributes from Database** **Retrieve Attributes from Directory Server** **Retrieve Attributes from User Store** **SAML Authentication** **SSL Authentication** **SiteMinder Certificate Authentication** **SiteMinder Session Validation** **XML-Signature Authentication**
Consumed By	
Required By	**Retrieve Attributes from Database** **Retrieve Attributes from Directory Server** **Retrieve Attributes from SAML Attribute Assertion** **Retrieve Attributes from SAML PDP** **Retrieve Attributes from User Store**

authentication.subject.orig.format

Name	`authentication.subject.orig.format`
Description	The ID of the subject before any credential mapping was performed. For example, the subject's ID may be mapped from an X.509 DName to the username stored in an external database. In this case, the subject's original ID was a DName.
Type	`java.lang.String`
Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication**
Consumed By	
Required By	

authentication.subject.orig.id

Name	`authentication.subject.orig.id`
Description	Contains the ID of the authenticated subject before credential mapping is performed (for example, from username to DName).
Type	`java.lang.String`
Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication**
Consumed By	
Required By	

authentication.subject.password

Name	`authentication.subject.password`
Description	If a user authenticates to the API Gateway using a username and password combination (either with HTTP digest/basic authentication or with a WS-Security Username token), the user's password is stored in this attribute.
Type	`java.lang.String`
Generated By	**HTTP Basic Authentication** **HTTP Digest Authentication**
Consumed By	
Required By	

authentication.ws.wsblockinfo

Name	`authentication.ws.wsblockinfo`

Description	Contains a WS-Security `<Header>` block that has been extracted from a message.
Type	`java.lang.String`
Generated By	**Extract WSS Header**
Consumed By	
Required By	

cert.basic.constraints

Name	`cert.basic.constraints`
Description	If the subject is a Certificate Authority (CA), and the `BasicConstraints` extension exists, this attribute gives the maximum number of CA certificates that may follow this certificate in a certification path. A value of zero indicates that only an end-entity certificate may follow in the path. This contains the value of `pathLenConstraint` if the `BasicConstraints` extension is present in the certificate and the subject of the certificate is a CA, otherwise its value is -1. If the subject of the certificate is a CA and `pathLenConstraint` does not appear, there is no limit to the allowed length of the certification path.
Type	`java.lang.Integer`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.extended.key.usage

Name	`cert.extended.key.usage`
Description	A string representing the OBJECT IDENTIFIERs of the `ExtKeyUsageSyntax` field of the extended key usage extension (`OID = 2.5.29.37`). It indicates a purpose for which the certified public key may be used, in addition to, or instead of, the basic purposes indicated in the key usage extension field.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.hash.md5

Name	`cert.hash.md5`
Description	An MD5 hash of the certificate.

Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.hash.sha1

Name	`cert.hash.sha1`
Description	An SHA1 hash of the certificate.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.alternative.name

Name	`cert.issuer.alternative.name`
Description	An alternative name for the certificate issuer from the `IssuerAltName` extension (`OID = 2.5.29.18`)
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id

Name	`cert.issuer.id`
Description	The Distinguished Name (DName) of the issuer of the certificate.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.c

Name	`cert.issuer.id.c`
Description	The c attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`

Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.cn

Name	`cert.issuer.id.cn`
Description	The `cn` attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.emailaddress

Name	`cert.issuer.id.emailaddress`
Description	The `email` or `emailaddress` attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.l

Name	`cert.issuer.id.l`
Description	The `l` attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.o

Name	`cert.issuer.id.o`
Description	The `o` attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	

Required By	

cert.issuer.id.ou

Name	`cert.issuer.id.ou`
Description	The ou attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.issuer.id.st

Name	`cert.issuer.id.st`
Description	The `st` attribute of the issuer of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.cRLSign

Name	`cert.key.usage.cRLSign`
Description	Set to `true` or `false` if the key can be used for `crlSign` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.dataEncipherment

Name	`cert.key.usage.dataEncipherment`
Description	Set to `true` or `false` if the key can be used for `dataEncipherment` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.decipherOnly

Name	`cert.key.usage.decipherOnly`
Description	Set to `true` or `false` if the key can be used for `decipherOnly` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.digitalSignature

Name	`cert.key.usage.digitalSignature`
Description	Set to `true` or `false` if the key can be used for digital signature.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.encipherOnly

Name	`cert.key.usage.encipherOnly`
Description	Set to `true` or `false` if the key can be used for `encipherOnly` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.keyAgreement

Name	`cert.key.usage.keyAgreement`
Description	Set to `true` or `false` if the key can be used for `keyAgreement` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.keyCertSign

Name	`cert.key.usage.keyCertSign`
Description	Set to `true` or `false` if the key can be used for `keyCertSign` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.keyEncipherment

Name	`cert.key.usage.keyEncipherment`
Description	Set to `true` or `false` if the key can be used for `keyEncipherment` .
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.key.usage.nonRepudiation

Name	`cert.key.usage.nonRepudiation`
Description	Set to `true` or `false` if the key can be used for non-repudiation.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.not.after

Name	`cert.not.after`
Description	Not after date for the validity of the certificate.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.not.before

Name	`cert.not.before`
Description	Not before date for the validity of the certificate.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.serial.number

Name	`cert.serial.number`
Description	Certificate serial number.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.signature.algorithm

Name	`cert.signature.algorithm`
Description	The signature algorithm for the certificate signature.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.alternative.name

Name	`cert.subject.alternative.name`
Description	An alternative name for the subject from the `SubjectAltName` extension (`OID = 2.5.29.17`).
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id

Name	`cert.subject.id`
Description	The Distinguished Name (DName) of the subject of the cer-

	tificate.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.c

Name	`cert.subject.id.c`
Description	The `c` attribute of the subject of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.cn

Name	`cert.subject.id.cn`
Description	The `cn` attribute of the subject of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.emailaddress

Name	`cert.subject.id.emailaddress`
Description	The `emailaddress` attribute of the subject of the certificate, if it exists.
Type	`java.lang.String`
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.l

Name	`cert.subject.id.l`
Description	The `l` attribute of the subject of the certificate, if it exists.
Type	`java.lang.String`

Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.o

Name	cert.subject.id.o
Description	The o attribute of the subject of the certificate, if it exists.
Type	java.lang.String
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.ou

Name	cert.subject.id.ou
Description	The ou attribute of the subject of the certificate, if it exists.
Type	java.lang.String
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.subject.id.st

Name	cert.subject.id.st
Description	The st attribute of the subject of the certificate, if it exists.
Type	java.lang.String
Generated By	**Extract Certificate Attributes**
Consumed By	
Required By	

cert.version

Name	cert.version
Description	The version of the certificate.
Type	java.lang.String
Generated By	**Extract Certificate Attributes**
Consumed By	

Required By	

certificate

Name	`certificate`
Description	Used to store a certificate in addition to the certificate stored in the authentication.cert attribute. For example, the **Find Certificate** filter can extract a certificate from an LDAP directory, HTTP header, or the User Store. By default, it sets this certificate to the certificate attribute.
Type	`java.lang.String`
Generated By	**Find Certificate** **HTTP Header Authentication** **XML-Signature Verification** **SAML Authentication XML-Signature Verification** **SAML PDP XML-Signature Response Verification** **SAML PDP Response XML-Signature Verification** **SSL Authentication** **XML-Signature Authentication**
Consumed By	
Required By	**Create Thumbprint from Certificate** **Extract Certificate Attributes** **SiteMinder Certificate Authentication**

certificate.thumbprint

Name	`certificate.thumbprint`
Description	Contains a human-readable thumbprint (or fingerprint), which is generated from the X.509 certificate stored in the `certificate` message attribute.
Type	`java.lang.String`
Generated By	**Create Thumbprint from Certificate**
Consumed By	
Required By	

certificates

Name	`certificates`
Description	Contains an array of X.509 certificates for use in the **Certificate Chain Check** and **Certificate Validation** filters.
Type	`java.util.ArrayList`
Generated By	**Find Certificate** **HTTP Header Authentication** **XML-Signature Verification**

	SAML Authentication XML-Signature Verification **SAML PDP XML-Signature Response Verification** **SAML PDP Response XML-Signature Verification** **SSL Authentication** **XML-Signature Authentication**
Consumed By	
Required By	**Certificate Chain** **Certificate Revocation List (Static)** **Certificate Revocation List (LDAP)** **OCSP (Online Certificate Status Protocol)** **SiteMinder Certificate Authentication** **XKMS (XML Key Management and Security)**

circuit.abort.exception

Name	`circuit.abort.exception`
Description	Whenever a filter throws an exception, the exception is stored in the `circuit.abort.exception` message attribute. The exception can then be used, for example, to return customized SOAP faults that describe the verbose details of the error. ⚠ **Important** A filter only throws an exception if it cannot carry out its core task. For example, an authentication filter throws an exception if it cannot connect to the authentication repository, a Schema validation filter aborts if the request is not XML, and an attribute retrieval filter aborts if it cannot connect to the user profile store (for example, database, LDAP, and so on).
Type	`java.lang.String`
Generated By	`circuit.abort.exception` can be thrown by all filters.
Consumed By	
Required By	`circuit.abort.exception` can be used as a selector in appropriate filters. For example, you can specify a `${circuit.abort.exception}` selector in the **Set Message** filter, which can then be returned to the client using a **Reflect** filter.

content.body

Name	`content.body`
Description	Contains the parsed body of the incoming HTTP request.
Type	`com.vordel.mime.Body`

Generated By	**Load File**
Consumed By	
Required By	**Content Type Filtering** **Connection** **Content Validation** **Entrust GetAccess Authorization** **Insert SAML Attribute Assertion** **SAML Authentication** **Insert SAML Authorization Assertion** **XML-Signature Verification** **Throttling** **McAfee Anti-Virus** **Operation Name** **Reflect** **Reflect Message and Attributes** **Retrieve Attribute from HTTP Header** **Retrieve Attribute from Message** **Retrieve Attributes from SAML Attribute Assertion** **SAML Authorization** **Save to File** **Schema Validation** **Sign Message** **SiteMinder Session Validation** **SSL Authentication** **XML Complexity** **XML-Decryption** **XML-Encryption** **XML-Signature Authentication** **XSLT Transformation** **Web Service Filter**

decryption.properties

Name	`decryption.properties`
Description	Indicates the XML-Encrypted block(s) to decrypt. The actual decryption is performed by the **XML-Decryption** filter.
Type	`java.util.Map`
Generated By	**XML-Decryption Settings**
Consumed By	
Required By	**XML-Decryption**

encryption.properties

Name	`encryption.properties`
Description	Allows the user to encrypt (part of) the message for a number of recipients so that only those recipients can decrypt the encrypted data. The encryption is performed by the **XML-Encryption** filter.

Type	`java.util.Map`
Generated By	**XML-Encryption Settings**
Consumed By	
Required By	**XML-Encryption**

http.destination.host

Name	`http.destination.host`
Description	The host on which the destination Web Service is running.
Type	`java.lang.String`
Generated By	**Dynamic Router** **Static Router** **Web Service Filter**
Consumed By	
Required By	**Connection**

http.destination.port

Name	`http.destination.port`
Description	The port on which the destination Web Service is listening.
Type	`java.lang.String`
Generated By	**Dynamic Router** **Static Router** **Web Service Filter**
Consumed By	
Required By	**Connection**

http.destination.protocol

Name	`http.destination.protocol`
Description	Indicates the protocol to use when routing messages to the destination Web Service. Typically, the protocol is either HTTP or HTTPS.
Type	`java.lang.String`
Generated By	**Dynamic Router** **Static Router** **Web Service Filter**
Consumed By	
Required By	**Connection**

http.headers

Name	`http.headers`
Description	Contains a list of all HTTP headers from the incoming request.
Type	`com.vordel.mime.HeaderSet`
Generated By	**Connection** **Web Service Filter**
Consumed By	
Required By	**Add HTTP Header** **Connection** **HTTP Basic Authentication** **HTTP Digest Authentication** **HTTP Header Authentication** **Reflect** **Reflect Message and Attributes** **Remove HTTP Header** **Return WSDL** **SOAPAction** **Validate HTTP Headers** **Web Service Filter**

http.request.clientaddr

Name	`http.request.clientaddr`
Description	Contains the IP address of the client machine from which the HTTP request was sent to API Gateway.
Type	`java.net.InetSocketAddress`
Generated By	
Consumed By	
Required By	**IP Address**

http.request.connection.error

Name	`http.request.connection.error`
Description	If an error occurs when API Gateway is routing on to the target Web Service, the error status will be recorded in this attribute.
Type	`java.lang.String`
Generated By	**Connection** **Web Service Filter**
Consumed By	
Required By	

http.request.clientcert

Name	`http.request.clientcert`
Description	Contains the certificate used by the client in the HTTP request.
Type	`java.security.cert.X509Certificate`
Generated By	
Consumed By	
Required By	

http.request.path

Name	`http.request.path`
Description	Contains the path on which the HTTP request from the client is received by the API Gateway (for example, `/test`).
Type	`java.lang.String`
Generated By	
Consumed By	
Required By	

http.request.uri

Name	`http.request.uri`
Description	Contains the URI on which the HTTP request is received by the API Gateway (for example, `/test?location=dublin`).
Type	`java.net.URI`
Generated By	**Web Service Filter**
Consumed By	
Required By	**Connection** **Dynamic Router** **Operation Name** **Rewrite URL** **Relative Path** **Return WSDL**

http.request.verb

Name	`http.request.verb`
Description	Contains the HTTP verb used in the client HTTP request to the API Gateway.
Type	`java.lang.String`
Generated By	

Consumed By	
Required By	**Connection** **HTTP Basic Authentication** **HTTP Digest Authentication** **Web Service Filter**

http.request.version

Name	`http.request.version`
Description	Contains the HTTP version used in the request from the client to the API Gateway.
Type	`java.lang.String`
Generated By	
Consumed By	
Required By	

http.response.info

Name	`http.response.info`
Description	Contains the `reason-phrase` from the HTTP status-line (for example, `Not found` from the `404 Not found` status-line).
Type	`java.lang.String`
Generated By	**Connection** **Web Service Filter**
Consumed By	
Required By	

http.response.status

Name	`http.response.status`
Description	Stores the HTTP status-code of the response from the Web Service, (for example, `404` from the `404 Not found` status-line).
Type	`java.lang.Integer`
Generated By	**Connection** **Web Service Filter**
Consumed By	
Required By	

http.response.version

Name	`http.response.version`
Description	Stores the HTTP version used in the response from the Web Service.
Type	`java.lang.String`
Generated By	**Connection** **Web Service Filter**
Consumed By	
Required By	

kerberos.client.context.established

Name	`kerberos.client.context.established`
Description	Indicates whether the client-side context has been established (true or false).
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.context

Name	`kerberos.context`
Description	Used on client-side to reload the context to consume the service-side token, if there is one.
Type	`org.ietf.jgss.GSSContext`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.mechanism.oid

Name	`kerberos.mechanism.oid`
Description	Contains the mechanism OID (`SPNEFGO` or `Kerberos`).
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication** **Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.profile.ap.req.bst.id

Name	`kerberos.profile.ap.req.bst.id`
Description	Contains the `wsu:Id` of the `BinarySecurityToken` containing the `AP_REQ` message generated by the `GssInitiatorFilter` .
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.profile.ap.req.sha1

Name	`kerberos.profile.ap.req.sha1`
Description	Contains the SHA1 hash of a Kerberos `AP_REQ` message. This is generated when a Kerberos service consumes it, or when a Kerberos client generates it.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication** **Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.authenticator.principal

Name	`kerberos.service.authenticator.principal`
Description	Contains the principal extracted from the `AP_REQ` message on the Kerberos acceptor side.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.authenticator.realm

Name	`kerberos.service.authenticator.realm`
Description	Contains the realm extracted from the `AP_REQ` message on the Kerberos acceptor side.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.authenticator.time

Name	`kerberos.service.authenticator.time`
Description	Contains the time extracted from the `AP_REQ` message on the Kerberos acceptor side.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.context.established

Name	`kerberos.service.context.established`
Description	Indicates whether the service-side context has been established (true or false).
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.subject.id

Name	`kerberos.service.subject.id`
Description	Contains the principal name of the Kerberios service.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication** **Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.ticket.principal

Name	`kerberos.service.ticket.principal`
Description	Contains the principal extracted from the AP_REQ message on the Kerberos acceptor side.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.service.ticket.realm

Name	`kerberos.service.ticket.realm`
Description	Contains the realm extracted from the `AP_REQ` message on the Kerberos acceptor side.
Type	`java.lang.String`
Generated By	**Kerberos Client Authentication**
Consumed By	
Required By	

kerberos.session.key

Name	`kerberos.session.key`
Description	On the Kerberos server-side, contains the session key extracted from the service ticket and authenticator in the Kerberos `AP_REQ` message. On the Kerberos client-side, contains the session key extracted from the private credentials of the client subject in the `KerebrosTicket` .
Type	`javax.security.auth.kerberos.KerberosKey`
Generated By	**Kerberos Client Authentication** **Kerberos Client Authentication**
Consumed By	
Required By	

mcafee.status

Name	`mcafee.status`
Description	Stores the overall message status of all message parts after a McAfee virus scan (for example, `NOVIRUS` , `INFECTED` , `REPAIRED` , and `REMOVED`).
Type	`java.lang.String`
Generated By	**McAfee Anti-Virus**
Consumed By	
Required By	

message.key

Name	`message.key`
Description	Contains a hash of the request message. By default, used as the key to search for objects in the cache.
Type	`java.lang.String`
Generated By	**Create Key**
Consumed By	

Required By	**Cache Attribute** **Is Cached?** **Remove Cached Attribute**

saml.assertion

Name	`saml.assertion`
Description	Contains the SAML assertion that was used for authentication, authorization, or attribute extraction.
Type	`org.w3c.dom.Element`
Generated By	**Retrieve Attributes from SAML Attribute Assertion** **SAML Authentication** **SAML Authorization**
Consumed By	
Required By	

saml.assertion.position

Name	`saml.assertion.position`
Description	Indicates the position of the SAML assertion within a given WS-Security block. There may be more than 1 assertion in a WS-Security block, and so this attribute can be used to select the appropriate one.
Type	`java.lang.Integer`
Generated By	**Retrieve Attributes from SAML Attribute Assertion** **SAML Authentication** **SAML Authorization**
Consumed By	
Required By	

saml.wsblockinfo

Name	`saml.wsblockinfo`
Description	Stores the WS-Security block that contains the relevant SAML assertion.
Type	`com.vordel.common.util.WSBlockInfo`
Generated By	**Retrieve Attributes from SAML Attribute Assertion** **SAML Authentication** **SAML Authorization**
Consumed By	
Required By	

samlpdp.response.assertion

Name	`samlpdp.response.assertion`
Description	Contains the SAML assertion from the SAMLP response.
Type	`org.w3c.dom.Element`
Generated By	**Retrieve Attributes from SAML Attribute Assertion** **SAML PDP Authorization**
Consumed By	
Required By	

samlpdp.response.doc

Name	`samlpdp.response.doc`
Description	Contains the SAMLP response from the SAML PDP as an XML Document.
Type	`org.w3c.dom.Document`
Generated By	**Retrieve Attributes from SAML PDP** **SAML PDP Authorization** **SAML PDP Response XML-Signature Verification** **SAML PDP XML-Signature Response Verification**
Consumed By	
Required By	

samlpdp.response.namespace.saml

Name	`samlpdp.response.namespace.saml`
Description	Stores the SAML namespace that was used in the SAMLP response.
Type	`java.lang.String`
Generated By	**Retrieve Attributes from SAML PDP** **SAML PDP Authorization**
Consumed By	
Required By	

samlpdp.response.namespace.samlp

Name	`samlpdp.response.namespace.samlp`
Description	Stores the SAMLP namespace that was used in the SAMLP response.
Type	`java.lang.String`
Generated By	**Retrieve Attributes from SAML PDP** **SAML PDP Authorization**
Consumed By	

Required By	

samlpdp.subject.format

Name	`samlpdp.subject.format`
Description	Contains the subject format used in the SAMLP request to the SAML PDP.
Type	`java.lang.String`
Generated By	**Retrieve Attributes from SAML PDP** **SAML PDP Authorization**
Consumed By	
Required By	

samlpdp.subject.id

Name	`samlpdp.subject.id`
Description	Identifies the subject used in the SAMLP request to the SAML PDP.
Type	`java.lang.String`
Generated By	**Retrieve Attributes from SAML PDP** **SAML PDP Authorization**
Consumed By	
Required By	

service.name

Name	`service.name`
Description	Stores the name of the backend Web Service. For example, this is used when the service is displayed in real-time monitoring tools.
Type	`java.lang.String`
Generated By	**Set Service Name** **Set Web Service Context** **Web Service Filter**
Consumed By	
Required By	

siteminder.agent

Name	`siteminder.agent`
Description	Indicates the name of the agent that API Gateway uses to

	connect to SiteMinder as.
Type	`java.lang.String`
Generated By	**SiteMinder Certificate Authentication** **SiteMinder Session Validation**
Consumed By	
Required By	**SiteMinder Authorization** **SiteMinder Logout**

siteminder.decision

Name	`siteminder.decision`
Description	API Gateway can ask SiteMinder to make an authorization decision based on whether or not SiteMinder authenticates the user. SiteMinder returns its decision to API Gateway where it is stored in this attribute.
Type	`com.vordel.circuit.siteminder.SiteMinderDecision`
Generated By	**SiteMinder Certificate Authentication** **SiteMinder Session Validation**
Consumed By	
Required By	**SiteMinder Authorization** **SiteMinder Logout**

soap.request.action

Name	`soap.request.action`
Description	Specifies the SOAP Action in the HTTP header.
Type	`java.lang.String`
Generated By	**SOAPAction**
Consumed By	
Required By	

soap.request.method

Name	`soap.request.method`
Description	Stores the SOAP operation name. This is the first element under the SOAP body for an RPC encoded SOAP message.
Type	`java.lang.String`
Generated By	**Operation Name**
Consumed By	
Required By	

soap.request.method.namespace

Name	`soap.request.method.namespace`
Description	Contains the namespace of the element identified by the value of the `soap.request.method attribute`. In other words, this attribute indicates the namespace of the SOAP operation.
Type	`java.lang.String`
Generated By	**Operation Name**
Consumed By	
Required By	

soasecuritymanager.action

Name	`soasecuritymanager.action`
Description	Contains the action to take.
Type	`java.lang.String`
Generated By	
Consumed By	
Required By	

soasecuritymanager.agent

Name	`soasecuritymanager.agent`
Description	Contains the name of the agent used by API Gateway to connect to the CA SOA Security Manager.
Type	`java.lang.String`
Generated By	**CA SOA Security Manager Authentication**
Consumed By	
Required By	**CA SOA Security Manager Authorization**

soasecuritymanager.decision

Name	`soasecuritymanager.decision`
Description	Contains the authentication/authorization decision made by CA SOA Security Manager.
Type	`java.lang.String`
Generated By	**CA SOA Security Manager Authentication**
Consumed By	
Required By	**CA SOA Security Manager Authorization**

soasecuritymanager.realmdef

Name	`soasecuritymanager.realmdef`
Description	Contains the authentication/authorization realm of the CA SOA Security Manager.
Type	`java.lang.String`
Generated By	**CA SOA Security Manager Authentication**
Consumed By	
Required By	**CA SOA Security Manager Authorization**

soasecuritymanager.resource

Name	`soasecuritymanager.resource`
Description	Contains the name of the resource that the client is attempting to access.
Type	`java.lang.String`
Generated By	
Consumed By	
Required By	

soasecuritymanager.resource.context

Name	`soasecuritymanager.resource.context`
Description	Describes the context of the resource that the user is attempting to access.
Type	`java.lang.String`
Generated By	**CA SOA Security Manager Authentication**
Consumed By	
Required By	**CA SOA Security Manager Authorization**

webservice.context

Name	`webservice.context`
Description	Stores the Web Service context of the backend Web Service. For example, this is used to identify the service in the Web Service Repository when a WSDL request is received.
Type	`java.lang.String`
Generated By	**Set Web Service Context** **Web Service Filter**
Consumed By	
Required By	**Return WSDL**

	Schema Validation

ws.username.token.name

Name	`ws.username.token.name`
Description	Associates the generated `UsernameToken` with the authenticated user.
Type	`java.lang.String`
Generated By	**Insert WS-Security Username Token**
Consumed By	
Required By	

wss.timestamp

Name	`wss.timestamp`
Description	The timestamp in the WSS header block that is obtained from the message for a particular SOAP actor/role. Indicates how long the security data remains valid for.
Type	`java.lang.String`
Generated By	**Extract WSS Timestamp**
Consumed By	
Required By	

wss.usernameToken

Name	`wss.usernameToken`
Description	The `usernameToken` in the WSS header block that is obtained from the message for a particular SOAP actor/role.
Type	`java.lang.String`
Generated By	**Extract WSS Username Token**
Consumed By	
Required By	

xacml.decision

Name	`xacml.decision`
Description	Contains the authorization decision sent by the XACML Policy Decision Point to the XACML Policy Enforcement Point (for example, `Permit` , `Deny` , `Indeterminate` , or `NotApplicable`).

Type	`java.lang.String`
Generated By	**XACML PEP**
Consumed By	
Required By	

xacml.result.xml

Name	`xacml.result.xml`
Description	Contains the XML response message that includes the authorization decision sent by the XACML Policy Decision Point to the XACML Policy Enforcement Point (for example, `Permit` , `Deny` , `Indeterminate` , or `NotApplicable`).
Type	`java.lang.String`
Generated By	**XACML PEP**
Consumed By	
Required By	

xacml.statuscode

Name	`xacml.statuscode`
Description	Contains the XACML status code message (for example, `urn:oasis:names:tc:xacml:1.0:status:ok` `syntax-error` , `processing-error` , or `missing-attribute`).
Type	`java.lang.String`
Generated By	**XACML PEP**
Consumed By	
Required By	

xsd.errors

Name	`xsd.errors`
Description	Stores validation errors generated when a schema validation check fails. For example, you can return an appropriate SOAP Fault to the client by writing out the contents of this attribute. You can configure a **Set Message** filter to write a custom response message back to the client, and place it on the failure path of the **Schema Validation** filter.
Type	`java.lang.String`
Generated By	**Schema Validation**
Consumed By	

Required By	

Message Filter Reference

Extract WSS Header

Name	Extract WSS Header
Description	Extracts a WS-Security `<Header>` block from a message, and stores it in the `authentication.ws.wsblockinfo` message attribute.
Category	Attributes
Required Attributes	
Consumed Attributes	
Generated Attributes	**authentication.ws.wsblockinfo**
Tutorial	*Extract WSS Header*

Extract WSS Timestamp

Name	Extract WSS Timestamp
Description	Extracts a WS-Utility Timestamp from a message. The timestamp is stored in a specified message attribute to be processed later in a policy. Defaults to the `wss.timestamp` message attribute.
Category	Attributes
Required Attributes	
Consumed Attributes	
Generated Attributes	**wss.timestamp**
Tutorial	*Extract WSS Timestamp*

Extract WSS Username Token

Name	Extract WSS Username Token
Description	Extracts a WS-Security UsernameToken from a message if it exists. The extracted UsernameToken is stored in the `wss.usernameToken` message attribute.
Category	Attributes
Required Attributes	
Consumed Attributes	
Generated Attributes	**wss.usernameToken**
Tutorial	*Extract WSS UsernameToken*

Insert SAML Attribute Assertion

Name	Insert SAML Attribute Assertion

Description	Inserts a SAML attribute assertion into the downstream message.
Category	Attributes
Required Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id** **content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Insert SAML Attribute Assertion*

Retrieve Attributes from Directory Server

Name	Retrieve Attribute from Directory Server
Description	Retrieves user attributes from an LDAP directory.
Category	Attributes
Required Attributes	**authentication.subject.id** **authentication.subject.format**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id**
Tutorial	*Retrieve Attributes from Directory Server*

Retrieve Attribute from HTTP Header

Name	Retrieve Attribute from HTTP Header
Description	Retrieves the value of an HTTP header and sets it to a user-specified message attribute.
Category	Attributes
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Retrieve Attribute from HTTP Header*

Retrieve Attributes from Database

Name	Retrieve Attributes from Database
Description	Retrieves user attributes from a specified database.
Category	Attributes
Required Attributes	**authentication.subject.id**

	authentication.subject.format
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **authentication.subject.id** **authentication.subject.format**
Tutorial	*Retrieve Attribute from Database*

Retrieve Attribute from Message

Name	Retrieve Attribute from Message
Description	Retrieves the value of an XML attribute or element from the message and sets it to a user-specified message attribute.
Category	Attributes
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Retrieve Attribute from Message*

Retrieve Attributes from SAML Attribute Assertion

Name	Retrieve Attribute from SAML Attribute Assertion
Description	Retrieves user attributes from a SAML attribute assertion and stores them in the attribute.lookup.list message attribute.
Category	Attributes
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id** **saml.assertion** **saml.assertion.position** **saml.wsblockinfo**
Tutorial	*Retrieve Attribute from SAML Attribute Assertion*

Retrieve Attributes from SAML PDP

Name	Retrieve Attribute from SAML PDP
Description	When a user has been successfully authenticated, the API Gateway can send a SAMLP (SAML Protocol) request to the SAML PDP to obtain user attributes. The PDP packages the relevant attributes into a SAML attribute assertion and returns the assertion to API Gateway in a SAMLP re-

	sponse. API Gateway validates the response and can optionally insert the attribute assertion into the downstream message.
Category	Attributes
Required Attributes	**authentication.subject.id** **authentication.subject.format** **authentication.method**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **samlpdp.response.assertion** **samlpdp.response.doc** **samlpdp.response.namespace.saml** **samlpdp.response.namespace.samlp** **samlpdp.subject.format** **samlpdp.subject.id**
Tutorial	*SAML PDP Attributes*

Retrieve Attributes from Tivoli

Name	Retrieve Attributes from Tivoli
Description	You can use this filter when you need to retrieve user attributes independently from authorizing the user against Tivoli Access Manager.
Category	Attributes
Required Attributes	**authentication.subject.id**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id**
Tutorial	*Retrieve Attributes from Tivoli*

Retrieve Attributes from User Store

Name	Retrieve from User Store
Description	Retrieves user attributes from the User Store and stores them in the attribute.lookup.list message attribute.
Category	Attributes
Required Attributes	**authentication.subject.id**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **authentication.subject.id** **authentication.subject.format**
Tutorial	*Retrieve Attribute from User Store*

SAML PDP XML-Signature Response Verification

Name	SAML PDP XML-Signature Response Verification
Description	Typically, a SAML PDP will sign SAMLP responses returned to API Gateway. In such cases, API Gateway can validate the signature on the response using this filter.
Category	Attributes
Required Attributes	**samlpdp.response.doc**
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*SAML PDP Response XML-Signature Verification*

Attribute Authentication

Name	Attribute Authentication
Description	Authenticates user credentials specified in API Gateway message attributes against a configured user store.
Category	Authentication
Required Attributes	**authentication.subject.id** **authentication.subject.password**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **authentication.subject.orig.format** **authentication.subject.orig.id** **authentication.subject.password**
Tutorial	*Attribute Authentication*

HTML Form-based Authentication

Name	HTML Form-based Authentication
Description	Authenticates API Gateway client user credentials specified in an HTML form against a configured user store.
Category	Authentication
Required Attributes	
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **authentication.subject.orig.format** **authentication.subject.orig.id** **authentication.subject.password**
Tutorial	*HTML Form-based Authentication*

HTTP Basic Authentication

Name	HTTP Basic Authentication
Description	Authenticates a client against a configured user store using HTTP basic authentication.
Category	Authentication
Required Attributes	**http.headers** **http.request.verb**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **authentication.subject.orig.format** **authentication.subject.orig.id** **authentication.subject.password**
Tutorial	*HTTP Basic Authentication*

HTTP Digest Authentication

Name	HTTP Digest Authentication
Description	Authenticates a client against a configured user store using HTTP digest authentication.
Category	Authentication
Required Attributes	**http.headers** **http.request.verb**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **authentication.subject.orig.format** **authentication.subject.orig.id** **authentication.subject.password**
Tutorial	*HTTP Digest Authentication*

IP Address

Name	IP Address
Description	Allows or denies access to an IP address or range of IP addresses.
Category	Authentication
Required Attributes	**http.request.clientaddr**
Consumed Attributes	
Generated Attributes	
Tutorial	*IP Address*

SSL Authentication

Name	SSL Authentication
Description	Authenticates a user's SSL certificate.
Category	Authentication
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**authentication.cert** **authentication.issuer.format** **authentication.issuer.id** **authentication.method** **authentication.subject.format** **authentication.subject.id** **certificate** **certificates**
Tutorial	*SSL Authentication*

CA SOA Security Manager Authentication

Name	CA SOA Security Manager Authentication
Description	Authenticates a user against CA SOA Security Manager.
Category	Authentication
Required Attributes	**content.body** **http.request.clientaddr** **http.request.uri** **http.request.verb**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **soasecuritymanager.agent** **soasecuritymanager.decision** **soasecuritymanager.realmdef** **soasecuritymanager.resource.context**
Tutorial	*CA SOA Security Manager Authorization*

HTTP Header Authentication

Name	HTTP Header Authentication
Description	Extracts a user credential from an HTTP header and uses it to authenticate the user. Typically, a username, X.509 certificate, or Distinguished Name is extracted from the HTTP header.
Category	Authentication
Required Attributes	**http.headers**
Consumed Attributes	

Generated Attributes	**authentication.cert** **authentication.issuer.format** **authentication.issuer.id** **authentication.issuer.orig.format** **authentication.issuer.orig.id** **authentication.method** **authentication.subject.format** **authentication.subject.id** **authentication.subject.orig.format** **authentication.subject.orig.id** **certificate** **certificates**
Tutorial	*HTTP Header Authentication*

Insert SAML Authentication Assertion

Name	Insert SAML Authentication Assertion
Description	Inserts a SAML authentication assertion into the downstream message on behalf of an authenticated user.
Category	Authentication
Required Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Insert SAML Authentication Assertion*

Insert WS-Security Username Token

Name	Insert WS-Security Username Token
Description	Inserts a WS-Security Token into the downstream message on behalf of an authenticated client.
Category	Authentication
Required Attributes	**authentication.subject.id** **content.body**
Consumed Attributes	
Generated Attributes	**ws.username.token.name**
Tutorial	*Insert WS-Security Username Token*

Insert Timestamp

Name	Insert Timestamp
Description	Inserts a WS-Utility (WSU) Timestamp into a WS-Security

	Header to specify the lifetime of the message to which it is added.
Category	Authentication
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Insert Timestamp*

Kerberos Client Authentication

Name	Kerberos Client Authentication
Description	Obtains a service ticket for a Kerberos Service, and uses it to authenticate to the service.
Category	Authentication
Required Attributes	**http.destination.host**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **kerberos.client.context.established** **kerberos.context** **kerberos.mechanism.oid** **kerberos.profile.ap.req.bst.id** **kerberos.profile.ap.req.sha1** **kerberos.service.subject.id** **kerberos.session.key**
Tutorial	*Kerberos Client Authentication*

Kerberos Service Authentication

Name	Kerberos Service Authentication
Description	Consumes a Kerberos token to authenticate a Kerberos Client.
Category	Authentication
Required Attributes	**http.destination.host**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **kerberos.mechanism.oid** **kerberos.profile.ap.req.sha1** **kerberos.service.authenticator.principal** **kerberos.service.authenticator.principal** **kerberos.service.authenticator.principal** **kerberos.service.context.established**

	kerberos.service.subject.id **kerberos.service.subject.id** **kerberos.service.subject.id** **kerberos.session.key**
Tutorial	*Kerberos Service Authentication*

SAML Authentication

Name	SAML Authentication
Description	Validates a SAML authentication assertion to make sure it has not expired.
Category	Authentication
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.id** **authentication.subject.format** **saml.assertion** **saml.assertion.position** **saml.wsblockinfo**
Tutorial	*SAML Authentication*

SAML Authentication XML-Signature Verification

Name	SAML Authentication XML-Signature Verification
Description	Validates the signature on a SAML authentication assertion.
Category	Authentication
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**authentication.cert** **authentication.issuer.format** **authentication.issuer.id** **certificate** **certificates**
Tutorial	*SAML Authentication XML-Signature Verification*

SAML PDP Response XML-Signature Verification

Name	SAML PDP Response XML-Signature Verification
Description	Typically a SAML PDP will sign SAMLP responses and/or the issued SAML assertion itself. This filter can be used to validate the signature on the SAMLP response.
Category	Authentication

Required Attributes	**samlpdp.response.doc**
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*SAML PDP Response XML-Signature Verification*

XML-Signature Authentication

Name	XML-Signature Authentication
Description	API Gateway can authenticate a client by validating the XML-Signature on an incoming request. A successful signature validation proves that the client had access to the private key that was used to sign the request.
Category	Authentication
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**authentication.cert** **authentication.issuer.format** **authentication.issuer.id** **authentication.method** **authentication.subject.format** **authentication.subject.id** **certificate** **certificates**
Tutorial	*XML Signature Authentication*

Attribute Authorization

Name	Attribute Authorization
Description	This filter checks the values of user attributes that are stored in the `attribute.lookup.list` message attribute.
Category	Authorization
Required Attributes	**attribute.lookup.list**
Consumed Attributes	
Generated Attributes	
Tutorial	*Attributes*

CA SOA Security Manager Authorization

Name	CA SOA Security Manager Authorization
Description	Authorizes an authenticated user against CA SOA Security Manager.

Category	Authorization
Required Attributes	**http.request.clientaddr** **soasecuritymanager.agent** **soasecuritymanager.decision** **soasecuritymanager.realmdef** **soasecuritymanager.resource.context**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id**
Tutorial	*CA SOA Security Manager Authentication*

Certificate Attributes Authorization

Name	Certificate Attributes Authorization
Description	Authorizes a user by examining the attributes in that user's X.509 certificate.
Category	Authorization
Required Attributes	**authentication.subject.id** **authentication.subject.format**
Consumed Attributes	
Generated Attributes	
Tutorial	*Certificate Attributes*

Check Group Membership

Name	Check Group Membership
Description	Checks whether the specified API Gateway User is a member of the specified API Gateway Group.
Category	Authorization
Required Attributes	**authentication.subject.id**
Consumed Attributes	
Generated Attributes	
Tutorial	*Check Group Membership*

Entrust GetAccess Authorization

Name	GetAccess Authorization
Description	Authorizes an authenticated user against Entrust's GetAccess.
Category	Authorization
Required Attributes	**authentication.subject.id**

	authentication.subject.format content.body
Consumed Attributes	
Generated Attributes	
Tutorial	*Entrust GetAccess Authorization*

Insert SAML Authorization Assertion

Name	Insert SAML Authorization Assertion
Description	When the user has been successfully authorized, API Gateway can insert a SAML authorization assertion into the downstream message.
Category	Authorization
Required Attributes	**authentication.subject.id** **authentication.subject.format** **content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Insert SAML Authorization Assertion*

RSA Access Manager Authorization

Name	Access Manager
Description	Authorizes an authenticated user against RSA's ClearTrust Authorization Server.
Category	Authorization
Required Attributes	**authentication.subject.id** **authentication.subject.format**
Consumed Attributes	
Generated Attributes	
Tutorial	*RSA Access Manager Authorization*

SAML Authorization

Name	SAML Authorization
Description	Authorizes a user by validating the SAML authorization assertion in an incoming request.
Category	Authorization
Required Attributes	**authentication.subject.id** **authentication.subject.format** **content.body**
Consumed Attributes	

Generated Attributes	**saml.assertion** **saml.assertion.position** **saml.wsblockinfo**
Tutorial	*SAML Authorization Assertion*

SAML Authorization XML-Signature Verification

Name	SAML Authorization XML-Signature Verification
Description	Validates the XML-Signature on a SAML authorization assertion.
Category	Authorization
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*SAML Authorization XML-Signature Verification*

SAML PDP Authorization

Name	SAML PDP Authorization
Description	Generates a SAMLP authorization request to a SAML PDP on behalf of an authenticated user. The SAML PDP generates a SAML authorization assertion and returns it to API Gateway in a SAMLP response. API Gateway validates the response and can optionally insert the assertion into the downstream message.
Category	Authorization
Required Attributes	**authentication.method** **authentication.subject.id** **authentication.subject.format**
Consumed Attributes	
Generated Attributes	**samlpdp.response.assertion** **samlpdp.response.doc** **samlpdp.response.namespace.saml** **samlpdp.response.namespace.samlp** **samlpdp.subject.id** **samlpdp.subject.format**
Tutorial	*SAML PDP Authorization*

SAML PDP Response XML-Signature Verification

Name	SAML PDP Response XML-Signature Verification
Description	Typically a SAML PDP will sign SAMLP responses and/or the issued SAML assertion itself. This filter can be used to

	validate the signature on the SAMLP response.
Category	Authorization
Required Attributes	**samlpdp.response.doc**
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*SAML PDP Response XML-Signature Verification*

Tivoli Authorization

Name	Tivoli Authorization
Description	Authorizes an authenticated user against IBM's Tivoli Access Manager.
Category	Authorization
Required Attributes	**authentication.subject.id** **authentication.subject.format**
Consumed Attributes	
Generated Attributes	
Tutorial	*Tivoli Authorization*

XACML PEP

Name	XACML PEP
Description	Configures an eXtensible Access Control Markup Language (XACML) Policy Enforcement Point (PEP)
Category	Authorization
Required Attributes	
Consumed Attributes	
Generated Attributes	**xacml.decision** **xacml.result.xml** **xacml.statuscode**
Tutorial	*XACML Policy Enforcement Point*

SiteMinder Authorization

Name	SiteMinder Authorization
Description	Authorizes a user against CA's SiteMinder. The user must have been authenticated to SiteMinder before they can be authorized.
Category	CA SiteMinder
Required Attributes	**siteminder.agent**

	siteminder.decision
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id**
Tutorial	*SiteMinder Authorization*

SiteMinder Certificate Authentication

Name	SiteMinder Certificate Authentication
Description	Authenticates a user's certificate against SiteMinder.
Category	CA SiteMinder
Required Attributes	**certificate** **certificates**
Consumed Attributes	
Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **siteminder.agent** **siteminder.decision**
Tutorial	*SiteMinder Certificate Authentication*

SiteMinder Logout

Name	SiteMinder Logout
Description	Terminates a user's SiteMinder session by invalidating the user's single sign-on token.
Category	CA SiteMinder
Required Attributes	**siteminder.agent** **siteminder.decision**
Consumed Attributes	
Generated Attributes	
Tutorial	*SiteMinder Logout*

SiteMinder Session Validation

Name	SiteMinder Session Validation
Description	Extracts a user's single sign-on token from the message and validates it against SiteMinder.
Category	CA SiteMinder
Required Attributes	**content.body**
Consumed Attributes	

Generated Attributes	**authentication.method** **authentication.subject.format** **authentication.subject.id** **siteminder.agent** **siteminder.decision**
Tutorial	*SiteMinder Session Validation*

Cache Attribute

Name	Cache Attribute
Description	Specifies which part of the message is cached. Typically, response messages are cached, so this filter is usually configured after the routing filters in a policy.
Category	Cache
Required Attributes	**message.key** **content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Cache Attribute*

Create Key

Name	Create Key
Description	Specifies which part of a message determines if the message is unique (for example, message body, HTTP header, client IP address, and so on).
Category	Cache
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**message.key**
Tutorial	*Create Key*

Is Cached?

Name	Is Cached?
Description	Looks up a named cache to see if a specified message attribute is already cached. A message attribute is used as the key to search for in the cache (defaults to `message.key`). If the lookup succeeds, the retrieved value overrides a specified message attribute (defaults to `content.body`).
Category	Cache
Required Attributes	**message.key**

Consumed Attributes	
Generated Attributes	**content.body**
Tutorial	*Is Cached?*

Remove Cached Attribute

Name	Remove Cached Attribute
Description	Deletes a message attribute value that has been stored in a cache.
Category	Cache
Required Attributes	**message.key**
Consumed Attributes	
Generated Attributes	
Tutorial	*Removed Cached Attribute*

Certificate Chain

Name	Certificate Chain
Description	Ensures that a trusted CA (Certificate Authority) issued the certificate. Trusted CA certificates are stored in the Certificate Store.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*Certificate Chain Check*

Certificate Revocation List (Dynamic)

Name	Certificate Revocation List (dynamic)
Description	Validates a certificate against a CRL and automatically retrieves the CRL periodically.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*Dynamic CRL Certificate Validation*

Certificate Revocation List (LDAP)

Name	CRL (in LDAP)
Description	Looks up a user's certificate in an LDAP-based CRL to see if that user has been revoked.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*CRL LDAP Validation*

Certificate Revocation List Responder

Name	CRL Responder
Description	Configures the API Gateway to act as CRL responder by returning CRL files to clients.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*CRL Responder*

Certificate Revocation List (Static)

Name	CRL (static)
Description	Looks up a user's certificate in a file-based CRL to see if that user has been revoked.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*Static CRL Certificate Validation*

Create Thumbprint from Certificate

Name	Create Thumbprint
Description	Used to create a human-readable thumbprint (or fingerprint) from the X.509 certificate that is stored in the `certificate` message attribute. The generated thumbprint is stored in the `certificate.thumbprint` attribute.
Category	Certificate
Required Attributes	**certificate**
Consumed Attributes	

Generated Attributes	**certificate.thumbprint**
Tutorial	**certificate.thumbprint**

Extract Certificate Attributes

Name	Extract Certificate Attributes
Description	Extracts the X.509 attributes from a certificate stored in a specified Oracle message attribute. Typically, this filter is used in conjunction with a **Find Certificate** filter.
Category	Certificate
Required Attributes	**certificate**
Consumed Attributes	
Generated Attributes	**attribute.lookup.list** **attribute.subject.format** **attribute.subject.id** **cert.basic.constraints** **cert.extended.key.usage** **cert.hash.md5** **cert.hash.sha1** **cert.issuer.alternative.name** **cert.issuer.id** **cert.issuer.id.c** **cert.issuer.id.cn** **cert.issuer.id.emailaddress** **cert.issuer.id.l** **cert.issuer.id.o** **cert.issuer.id.ou** **cert.issuer.id.st** **cert.key.usage.cRLSign** **cert.key.usage.dataEncipherment** **cert.key.usage.digitalSignature** **cert.key.usage.encipherOnly** **cert.key.usage.keyAgreement** **cert.key.usage.keyCertSign** **cert.key.usage.keyEncipherment** **cert.key.usage.nonRepudiation** **cert.not.after** **cert.not.before** **cert.serial.number** **cert.signature.algorithm** **cert.subject.alternative.name** **cert.subject.id** **cert.subject.id.c** **cert.subject.id.cn** **cert.subject.id.emailaddress** **cert.subject.id.o** **cert.subject.id.ou** **cert.subject.id.st** **cert.version**
Tutorial	*Extract Certificate Attributes*

Find Certificate

Name	Find Certificate
Description	Locates a certificate from a message attribute, HTTP header, message attachment, or extracts a certificate from the User Store. The extracted certificate is stored in a user-specified message attribute. This new attribute will then appear as a **Generated Attribute** in the policy. The certificate is stored in the certificate attribute by default.
Category	Certificate
Required Attributes	
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*Find Certificate*

OCSP (Online Certificate Status Protocol)

Name	OCSP Certificate Validation
Description	Checks the status of a user's certificate against a group of OCSP responders.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*OCSP Certificate Validation*

XKMS (XML Key Management and Security)

Name	XKMS Certificate Validation
Description	Validates a user's certificate against a group of XKMS responders.
Category	Certificates
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	
Tutorial	*XKMS Certificate Validation*

Content Type Filtering

Name	Content Type Filtering
Description	Filters MIME and DIME messages based on the types of their attachments.

Category	Content Filtering
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Content Type Filtering*

Content Validation

Name	Content Validation
Description	Runs a boolean XPath expression on the incoming request.
Category	Content Filtering
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Content Validation*

Throttling

Name	Throttling
Description	Limits the number of messages a client can send in a specified interval through the policy in which this filter is configured. In other words, it provides filtering of messages on a per client, per service basis.
Category	Content Filtering
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*Throttling*

McAfee Anti-Virus

Name	McAfee Anti-Virus
Description	Scans incoming HTTP requests and their attachments for viruses and exploits. Supports cleaning of messages from infections, provides scan presets for detection levels, and reports overall message status after scanning.
Category	Content Filtering
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**mcafee.status**

Tutorial	*McAfee Anti-Virus*

Schema Validation

Name	Schema Validation
Description	Validates the contents of the message body against a selected XML Schema. This ensures that the message adheres to the correct message format, and can also ensure that the message contains appropriate data.
Category	Content Filtering
Required Attributes	**content.body** **webservice.context**
Consumed Attributes	
Generated Attributes	**xsd.errors**
Tutorial	*Schema Validation*

Validate HTTP Headers

Name	Validate HTTP Headers
Description	Filters MIME and DIME messages based on the types of their attachments.
Category	Content Filtering
Required Attributes	**http.headers**
Consumed Attributes	
Generated Attributes	
Tutorial	*HTTP Header Validation*

Validate Message Attribute

Name	Validate Message Attribute
Description	Compares the value of a message attribute to a configured regular expression. It can also check for the presence of **Threatening Content** regular expressions such as SQL injection and buffer overflow attacks.
Category	Content Filtering
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Validate Message Attributes*

XML Complexity

Name	XML Complexity
Description	Checks the depth and complexity of XML messages.
Category	Content Filtering
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*XML Complexity*

Add HTTP Header

Name	Add HTTP Header
Description	Adds a user-specified HTTP header to the downstream message.
Category	Conversion
Required Attributes	**http.headers**
Consumed Attributes	
Generated Attributes	
Tutorial	*Add HTTP Header*

Set Message

Name	Set Message
Description	Sets the content of the message payload.
Category	Conversion
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**content.body**
Tutorial	*Set Message*

Load File

Name	Load file
Description	Loads the contents of the specified file, and sets them as message content to be processed.
Category	Conversion
Required Attributes	
Consumed Attributes	
Generated Attributes	**content.body**
Tutorial	*Load File*

Remove HTTP Header

Name	Remove HTTP Header
Description	Removes a user-specified HTTP header from the downstream message.
Category	Conversion
Required Attributes	**http.headers**
Consumed Attributes	
Generated Attributes	
Tutorial	*Remove HTTP Header*

XSLT Transformation

Name	XSLT Transformation
Description	This filter uses an XSLT stylesheet to convert the body of the incoming request to an alternative XML grammar or format.
Category	Conversion
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*XSLT Transformation*

XML-Decryption

Name	XML-Decryption
Description	Decrypts an XML-Encrypted message according to the settings configured in the **XML-Decryption Settings** filter. These settings are stored in the `decryption.properties` message attribute, which is a required attribute for this filter.
Category	Encryption
Required Attributes	**content.body** **decryption.properties**
Consumed Attributes	
Generated Attributes	
Tutorial	*XML-Decryption*

XML-Decryption Settings

Name	XML-Decryption Settings
Description	This filter is used to specify the XML-Encrypted blocks to decrypt in the message. All of the encrypted blocks can be

	decrypted or a single encrypted block can be selected using an XPath expression. The actual decryption is performed by the **XML-Decryption** filter.
Category	Encryption
Required Attributes	
Consumed Attributes	
Generated Attributes	**decryption.properties**
Tutorial	*XML-Decryption Settings*

XML-Encryption

Name	XML-Encryption
Description	Encrypts (part of) an XML message as specified in the **XML Encryption Settings** filter. The message will be encrypted such that only its intended recipients can decrypt it.
Category	Encryption
Required Attributes	**content.body** **encryption.properties**
Consumed Attributes	
Generated Attributes	
Tutorial	*XML-Encryption*

XML-Encryption Settings

Name	XML-Encryption Settings
Description	This filter is used to specify the part(s) of the message to encrypt, and for whom the message is to be encrypted. Only the intended recipients will be able to decrypt the message.
Category	Encryption
Required Attributes	
Consumed Attributes	
Generated Attributes	**encryption.properties**
Tutorial	*XML-Encryption Settings*

SOAP Fault

Name	SOAP Fault
Description	If a **SOAP Fault** handler is configured for a policy, it handles any exceptions that occur in the policy. As such it dictates the format of the SOAP Fault that is returned to the client.

Category	Fault Handlers
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*SOAP Fault*

Sign Message

Name	XML Signature Generation
Description	Signs the selected part of the incoming request.
Category	Integrity
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	
Tutorial	*XML Signature Generation*

XML-Signature Verification

Name	XML-Signature Verification
Description	Verifies the integrity of the incoming message by validating its XML-Signature. This ensures that the message was not tampered with after it was signed.
Category	Integrity
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**certificate** **certificates**
Tutorial	*XML Signature Verification*

Set Service Name

Name	Set Service Name
Description	Configures service-level monitoring details. For example, you can specify the service name displayed in real-time monitoring tools, and whether to store service usage metrics.
Category	Monitoring
Required Attributes	
Consumed Attributes	
Generated Attributes	**service.name**
Tutorial	*Set Service Context*

Alert

Name	Alert
Description	Sends an alert to a configured alerting destination.
Category	Monitoring
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*System Alerting*

Log Access

Name	Log Access
Description	Logs message details in Common Log Format to an Access Log. The log file is written to the /logs directory of your API Gateway installation.
Category	Monitoring
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Log Access Filter*

Log Message Payload

Name	Log Message Payload
Description	Logs the message payload, including HTTP headers and MIME/DIME attachments, at a particular point in the policy.
Category	Monitoring
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Log Message Payload*

Operation Name

Name	Operation Name
Description	Compares the first element of the SOAP body (i.e. the SOAP operation) and its namespace to the values configured here.
Category	Resolvers
Required Attributes	**content.body**

Consumed Attributes	
Generated Attributes	**soap.request.method** **soap.request.method**
Tutorial	*Operation Name*

Relative Path

Name	Relative Path
Description	Matches the relative path (URI) on which the request was received to the value configured here.
Category	Resolvers
Required Attributes	**http.request.uri**
Consumed Attributes	
Generated Attributes	
Tutorial	*Relative Path Resolver*

SOAPAction

Name	SOAPAction
Description	Matches the SOAPAction HTTP header on the incoming request to the value configured in this filter.
Category	Resolvers
Required Attributes	**http.headers**
Consumed Attributes	
Generated Attributes	**soap.request.action**
Tutorial	*SOAP Action Resolver*

Connection

Name	Connection
Description	This filter is responsible for connecting to the target Web Service or system. API Gateway can mutually authenticate to the endpoint using SSL certificates or HTTP basic/digest authentication.
Category	Routing
Required Attributes	**content.body** **http.destination.host** **http.destination.port** **http.destination.protocol** **http.headers** **http.request.uri** **http.request.verb**

Consumed Attributes	
Generated Attributes	**http.headers** **http.request.connection.error** **http.response.info** **http.response.status** **http.response.version**
Tutorial	*Connection*

Dynamic Router

Name	Dynamic Router
Description	In cases where API Gateway is acting as a proxy, it can extract the URL from the request line of the HTTP request and route the message to this address.
Category	Routing
Required Attributes	**http.request.uri**
Consumed Attributes	
Generated Attributes	**http.destination.host** **http.destination.port** **http.destination.protocol**
Tutorial	*Dynamic Router*

Rewrite URL

Name	Rewrite URL
Description	In cases where API Gateway is acting as a proxy, it can forward the message on to the address specified in the request line of the HTTP request. This filter can be used to rewrite the URL of the original request to an alternative one, i.e. service virtualization.
Category	Routing
Required Attributes	**http.request.uri**
Consumed Attributes	
Generated Attributes	
Tutorial	*Rewrite URL*

Static Router

Name	Static Router
Description	The static router is used to configure connection details for a particular endpoint. API Gateway will route messages to the endpoint configured here.
Category	Routing

Required Attributes	
Consumed Attributes	
Generated Attributes	**http.destination.host** **http.destination.port** **http.destination.protocol**
Tutorial	*Static Router*

Insert WS-Addressing

Name	Insert WS-Addressing
Description	Inserts WS-Addressing information into a SOAP message.
Category	Routing
Required Attributes	**http.destination.host** **http.destination.port** **http.destination.protocol** **http.headers** **http.request.uri** **soap.request.action**
Consumed Attributes	
Generated Attributes	
Tutorial	*Insert WS-Addressing*

Read WS-Addressing

Name	Read WS-Addressing
Description	Uses WS-Addressing information contained within a SOAP message to route the message.
Category	Routing
Required Attributes	**content.body**
Consumed Attributes	
Generated Attributes	**http.destination.host** **http.destination.port** **http.destination.protocol** **http.request.uri**
Tutorial	*Read WS-Addressing*

Save to File

Name	Save to file
Description	Writes the current message contents to a file.
Category	Routing
Required Attributes	**content.body**

Consumed Attributes	
Generated Attributes	
Tutorial	*Save to File*

Abort

Name	Abort
Description	Forces a policy path to abort and throw an exception. This causes a SOAP Fault to be returned to the client.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Abort Filter*

Copy/Modify Attributes

Name	Copy/Modify Attributes
Description	This filter can be used to copy the value of one message attribute to another.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Copy/Modify Attributes*

False

Name	False
Description	Forces the policy path to return false.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*False Filter*

Policy Shortcut

Name	Policy Shortcut

Description	This filter can be used to pass control to another policy. It is very useful for creating a **policy macro** that contains small pieces of logic that you may wish to keep outside of a policy so that it can be re-used. This helps to keep the main logic of a policy uncluttered.
Category	Utility
Required Attributes	The required attributes for this filter are whatever attributed are required by the **start node** of the policy shortcut.
Consumed Attributes	
Generated Attributes	The generated attributes for this filter are the attributes that are returned from the **end node** of the policy shortcut.
Tutorial	*Policy Shortcut*

Reflect

Name	Reflect
Description	Echoes the request body back to the client.
Category	Utility
Required Attributes	**content.body** **http.headers**
Consumed Attributes	
Generated Attributes	**http.response.status**
Tutorial	*Reflect Message Filter*

Reflect Message and Attributes

Name	Reflect Message and Attributes
Description	Echoes the request body and the current message attributes back to the client.
Category	Utility
Required Attributes	**content.body** **http.headers**
Consumed Attributes	
Generated Attributes	
Tutorial	*Reflect Message And Attributes Filter*

True

Name	True
Description	Forces a true result from a policy path.
Category	Utility

Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*True Filter*

Execute process

Name	Execute process
Description	Executes an external process from a policy.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Execute External Process*

Pause

Name	Pause
Description	This filter forces the policy to suspend processing for a specified time interval. When this interval has elapsed, the next filter in the policy path is executed immediately.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Pause Filter*

Set Response Status

Name	Set Response Status
Description	Explicitly sets the response status of a given message.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Set Response Status*

String Replace

Name	String Replace
Description	Replaces all or part of the value of the specified string in a message attribute.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*String Replace Filter*

Trace

Name	Trace
Description	Forces the API Gateway to trace the current message attributes to the configured trace destination. By default, trace files are written to the /trace directory of your API Gateway installation.
Category	Utility
Required Attributes	
Consumed Attributes	
Generated Attributes	
Tutorial	*Trace Filter*

Return WSDL

Name	Return WSDL
Description	Returns a WSDL file from the Web Services Repository. This filter is configured automatically when a WSDL file is imported into the repository.
Category	Web Service
Required Attributes	**http.headers** **http.request.uri** **webservice.context**
Consumed Attributes	
Generated Attributes	
Tutorial	*Return WSDL*

Set Web Service Context

Name	Set Web Service Context
Description	Specifies which service to take resources from in the Web Service Repository. For example, if you set this filter to a `getQuote` service in the repository, and configure the **Re-**

	turn WSDL filter, the WSDL definition for the `getQuote` service is returned when a WSDL request is received.
Category	Web Service
Required Attributes	
Consumed Attributes	
Generated Attributes	**service.name** **webservice.context**
Tutorial	*Set Web Service Context*

Web Service Filter

Name	Web Service Filter
Description	Controls and validates requests to the Web Service and responses from the Web Service. This is automatically generated as the **Service Handler** when a WSDL file is imported into the Web Services Repository.
Category	Web Service
Required Attributes	**content.body** **http.headers** **http.request.verb**
Consumed Attributes	
Generated Attributes	**http.destination.host** **http.destination.port** **http.destination.protocol** **http.headers** **http.request.connection.error** **http.request.uri** **http.response.info** **http.response.status** **http.response.version** **service.name** **webservice.context**
Tutorial	*Web Service Filter*

WS-Policy Reference

Asymmetric Binding WS-Policies

WS-Policy Name	*Description*
AsymmetricBinding with Encrypted UsernameToken	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509v3 tokens to sign and encrypt the message. An encrypted `UsernameToken` with hash password must be included in all messages from the client to the server.
AsymmetricBinding with SAML 1.1 (Sender Vouches) Assertion and Signed Supporting Token	The service is secured with an `AsymmetricBinding` where the client and server use their respective X.509v3 certificates to secure the message. The client must include a SAML 1.1 Assertion (sender vouches) in all messages it sends to the service.
AsymmetricBinding with Signed and Encrypted UsernameToken	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509v3 tokens to sign and encrypt the message. A signed and encrypted `UsernameToken` with plaintext password must be included in all messages from the client to the service.
AsymmetricBinding with WSS 1.0 Mutual Authentication with X509 Certificates, Sign, Encrypt	The service exposes an `AsymmetricBinding` interface where the client and server use their respective X.509v3 certificates for mutual authentication, signing, and encrypting.
AsymmetricBinding with X509v3 Tokens	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509v3 tokens to sign and encrypt the message.

Message Level WS-Policies

WS-Policy Name	*Description*
Encrypt SOAP Body	The SOAP body must be encrypted.
Sign and Encrypt SOAP Body	The SOAP body must be signed and encrypted.
Sign SOAP Body	The SOAP body must be signed.

Oracle Web Services Manager WS-Policies

WS-Policy Name	*Description*
WS-Security 1.0 Mutual Auth with Certificates	`AsymmetricBinding` where the client and server use their respective X.509v3 certificates to secure the message.
WS-Security 1.0 SAML with Certificates	`AsymmetricBinding` with SAML assertion as `SignedSupportingToken`.
WS-Security 1.0 Username with Certificates	`AsymmetricBinding` with WS-Security `UsernameToken` as `SignedSupportingToken`.
WS-Security 1.1 Mutual Auth with Certificates	`SymmetricBinding` where the same X.509v3 certificate is used to secure all messages between the client and the

	service.
WS-Security 1.1 Username with Certificates	SymmetricBinding with a WS-Security UsernameToken as a SignedSupportingToken. The message is endorsed with an asymmetric Signature.
WS-Security SAML Token Over SSL	TransportBinding with a SAML Token as a SupportingToken.
WS-Security UsernameToken Over SSL	TransportBinding with a WS-Security UsernameToken as a SupportingToken.

Simple WS-Policies

WS-Policy Name	*Description*
SAML 1.1 Bearer	The client must include a SAML 1.1 Assertion (bearer) representing the Requestor in all messages from the client to the service.
Username SupportingToken Hash Password	The client must authenticate with a WS-Security SAML UsernameToken with hash password.
Username SupportingToken No Password	The client must authenticate with a WS-Security UsernameToken without a password.
Username SupportingToken Plaintext Password	The client must authenticate with a WS-Security UsernameToken with a plaintext password.

Symmetric Binding WS-Policies

WS-Policy Name	*Description*
SymmetricBinding with SAML 2.0 (Sender Vouches) Assertion and Endorsing Supporting Token	The service exposes a SymmetricBinding that requires the client to send a SAML 2.0 Assertion to the service. An X.509v3 token is also included in all messages from the client to the service as an EndorsingSupportingToken.
SymmetricBinding with Signed and Encrypted UsernameToken	The service uses a SymmetricBinding where the client and service use the same X.509v3 token to sign and encrypt the message. A signed and encrypted UsernameToken with plaintext password must be included in all messages from the client to the service. The policy uses WSS SOAP Message Security 1.1 options.
SymmetricBinding with WSS 1.1 Anonymous Authentication with X.509v3, Sign, Encrypt	The service is secured by a SymmetricBinding where the same X.509v3 certificate is used to secure all messages between the client and the service. Derived Keys are used for signing and encrypting and Signature Confirmation is required by the Policy.
SymmetricBinding with WSS 1.1 Mutual Authentication with X.509v3, Sign, Encrypt	The service exposes a SymmetricBinding where the same X.509v3 certificate is used to secure all messages between the client and the service. The client also endorses the primary message signature using another X.509v3 certificate.

Transport Binding WS-Policies

WS-Policy Name	*Description*
SAML 1.1 Holder-of-Key over SSL	The client includes a SAML 1.1 `Assertion` (sender vouches) in all messages from the client to the service. The client provides an endorsing signature to prove that it is the holder-of-key. A `TransportBinding` is used to sign and encrypt the message.
SAML 1.1 Sender-Vouches over SSL	The client includes a SAML 1.1 `Assertion` (sender vouches) on behalf of the Requestor to all messages from the client to the service. The service uses a `TransportBinding` to ensure that all messages are signed and encrypted.
SAML 2.0 Holder-of-Key over SSL	The client includes a SAML 2.0 `Assertion` (sender vouches) in all messages from the client to the service. The client provides an endorsing signature to prove that it is the holder-of-key. A `TransportBinding` is used to sign and encrypt the message.
SAML 2.0 Sender-Vouches over SSL	The client includes a SAML 2.0 Assertion (sender vouches) on behalf of the Requestor to all messages from the client to the service. The service uses a `TransportBinding` to ensure that all messages are signed and encrypted.
SSL Transport Binding	The service is secured by SSL (HTTPS).
Username Token over SSL with no Timestamp	The service is secured over SSL (HTTPS), the client is authenticated with a `UsernameToken`, and no timestamp should be included in the Security header.
Username Token over SSL with Timestamp	The service is secured over SSL (HTTPS), the client is authenticated with a `UsernameToken`. The Security header contains a timestamp.

Glossary of Terms

ASN.1

ASN.1 (Abstract Syntax Notation One) is a standard format for transmitting messages over a network. The standard defines a syntax for describing the structure of a message, and also rules for encoding the various data types contained within the message. ASN.1 is defined in the following ISO standards:

- ISO 8824/ITU X.208 specifies the ASN.1 syntax.
- ISO 8825/ITU X.209 specifies the encoding rules for ASN.1

Base64

A method of encoding 8-bit characters as ASCII printable characters. It is typically used to encode binary data so that it may be sent over text-based protocols such as HTTP and SMTP. Base64 is a scheme where 3 bytes are concatenated, then split to form 4 groups of 6-bits each; and each 6-bits gets translated to an encoded printable ASCII character, via a table lookup. The specification is described in RFC 2045.

CA

A Certificate Authority (CA) issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate.

cacert

A file used to keep the root certificates of signing authorities. The default password is *changeit*. It is typically stored in `c:\jdk1.6\jre\lib\security\cacerts`. Each entry is identified by a unique alias, and is either a key entry or a certificate entry. Key entries consist of a key pair, whereas certificate entries consist of just a certificate.

Since you implicitly trust all the Certificate Authorities in the cacerts file for code signing and verification, you must manage the cacerts file carefully. The cacerts file should contain only certificates of the CAs you trust.

CRL

A Certificate Revocation List is a signed list indicating a set of certificates that are no longer considered valid by the certificate issuer. CRLs may be used to identify revoked public-key certificates or attribute certificates and may represent revocation of certificates issued to authorities or to users. The term CRL is also commonly used as a generic term applying to different types of revocation lists.

DER

Distinguished Encoding Rules is a type of ASN.1 encoding and is widely used to define the format of X.509 certificates.

DName

An identifier that uniquely represents an object in the X.500 Directory Information Tree (DIT). A DName is a set of attribute values that identify the path leading from the base of the DIT to the object that is named. An X.509 public-key certificate or CRL contains a DName that identifies its issuer, and an X.509 attribute certificate contains a DN or other form of name that identifies its subject.

DOM

Document Object Model (DOM) is a generic interface (platform- and language-neutral) that allows external programs to edit a document's contents, structure, and style.

DTD

A Document Type Definition defines a formal grammar for specifying the structure of an XML document. An XML document is said to be *valid* if it conforms to the syntactic rules specified in the DTD.

ISO

ISO is a worldwide consortium of national standards bodies from more than 140 countries. The goal of ISO is to promote standardization in the world with a view to facilitating the international exchange of goods and services, and to develop cooperation in scientific, technological and economic activity.

keystore

The keystore file of the JDK contains your public and private keys. It has a file name ".keystore", the peculiar leading dot makes the file read-only in Unix. It is stored in PKCS #12 format, contains both public and private keys, and is protected by a passphrase.

LDAP

LDAP is a "lightweight" version of Directory Access Protocol (DAP), which is part of X.500, a standard for directory services in a network. An LDAP directory stores information on resources in a hierarchical fashion. This makes data retrieval very efficient.

OCSP

Online Certificate Status Protocol is an automated certificate checking network protocol. A client will query the OCSP responder for the status of a certificate. The responder returns whether the certificate is still trusted by the CA that issued it.

PEM

PEM (Privacy Enhanced Mail) was originally intended for securing Internet mail through authentication, message integrity and confidentiality using various encryption techniques. Its scope was widened in later years for use in a broader range of applications, such as Web Servers. Its format is essentially a base64-encoded certificate wrapped in *BEGIN CERTIFCATE* and *END CERTIFICATE* directives.

PKCS#12

PKCS#12 is a standard for storing private keys and certificates securely. It is used in (among other things) Netscape and Microsoft Internet Explorer with their import and export options.

Private-Key

The secret component of a pair of cryptographic keys used for asymmetric cryptography.

Public-Key

The publicly-disclosable component of a pair of cryptographic keys used for asymmetric cryptography.

SAML

Security Assertion Markup Language (SAML) is an XML standard for establishing trust between entities. SAML assertions can contain identity information about users (authentication assertions) and also information about the access permissions of users (authorization assertions). The basic idea is that when a user is authenticated at one site, that site issues a SAML authentication assertion and gives it to the user. The user can then use this assertion in requests at other affiliated sites. These sites need only check the details contained within the authentication assertion in order to authenticate the user. In this way, SAML allows authentication and authorization information to be shared between separate sites.

SAX

The Simple API for XML is an interface that allows applications to read in XML data. SAX is an *event-based* interface, which means that it responds to certain *events*. SAX is a read-only interface, which means that it cannot be used to generate XML elements in the same way that the DOM can. However, it is generally more efficient than DOM for reading XML documents since it does not keep the entire XML tree in memory like DOM parsing does.

Signature

A value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity.

SOAP

SOAP, or Simple Object Access Protocol is an XML-based object invocation protocol. SOAP was originally developed for distributed applications to communicate over HTTP and through corporate firewalls. SOAP defines the use of XML and HTTP to access services, objects and servers in a platform-independent manner. SOAP provides a way to access services, objects, and servers in a completely platform-independent manner. SOAP is a wire protocol that can be used to facilitate highly ultra-distributed architecture.

SOAP is simple. It is nothing more and nothing less than a protocol that defines how to access services, objects, and servers in a platform-independent manner using HTTP (also SMTP) and XML. See the Simple Object Access Protocol Specification [http://www.w3.org/TR/SOAP/] for more details.

SSL

Secure Sockets Layer (SSL) is an encrypted communications protocol for sending information securely across the Internet. It sits just above the transport layer, and below the application layer and transparently handles the encryption and decryption of data when a client establishes a secure connection to the server. It optionally provides peer entity authentication between client and server.

TLS

Transport Layer Security is the successor to SSL 3.0. Like SSL, it allows applications to communicate over a secure channel.

UDDI

Universal Description, Discovery, and Integration (UDDI) is an XML-based lookup service for locating Web Services in an Internet scenario. See the Universal Description Discovery Integration (UDDI) standard [http://www.uddi.org/] for more details.

URI

Uniform Resource Identifiers (URIs), are a platform-independent way to specify a file or resource somewhere on the web. Strictly speaking, every URL is also a URI, but not every URI is also a URL. Two RFCs specify the format of a URI:

- RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax [http://www.faqs.org/rfcs/rfc2396.html]
- RFC 2732: Format for Literal IPv6 Addresses in URIs [http://www.faqs.org/rfcs/rfc2732.html]

WSDL

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME. See the Web Services Description Language Specification [http://www.w3.org/TR/wsdl] for more details.

X509

X509 is the standard which defines the contents and data format of a public-key certificate.

XKMS

XML Key Management Specification (XKMS) uses the relative simplicity of XML to provide key management services so that a Web Service can query the trustworthiness of a user's certificate over the Internet. XKMS aims to simplify application building by separating digital-signature handling and encryption from the applications themselves. See the XML Key Management Specification [http://www.w3.org/TR/xkms/] for more details.

XPath

XML Path Language (XPath), is a language that describes how to locate and process specific parts of an XML document. See the XML Path Language Specification [http://www.w3.org/TR/xpath] for more details.

XSL

XML Stylesheet Language is used to convert XML documents into different formats, the most common of which is HTML. In a typical scenario, an XML document will reference an XSL stylesheet, which will define how the XML elements of the document should be displayed as HTML. Therefore, a clear separation of content and presentation is achieved.

XSLT

Extensible Stylesheet Language Transformations are used to convert XML documents into other formats.