Oracle® Fusion Middleware

Oracle API Gateway User Guide
11g Release 2 (11.1.2.3.0)

April 2014

Oracle API Gateway User Guide, 11g Release 2 (11.1.2.3.0)

30 April 2014

Contents

1. Getting Started .....

1. API Gateway management with Policy Studio 29
    - Overview 29
    - API Gateway instances and groups 29
    - Topology view 29
        - API Gateway groups 29
        - API Gateway instances 30
    - Filters 30
    - Policies 30
    - Message attributes 31
    - Selectors 33
    - Faults and errors 33
    - Policy shortcuts 34
    - Alerts 34
    - Policy containers 35
    - Policy contexts 35
    - Listeners 35
    - Remote hosts 35
    - Servlet applications 36
    - Service virtualization 36
2. Start the API Gateway tools 37
    - Overview 37
    - Before you begin 37
    - Launch API Gateway Manager 37
    - Start Policy Studio 38
3. Configuring the Sample Policies 39
    - Overview 39
    - Enabling the Sample Services Interface 39
    - Configuring a Different Sample Services Interface 40
    - StockQuote Demo Service 40
    - Remote Host Settings 41
4. Conversion Sample Policy 43
    - Overview 43
    - REST to SOAP Policy 43
    - Running the Conversion Sample 44
5. Security Sample Policies 45
    - Overview 45
    - Signature Verification 45
    - Encryption and Decryption 46
6. Throttling Sample Policy 49
    - Overview 49
    - Throttling Policy 49
    - Running the Throttling Sample 49
7. Virtualized Service Sample Policy 51
    - Overview 51
    - Virtualized Service policies 51
    - Running the Virtualized Service Sample 55
8. Stress Testing with Send Request (SR) 56
    - Overview 56
    - Basic SR Examples 56
    - Advanced SR Examples 57
    - SR Arguments 57

9. Sending a Request with API Gateway Explorer 59
Overview 59
Creating a Request in API Gateway Explorer 59
Further Information 60
10. License Acknowledgments 61
Overview 61
Acknowledgments 61
2. Managing Policies .....
1. Configure policies from WSDL files 62
Overview 62
API Gateway as the web service initiator 62
API Gateway as the web service recipient 63
Import WSDL summary 63
Import a WSDL file 63
Configure a security policy 65
Configure recipient security settings 65
Configure initiator security settings 65
Configure recipient policy filters 66
Configure initiator policy filters 67
Edit the recipient or initiator WS-Policy 69
Configure a recipient WCF WS-Policy 70
Remove security tokens 71
Related information 72
2. Configure policies manually 74
Overview 74
Configuration 74
3. Configure global policies 76
Overview 76
Global policy roles 76
Select a global policy 77
Configure global policies in a policy shortcut chain 78
Configure global policies for a service 80
Show global policies 80
4. Configure policy assemblies 82
Overview 82
Configure a policy assembly 82
Apply a policy assembly 83
3. Managing Deployments .....
1. Manage API Gateway deployments 84
Overview 84
Connect to a server in Policy Studio 84
Edit a server configuration in Policy Studio 84
Manage deployments in API Gateway Manager 85
Compare and merge configurations in Policy Studio 85
Manage Admin users in API Gateway Manager 85
Configure policies in Policy Studio 85
2. Deploy API Gateway configuration 86
Overview 86
Create a package in Policy Studio 86
Configure package properties in Policy Studio 86
Deploy packages in Policy Studio 87
Deploy a factory configuration in Policy Studio 88
Deploy currently loaded configuration in Policy Studio 88
Push configuration to a group in Policy Studio 88
View deployment results in Policy Studio 88
Deploy on the command line 89
Deploy packages in API Gateway Manager 89

3. Compare and merge API Gateway configurations 90
Overview 90
Compare and merge configurations 90
Comparison results 90
Filter differences 92
Select differences for merging 92
4. Manage Admin users 93
Overview 93
Admin user privileges 93
Admin user roles 93
Add a new Admin user 94
Remove an Admin user 94
Reset an Admin user password 94
Manage Admin user roles 95
4. General Configuration .....
1. Manage connection details 96
Overview 96
Connect to a URL 96
Connect to a file 96
Unlock a server connection 97
2. Global configuration 98
Overview 98
API Gateway settings 98
Web service repository 98
API Gateway instances 99
Policies 99
Certificates and keys 99
API Gateway user store 100
System alerts 100
External connections 100
Caches 101
Black list and White list 101
WSDL and XML schema document bundles 101
Scripts 102
Stylesheets 102
References 102
3. Policy Studio preferences 103
Overview 103
Environmentalization setting 103
Management services 103
Policy color settings 103
Proxy settings 104
Runtime dependencies 104
SSL settings 104
Status bar setting 105
Topology screen settings 105
Trace level setting 105
Web and XML settings 105
WS-I settings 106
4. Policy Studio viewing options 108
Overview 108
Filter the tree 108
Configure viewing options 108
Configure the policy filter palette 108
5. Manage the web service repository 109
Overview 109
Manage web services and groups 109

Register a web service 109
Results of registering a web service 109
Export a web service 111
Update a web service 111
Change the operations exposed by a web service 112
Publish the WSDL 112
6. Oracle Security Service Module settings (10g) 114
Overview 114
Prerequisites 114
Settings 115
Name authority definition settings 116
Further information 116
7. Kerberos configuration 117
Overview 117
Kerberos configuration file—krb5.conf 117
Advanced settings 117
Native GSS library 118
8. Tivoli integration 119
Overview 119
Integration architecture 119
Prerequisites 120
Global Tivoli configuration 123
Tivoli authorization 124
Tivoli authentication 125
Tivoli attribute retrieval 127
9. Export API Gateway configuration 128
Overview 128
What is exported 128
Export configuration items 128
Export all API Gateway configuration 129
10. Import API Gateway configuration 130
Overview 130
Import configuration 130
View differences 130
What is imported 131
Import configuration from a previous version 131
5. API Gateway Instances .....
1. Configure API Gateway instances 133
Overview 133
Add remote hosts 133
Add HTTP services 133
Add SMTP services 133
Add file transfer services 133
Add policy execution scheduling 133
Configure JMS messaging system 133
Add Amazon SQS queue listener 134
Add FTP poller 134
Add directory scanner 134
Add POP client 134
Configure TIBCO 134
API Gateway settings 134
Cryptographic acceleration 134
2. Configure HTTP services 135
Overview 135
HTTP services groups 135
HTTP and HTTPS interfaces 137
HTTPS interfaces only 138

Management services 141
Change the management services port 142
3. Configure relative paths 143
Overview 143
Configure a relative path 143
Policies settings 143
Audit settings 144
HTTP method settings 145
Advanced settings 145
CORS settings 145
Nested relative paths 146
Static content providers 148
Static file providers 149
Servlet applications 149
Web service resolvers 150
4. Configure virtual hosts 153
Overview 153
Configure virtual hosts for HTTP services 153
Configure child resolvers 154
5. Configure SMTP services 155
Overview 155
Add SMTP service 155
Add SMTP interface 156
Configure policy handlers for SMTP commands 156
Add an HELO/EHLO policy handler 157
Add an AUTH policy handler 157
Add a MAIL policy handler 158
Add a RCPT policy handler 159
Add a DATA policy handler 159
SMTP authentication 160
SMTP Content-Transfer-Encoding 161
Deployment example 161
6. Configure a file transfer service 166
Overview 166
General settings 166
File upload settings 167
Secure services settings 168
Command settings 168
Access control settings 169
Message settings 169
Directory settings 170
Audit settings 170
Traffic monitor settings 171
7. Policy execution scheduling 172
Overview 172
Cron expressions 172
Add schedule 174
Add policy execution scheduler 174
8. Configure Amazon SQS queue listener 175
Overview 175
General settings 175
AWS settings 175
Poll settings 175
Response settings 176
Configure AWS client settings 176
Connection settings 176
Proxy settings 177
Advanced settings 177

Further information 177
9. Configure an FTP poller 178
Overview 178
General settings 178
Scan details 178
Connection type 179
FTP and FTPS connections 179
FTPS connections 179
SFTP connections 180
10. Configure directory scanner 181
Overview 181
General settings 181
Input settings 181
Processing settings 182
On completion settings 183
Traffic monitor settings 183
11. Packet sniffers 184
Overview 184
Configuration 184
12. Configure remote host settings 186
Overview 186
General settings 186
Address and load balancing settings 187
Advanced settings 188
Configure watchdogs 189
13. Configure WebSocket connections 190
WebSocket protocol overview 190
Configure a WebSocket connection 190
WebSocket configuration settings 191
Monitor a WebSocket connection 194
14. Configure HTTP watchdog 195
Overview 195
Configuration 195
15. Configure conditions for HTTP interfaces 196
Overview 196
Configure Requires Endpoint condition 196
Configure Requires Link condition 197
16. Configure a POP client 198
Overview 198
Configuration 198
17. TIBCO integration 199
Overview 199
TIBCO Rendezvous integration 199
18. Cryptographic acceleration 200
Overview 200
General configuration 200
Conversations for crypto engines 201
19. Cryptographic acceleration conversation: request-response 202
Conversations for crypto engines 202
20. TIBCO Rendezvous listener 203
Overview 203
Configuration 203
6. External Connections .....
1. External connections 204
Overview 204
Authentication repository profiles 204
Client credentials 204

Connection sets 205
Database connections 205
ICAP servers 205
JMS services 205
Kerberos connections 206
LDAP connections 206
Proxy servers 206
RADIUS clients 206
SiteMinder 206
SMTP servers 207
SOA security manager 207
Syslog servers 207
TIBCO 207
Tivoli 207
URL connection sets 207
XKMS connections 208
2. Authentication Repository 209
Overview 209
Axway PassPort Repositories 209
CA SiteMinder Repositories 209
Database Repositories 210
Entrust GetAccess Repositories 212
Local Repositories 213
LDAP Repositories 213
Oracle Access Manager Repositories 216
Oracle Entitlements Server 10g Repositories 217
RADIUS Repositories 217
RSA Access Manager Repositories 218
Tivoli Repositories 218
3. Axway PassPort Authentication Repository 220
Overview 220
Configuration 220
Axway PassPort Repository Registration 221
4. Configuring client credentials 224
Overview 224
Configuring API key client credential profiles 224
Adding API keys 224
Adding API key providers 225
Configuring HTTP Basic/Digest client credential profiles 225
Configuring Kerberos client credential profiles 225
5. Configure Sentinel servers 227
Sentinel server overview 227
General settings 227
Further information 227
6. Database Connection 228
Overview 228
Prerequisites 228
Configuring the Database Connection 228
Database Connection Pool Settings 229
Connection Validation 230
Test the Connection 230
7. Database Query 231
Overview 231
Configuration 231
8. Configuring ICAP Servers 233
Overview 233
General Settings 233
Server Settings 233

Security Settings 233
Advanced Settings 234
Further Information 234
9. JMS Services 235
Overview 235
Configuring a JMS Service 235
Configuring a JMS Session 236
Configuring a JMS Consumer 236
Configuring the JMS Wizard 237
10. Kerberos Clients 238
Overview 238
Ticket Granting Ticket Source 238
Kerberos Principal 239
Secret Key 239
Advanced Tab 240
11. Kerberos Principals 242
Overview 242
Configuration 242
12. Kerberos Services 244
Overview 244
Kerberos Endpoint Tab 244
Advanced Tab 245
13. Kerberos Keytab 246
Overview 246
Configuration 246
14. Configuring LDAP Directories 248
Overview 248
General Configuration 248
Authentication Configuration 248
Testing the LDAP Connection 249
Additional JNDI Properties 250
15. Proxy Servers 251
Overview 251
Configuration 251
16. RADIUS Clients 252
Overview 252
Configuration 252
17. SiteMinder/SOA Security Manager Connection 253
Overview 253
Prerequisites 253
SiteMinder and SOA Security Manager Connection Details 253
SOA Security Manager Connection Details Only 254
18. SMTP Servers 256
Overview 256
Configuration 256
19. TIBCO Rendezvous Daemon 257
Overview 257
Configuration 257
20. XKMS Certificate Validation Connection 259
Overview 259
Configuration 259
7. Resources and Libraries .....
1. Manage certificates and keys 260
Overview 260
View certificates and private keys 260
Configure an X.509 certificate 260
Create a certificate 260

Import certificates 261
Bind to a certificate at runtime 261
Configure a private key 261
Global options 262
Manage certificates and keystores 262
Export certificates to a keystore 262
Configure key pairs 262
Add a key pair 262
Manage OpenSSH keys 263
Configure PGP key pairs 263
Add a PGP key pair 263
Manage PGP keys 263
2. Manage API Gateway users 265
Overview 265
API Gateway users 265
Add API Gateway users 265
API Gateway user attributes 265
API Gateway user groups 266
Add API Gateway user groups 266
Update API Gateway users or groups 266
3. Manage WSDL and XML schema documents 267
Overview 267
Structure of the global cache 267
View cached WSDL or XML schema documents 268
Add XML schemas to the cache 269
Add WSDL documents to the cache 270
Update cached WSDL or XML schema documents 270
XML schema and WSDL document validation 271
XML schema and WSDL document limitations 271
Version and duplicate management 272
Validate messages against XML schemas 273
Test a WSDL for WS-I compliance 273
4. Global caches 275
Overview 275
Local caches 275
Distributed caches 276
Distributed cache settings 277
Example of caching response messages 278
5. Cross-Origin Resource Sharing 281
Overview 281
CORS request headers 281
CORS response headers 282
Adding a CORS Profile 282
General 282
Origins 282
Allowed Headers 282
Exposed Headers 283
Credentials Support 283
Preflight Results Cache 283
Configuring CORS for HTTP Services 283
Configuring CORS for Relative Paths 284
8. Amazon Web Services Filters .....
1. Send to Amazon SQS 286
Overview 286
General settings 286
AWS settings 286
Send message settings 286

Advanced settings 287
Further information 287
2. Upload to Amazon S3 288
Overview 288
General settings 288
AWS settings 288
S3 settings 288
Further information 289
9. Attribute Filters .....
1. Compare attribute 290
Overview 290
Configuration 290
2. Extract REST request attributes 291
Overview 291
Configuration 291
3. Extract WSS timestamp 292
Overview 292
Configuration 292
4. Extract WSS UsernameToken element 293
Overview 293
Configuration 293
5. Extract WSS header 294
Overview 294
Configuration 294
6. Get cookie 295
Overview 295
Configuration 295
Attribute storage 295
7. Insert SAML attribute assertion 297
Overview 297
General settings 297
Assertion Details 298
Assertion Location 298
Subject Confirmation Method 299
Advanced settings 301
8. LDAP attribute authorization 303
Overview 303
General configuration 303
Advanced configuration 305
9. Retrieve attribute from database 306
Overview 306
General settings 306
Database settings 306
Advanced settings 306
10. Retrieve attribute from directory server 309
Overview 309
General settings 309
Database settings 309
Advanced settings 310
11. Retrieve attribute from HTTP header 312
Overview 312
Configuration 312
12. Retrieve attributes from JSON message 313
Overview 313
Configuration 313
JSON Path examples 313
13. Retrieve attribute from message 317

Overview 317
Configuration 317
14. Retrieve attribute from SAML attribute assertion 318
Overview 318
Details 318
Trusted Issuers 319
Subject configuration 319
Lookup Attributes 319
15. Retrieve attribute from SAML PDP 321
Overview 321
Request configuration 321
Response configuration 323
16. Retrieve attribute from user store 324
Overview 324
General settings 324
Database settings 324
Advanced settings 324
10. Authentication Filters .....
1. Attribute Authentication 325
Overview 325
Configuration 325
2. Authenticate API Key 326
Overview 326
General Settings 326
API Key Settings 326
Advanced 327
3. CA SOA Security Manager Authentication 329
Overview 329
Prerequisites 329
Agent Configuration 329
Message Details Configuration 330
XmlToolkit.properties File 330
4. HTML Form-based Authentication 332
Overview 332
General Settings 332
Session Settings 332
5. HTTP Basic Authentication 334
Overview 334
Configuration 334
6. HTTP Digest Authentication 336
Overview 336
Configuration 336
7. HTTP Header Authentication 337
Overview 337
Configuration 337
8. IP Address 338
Overview 338
Configuration 338
Configuring Subnet Masks 338
9. SAML Authentication 341
Overview 341
General Settings 342
Details 342
Trusted Issuers 342
10. SAML PDP Authentication 343
Overview 343
Request Configuration 343

Response Configuration 345
11. Insert SAML Authentication Assertion 346
Overview 346
General Configuration 346
Assertion Details 347
Assertion Location 347
Subject Confirmation Method 348
Advanced 350
12. Insert Timestamp 352
Overview 352
Configuration 352
13. Insert WS-Security Username Token 353
Overview 353
General Configuration 353
Credential Details 353
Advanced 354
14. Kerberos Client Authentication 355
Overview 355
Kerberos Client 355
Kerberos Token Profile 356
15. Kerberos Service Authentication 357
Overview 357
Kerberos Service 357
Kerberos Standard 357
Message Level 357
Transport Level 358
Advanced SPNEGO 358
16. SSL Authentication 359
Overview 359
Configuration 359
17. Security Token Service Client 360
Overview 360
Example request 360
General settings 361
Request settings 361
Issue: POP Key 361
Issue: On Behalf Of Token 363
Issue: Token Scope and Lifetime 364
Validate: Target 365
Policies settings 365
Routing 365
Response settings 366
Advanced settings 366
18. WS-Security Username Authentication 368
Overview 368
General Configuration 368
Token Validation 369
Token Verification via Repository 369
11. Authorization Filters .....
1. RSA Access Manager Authorization 371
Overview 371
Prerequisites 371
General Details 371
Connection Details 371
Authorization Details 372
2. Attribute Authorization 373
Overview 373

Configuration 373
3. Axway PassPort Authorization 374
Overview 374
Configuration 374
4. CA SOA Security Manager Authorization 375
Overview 375
Prerequisites 375
Configuration 375
5. Certificate Attributes 377
Overview 377
Configuration 377
6. Entrust GetAccess Authorization 379
Overview 379
GetAccess WS-Trust STS 379
GetAccess SAML PDP 379
7. Insert SAML Authorization Assertion 381
Overview 381
General Configuration 381
Assertion Details 381
Assertion Location 382
Subject Confirmation Method 383
Advanced 385
8. Management Services RBAC filter 387
Overview 387
Configuration 387
9. SAML Authorization Assertion 388
Overview 388
General Settings 388
Details 388
Trusted Issuers 389
Optional Settings 389
10. SAML PDP Authorization 390
Overview 390
Request Configuration 390
Response 392
11. Tivoli Authorization 393
Overview 393
Adding a Tivoli Client 393
Adding Users and Web Services to Tivoli 393
Configuring Tivoli Authorization 394
Tivoli Authentication Refresh 395
12. Retrieve Attributes from Tivoli 396
Overview 396
Configuration 396
13. XACML Policy Enforcement Point 397
Overview 397
Example XACML request 398
General settings 398
XACML settings 398
Routing settings 401
Advanced settings 401
12. CA SiteMinder Filters .....
1. SiteMinder Certificate Authentication 403
Overview 403
Prerequisites 403
Configuration 403
2. SiteMinder Session Validation 405

Overview 405
Prerequisites 405
Configuration 405
3. SiteMinder Logout 407
Overview 407
Prerequisites 407
Configuration 407
4. SiteMinder Authorization 408
Overview 408
Prerequisites 408
Configuration 408
13. Certificate Filters .....
1. Static CRL certificate validation 410
Overview 410
Configuration 411
2. Dynamic CRL certificate validation 412
Overview 412
Configuration 412
3. CRL LDAP validation 413
Overview 413
Configuration 413
4. CRL responder 414
Overview 414
Configuration 414
5. Create thumbprint from certificate 415
Overview 415
Configuration 415
6. Certificate validity 416
Overview 416
Configuration 416
7. Find certificate 417
Overview 417
Configuration 417
8. Extract certificate attributes 418
Overview 418
Generated message attributes 418
Configuration 420
9. Certificate chain check 421
Overview 421
Configuration 421
10. OCSP client 422
Overview 422
General settings 422
Message settings 422
Routing settings 423
Advanced settings 423
Integration with Axway Validation Authority 423
11. Validate certificate store 425
Overview 425
Configuration 425
Deployment example 425
12. XKMS certificate validation 428
Overview 428
Configuration 428
14. Cache Filters .....
1. Cache Attribute 429
Overview 429

Configuration 429
2. Create Key 430
Overview 430
Configuration 430
3. Is Cached? 431
Overview 431
Configuration 431
4. Removed Cached Attribute 432
Overview 432
Configuration 432
15. Content Filters .....
1. Scan with ClamAV anti-virus 433
Overview 433
Configuration 433
2. Content type filtering 434
Overview 434
Allow or deny content types 434
Configure MIME/DIME types 434
3. Content validation 435
Overview 435
Manual XPath configuration 435
XPath wizard 436
4. HTTP header validation 437
Overview 437
Configure HTTP header regular expressions 437
Configure threatening content regular expressions 438
5. Send to ICAP 440
Overview 440
Configuration 440
Example policies 440
Further information 441
6. Scan with McAfee anti-virus 442
Overview 442
Prerequisites 442
Add McAfee binaries to API Gateway 442
Add McAfee binaries to Policy Studio 442
Configuration 442
Custom options 443
Message status 444
Load McAfee updates 444
7. Message size filtering 446
Overview 446
Configuration 446
8. Query string validation 447
Overview 447
Request query string 447
Configure query string attribute regular expressions 447
Configure threatening content regular expressions 449
9. Schema validation 450
Overview 450
General settings 450
Selecting the schema 450
Selecting which part of the message to match 450
Advanced settings 451
Reporting schema validation errors 453
10. JSON schema validation 455
Overview 455

Configuration 455
Generate a JSON schema using Jython 456
11. Scan with Sophos anti-virus 458
Overview 458
Prerequisites 458
General settings 458
Sophos configuration settings 458
12. Threatening content 460
Overview 460
Scanning settings 460
MIME type settings 460
13. Throttling 461
Overview 461
Rate limit settings 461
Advanced settings 462
Use multiple throttling filters 463
14. Validate selector expression 464
Overview 464
Configure selector-based regular expressions 464
Configure a Regular Expression 464
Threatening content regular expressions 465
15. Validate REST request 466
Overview 466
General settings 466
Adding REST request parameter restrictions 467
URI path templates 469
16. Validate timestamp 470
Overview 470
Configuration 470
17. Verify the WS-Policy security header layout 472
Overview 472
Configuration 472
18. XML complexity 473
Overview 473
Configuration 473
16. Conversion Filters .....
1. Add HTTP Header 474
Overview 474
Configuration 474
2. JSON Add Node 475
Overview 475
Configuration 475
Examples 476
3. Add XML Node 482
Overview 482
General Configuration 482
Configure where to Insert the New Nodes 482
Node Source 482
Configure New Node Details 483
Attribute Node Details 483
Examples 483
4. Contivo Transformation 485
Overview 485
Configuration 485
5. Multipart Bodypart Conversion 486
Overview 486
Configuration 486

6. Create Cookie 487
Overview 487
Configuration 487
7. Create REST Request 488
Overview 488
Configuration 488
8. Set HTTP Verb 489
Overview 489
Configuration 489
9. Insert MTOM Attachment 490
Overview 490
Configuration 491
10. JSON to XML 492
Overview 492
Configuration 492
Examples 492
11. Extract MTOM Attachment 495
Overview 495
Configuration 496
12. Load File 497
Overview 497
Input Settings 497
Processing Settings 497
On Completion Settings 497
13. Remove Attachments 499
Overview 499
Configuration 499
14. Remove HTTP Header 500
Overview 500
Configuration 500
15. JSON Remove Node 501
Overview 501
Configuration 501
Examples 501
16. Remove XML Node 504
Overview 504
Configuration 504
17. Restore Message 505
Overview 505
Configuration 505
18. Store Message 506
Overview 506
Configuration 506
19. Set Message 507
Overview 507
Configuration 507
Example of using selectors in the message body 507
20. XSLT Transformation 509
Overview 509
Stylesheet Location 509
Stylesheet Parameters 509
Advanced 509
21. XML to JSON 511
Overview 511
Configuration 511
17. Encryption Filters .....
1. Generate key 513

Overview 513
Configuration 513
2. PGP decrypt and verify 514
Overview 514
Configuration 514
3. PGP encrypt and sign 516
Overview 516
General settings 516
Encrypt and sign settings 516
Advanced settings 518
4. SMIME decryption 519
Overview 519
Configuration 519
5. SMIME encryption 520
Overview 520
General settings 520
Recipient settings 520
Advanced settings 520
6. XML decryption 521
Overview 521
Configuration 521
Auto-generation using the XML decryption wizard 521
7. XML decryption settings 522
Overview 522
XML encryption overview 522
Nodes to decrypt 524
Decryption key 525
Options 525
Auto-generation using the XML decryption wizard 526
8. XML encryption 527
Overview 527
Configuration 527
Auto-generation using the XML encryption settings wizard 527
9. XML encryption settings 528
Overview 528
XML encryption overview 528
Encryption key settings 530
Key info settings 531
Recipient settings 534
What to encrypt settings 536
Advanced settings 536
Auto-generation using the XML encryption settings wizard 537
10. XML encryption wizard 538
Overview 538
Configuration 538

18. Integrity Filters .....

1. XML Signature Generation 539
Overview 539
Signing Key 539
What to Sign 544
Where to Place Signature 549
Advanced 550
Additional 550
Algorithm Suite 552
Options 552
2. XML Signature Verification 555
Overview 555

Signature Verification 555
What Must Be Signed 555
Advanced 556
3. SMIME Sign 557
Overview 557
Configuration 557
4. SMIME Verify 558
Overview 558
Configuration 558
19. Fault Handler Filters .....
1. Generic Error 559
Overview 559
General Configuration 559
Generic Error Contents 559
2. JSON Error 561
Overview 561
General Configuration 561
JSON Error Contents 561
Customized JSON Errors 563
3. SOAP Fault 564
Overview 564
SOAP Fault Format 564
SOAP Fault Contents 564
Customized SOAP Faults 565
20. Monitoring Filters .....
1. Configure system alerts 568
Overview 568
Configure an alert destination 568
Syslog (local or remote) 568
Windows Event Log 569
Check Point FireWall-1 (OPSEC) 569
SNMP Network Management System 570
Email recipient 570
Amazon SNS 571
Twitter 572
Configure an alert filter 573
General settings 573
Notifications settings 574
Tracking settings 576
Default message settings 577
2. Set transaction log level and log message 578
Overview 578
Configuration 578
3. Log message payload 580
Overview 580
Configuration 580
4. Service level agreement 581
Overview 581
Response time requirements 581
HTTP status requirements 582
Communications failure requirements 583
Select alerting system 584
5. Set service context 585
Overview 585
General settings 585
6. Send event to Sentinel 586
Sentinel event filter overview 586

General settings 586
Settings tab 586
Tracking tab 587
Further information 587
7. Send cycle link event to Sentinel 588
Sentinel cycle link filter overview 588
General settings 588
Further information 589
21. Oracle Access Manager Filters .....
1. Oracle Access Manager Authorization 590
Overview 590
General Configuration 590
Request Configuration 590
OAM Access SDK Configuration 590
2. Oracle Access Manager Log in with Certificate 591
Overview 591
General Configuration 591
Resource Configuration 591
Session Configuration 591
OAM Access SDK Configuration 592
3. Logout from Oracle Access Manager SSO Session 593
Overview 593
Configuration 593
4. Oracle Access Manager SSO Token Validation 594
Overview 594
Configuration 594
22. Oracle Entitlements Server Filters .....
1. Oracle Entitlements Server 10g Authorization 595
Overview 595
General 595
Settings 595
Application Context 596
2. Get Roles from Oracle Entitlements Server 10g 597
Overview 597
General 597
Settings 597
Application Context 597
3. Oracle Entitlements Server 11g Authorization 598
Overview 598
Configuration 598
23. Resolver Filters .....
1. Relative Path Resolver 599
Overview 599
Configuration 599
2. SOAP Action Resolver 600
Overview 600
Configuration 600
3. Operation Name 601
Overview 601
Configuration 601
24. Routing Filters .....
1. Getting Started with Routing Configuration 602
Overview 602
Proxy or Endpoint Server 602
Service Virtualization 602
Choosing the Correct Routing Filters 602

Case 1: Proxy without Service Virtualization 603
Case 2: Proxy with Service Virtualization 604
Case 3: Endpoint without Service Virtualization 605
Case 4: Endpoint with Service Virtualization 606
Case 5: Simple Redirect 607
Case 6: Routing on to an HTTP Proxy 608
Summary 609
2. Call Internal Service 610
Overview 610
Configuration 610
3. Connection 611
Overview 611
General settings 611
SSL settings 611
Authentication settings 611
Additional settings 611
4. Connect to URL 612
Overview 612
General settings 612
Request settings 612
SSL settings 613
Trusted certificates 613
Client certificates 613
Authentication settings 613
Additional settings 614
Retry settings 614
Failure settings 614
Proxy settings 615
Redirect settings 616
Header settings 616
5. Dynamic Router 617
Overview 617
Configuration 617
6. Extract Path Parameters 618
Overview 618
Configuration 618
Required Input and Generated Output 619
Possible Outcomes 619
7. File Download 620
Overview 620
General Settings 620
File Details 620
Connection Type 620
FTP and FTPS Connections 621
FTPS Connections 621
SFTP Connections 621
8. File Upload 622
Overview 622
General Settings 622
File Details 622
Connection Type 623
FTP and FTPS Connections 623
FTPS Connections 623
SFTP Connections 624
9. HTTP Redirect 625
Overview 625
Configuration 625
10. HTTP Status Code 626

Overview 626
Configuration 626
11. Insert WS-Addressing 627
Overview 627
Configuration 627
12. Messaging System Filter 628
Overview 628
Request Settings 628
Response Settings 630
13. Read WS-Addressing 632
Overview 632
Configuration 632
14. Rewrite URL 633
Overview 633
Configuration 633
15. Save to File 634
Overview 634
Configuration 634
16. SMTP Routing 635
Overview 635
General Settings 635
Message Settings 635
17. Static Router 636
Overview 636
Configuration 636
18. TIBCO Rendezvous Routing 637
Overview 637
Configuration 637
19. Wait for Response Packets 638
Overview 638
Packet Sniffer Configuration 638
Sniffing Response Packets 639
25. Security Service Filters .....
1. DSS Signature Generation Service 640
Overview 640
Configuration 640
2. DSS Signature Verification 641
Overview 641
Configuration 641
3. Encrypt and Decrypt Web Services 642
Overview 642
Configuration 642
4. STS Web Service 643
Overview 643
Configuration 643
26. WS-Trust Filters .....
1. Consume WS-Trust Message 644
Overview 644
Consume WS-Trust Message Types 644
Message Consumption 644
Advanced 645
2. Create WS-Trust Message 646
Overview 646
Create WS-Trust Message Type 646
Message Creation 646
RST Creation 647
RSTR Creation 647

Advanced Settings 647
27. Utility Filters .....
1. Abort Filter 649
Overview 649
Configuration 649
2. Check Group Membership 650
Overview 650
Configuration 650
Possible Paths 650
3. Configuration Web Service 651
Overview 651
4. Copy/Modify Attributes 652
Overview 652
Configuration 652
5. Evaluate Selector 653
Overview 653
Configuration 653
6. Execute External Process 654
Overview 654
Configuration 654
7. False Filter 655
Overview 655
Configuration 655
8. HTTP Parser 656
Overview 656
Configuration 656
9. Insert BST 657
Overview 657
Configuration 657
10. Invoke Policy per Message Body 658
Overview 658
Configuration 658
11. Locate XML Nodes 659
Overview 659
Configuration 659
12. Pause Filter 661
Overview 661
Configuration 661
13. Policy Shortcut 662
Overview 662
Configuration 662
14. Policy Shortcut Chain 663
Overview 663
General Configuration 663
Add a Policy Shortcut 663
Edit a Policy Shortcut 664
15. Quote of the Day 665
Overview 665
Configuration 665
16. Reflect Message Filter 666
Overview 666
Configuration 666
17. Reflect Message And Attributes Filter 667
Overview 667
Configuration 667
18. Remove Attribute 668
Overview 668

Configuration 668
19. Set Response Status 669
Overview 669
Configuration 669
20. Set Attribute 670
Overview 670
Configuration 670
21. String Replace Filter 671
Overview 671
Configuration 671
22. Switch on Attribute Value 672
Overview 672
Configuration 672
Adding a Switch Case 672
23. Time Filter 674
Overview 674
General Configuration 674
Basic Time Options 674
Advanced Time Options 675
24. Trace Filter 676
Overview 676
Configuration 676
25. True Filter 677
Overview 677
Configuration 677
28. Web Services Filters .....
1. Web service filter 678
Overview 678
General settings 678
Routing settings 678
Validation settings 678
Configuring message interception points 679
WSDL settings 681
Monitoring settings 682
2. Return WSDL 683
Overview 683
Configuration 683
3. Set web service context 684
Overview 684
General settings 684
Service WSDL settings 684
Monitoring settings 684
29. Extending Filters .....
1. Advanced Filter View 685
Overview 685
Configuration 685
2. Selecting configuration values at runtime 686
Overview 686
Selector syntax 686
Accessing fields 686
Special selector keys 687
Resolving selectors 687
Example Selector Expressions 687
Message attribute 688
Environment variable 688
Key Property Store 688
Examples using reflection 688

Extracting Message Attributes 689
3. Key Property Stores 690
Overview 690
KPS Data Sources 690
Adding a KPS Collection 691
Editing a KPS Collection 691
Adding a KPS Table 692
Defining the KPS Table Structure 692
Further Information 693
4. Scripting Language Filter 694
Overview 694
Writing a Script 694
Adding your Script JARs to the Classpath 695
Configuring a Script Filter 695
Adding a Script to the Library 695
30. Configuring Common Settings .....
1. Certificate Chain Check 697
Overview 697
Configuration 697
2. Certificate validation 698
Overview 698
Configuration 698
3. Compressed Content Encoding 699
Overview 699
Encoding of HTTP Responses 699
Encoding of HTTP Requests 699
Delimiting the End of an HTTP Message 699
Configuring Content Encoding 700
Further Information 701
4. Configuring Connection Groups 702
Overview 702
Configuring a Connection Group 702
Configuring a Connection 702
5. Configuring Cron Expressions 703
Overview 703
Creating a Cron Expression Using the Time Tabs 703
Entering a Cron Expression 705
Testing the Cron Expression 706
Further Information 706
6. Signature Location 707
Overview 707
Configuration 707
7. Configuring a Transparent Proxy 710
Overview 710
Configuring Transparent Proxy Mode for Incoming Interfaces 710
Configuring Transparent Proxy Mode for Outgoing Calls 710
Configuration Example 710
8. Retrieving WSDL files from a UDDI registry 713
Overview 713
Introducing UDDI 713
UDDI definitions 713
Configuring a registry connection 715
WSDL search 715
Quick search 716
Name search 716
Advanced search 717
Advanced options 718

Publish 720
9. Connecting to a UDDI registry 722
Overview 722
Configuring a registry connection 722
Securing a connection to a UDDI registry 723
10. Publishing WSDL files to a UDDI registry 725
Overview 725
Finding WSDL files 725
Publishing WSDL files 725
Step 1: Enter virtualized service address and WSDL URL for publishing in UDDI registry 725
Step 2: View WSDL to UDDI mapping result 726
Step 3: Select a registry for publishing 727
Step 4: Select a duplicate publishing approach 727
Step 5: Create or search for business 728
Step 6: Publish WSDL 728
11. LDAP User Search 730
Configure Directory Search 730
12. Configuring URL Groups 731
Overview 731
Configuration 731
13. What To Sign 732
Overview 732
ID Configuration 732
Node Locations 734
XPath Configuration 734
XPath Predicates 735
Message Attribute 735
14. Configuring XPath Expressions 736
Overview 736
Manual Configuration 736
XPath Wizard 737
31. Reference .....
Message Attribute Reference 739
Message Filter Reference 771
WS-Policy Reference 804

API Gateway management with Policy Studio

Overview

This topic explains some of the main components and concepts used in Oracle API Gateway management, and shows examples of how they are displayed in the API Gateway management tools such as Policy Studio and API Gateway Explorer. For example, these include concepts such as filters, policies, message attributes, and listeners. For details on the core API Gateway features and architecture, see the *API Gateway Concepts Guide*. For example, this includes concepts such as API Gateway instances, groups, Node Manager, Admin Node Manager, and so on.

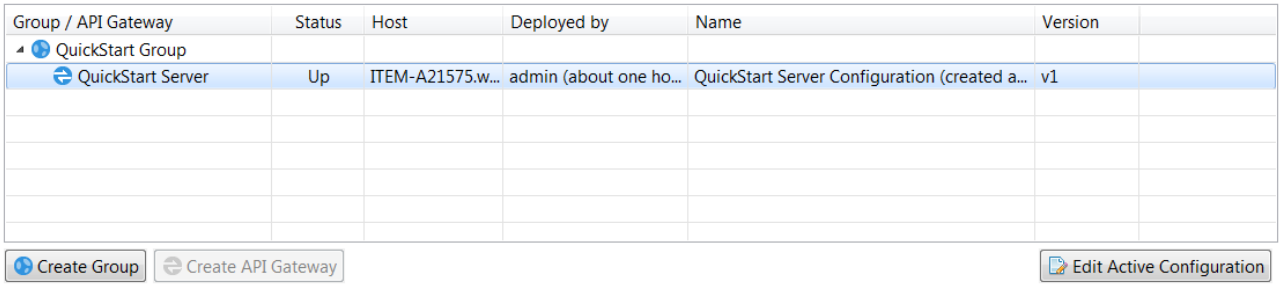
API Gateway instances and groups

You can use Policy Studio to configure and deploy API Gateway instances and groups. An API Gateway instance is an API Gateway capable of running on a host. You can configure various features at the instance level (for example, HTTP(S) interfaces, file transfer services, JMS services, remote hosts, and so on).

An API Gateway group is a collection of one or more API Gateway instances that share the same configuration. For more details, see the *API Gateway Concepts Guide*.

Topology view

When you connect to an Admin Node Manager session in Policy Studio, the API Gateway groups and API Gateway instances are displayed in the **Topology** view.



You can quickly identify any problems with the topology by looking at the icons in the **Group / API Gateway** column, and the server status in the **Status** column. The status of a group or API Gateway is also displayed in the status bar at the bottom.

API Gateway groups

The possible status of a API Gateway group in the **Topology** view is:

- *Consistent*—a group displayed with a blue icon is consistent. API Gateways in the group have the same configuration. The status bar displays the group name (for example, `QuickStart Group`).
- *Inconsistent*—a group displayed with a grey icon is inconsistent. API Gateways in the group do not have the same configuration. The status bar displays `<GroupName> configuration is inconsistent`.
- *Unlocked*—a group displayed without a padlock icon is unlocked. API Gateways in the group are available for editing. When a group is unlocked and consistent, the status bar displays the group name (for example, `QuickStart Group`).
- *Locked*—a group displayed with a padlock icon is locked. API Gateways in the group are locked by a specific user, and are not available for editing. The status bar displays `<GroupName> [locked by <UserName>]`. To lock a group, right-click the group name, and select **Lock Group Access**. To unlock a group, right-click, and select **Unlock Group Access**. Only admin users can unlock groups locked by other users.

Note

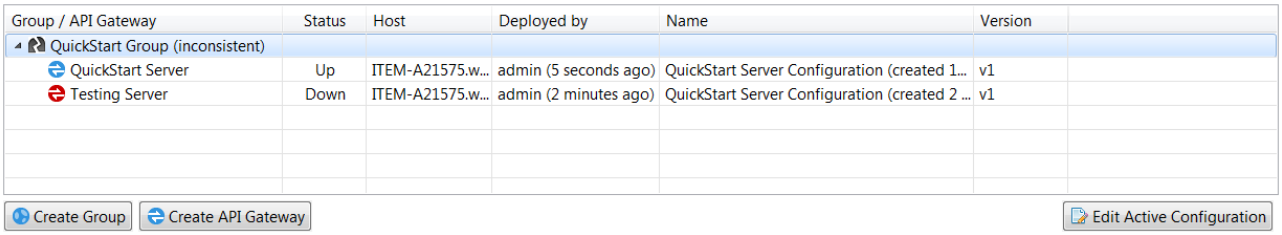
To get an inconsistent configuration back to a consistent state, you can push the configuration from one API Gateway instance to the other instances in the group. For more information, see the section called "Push configuration to a group in Policy Studio".

API Gateway instances

The possible status of an API Gateway instance in the **Topology** view is:

- *Running*—an API Gateway displayed with a blue icon is running. The status column shows the status as **Up**. The status bar displays `<ServerName> is running`.
- *Not running*—an API Gateway displayed with a red icon is not running. The status column shows the status as **Down**. The status bar displays `<ServerName> is not running`.

The following example shows a topology where the group is inconsistent, and an API Gateway is not running:



Tip

You can customize the Topology view in the Policy Studio preferences. For more details, see *Policy Studio preferences*.

Filters

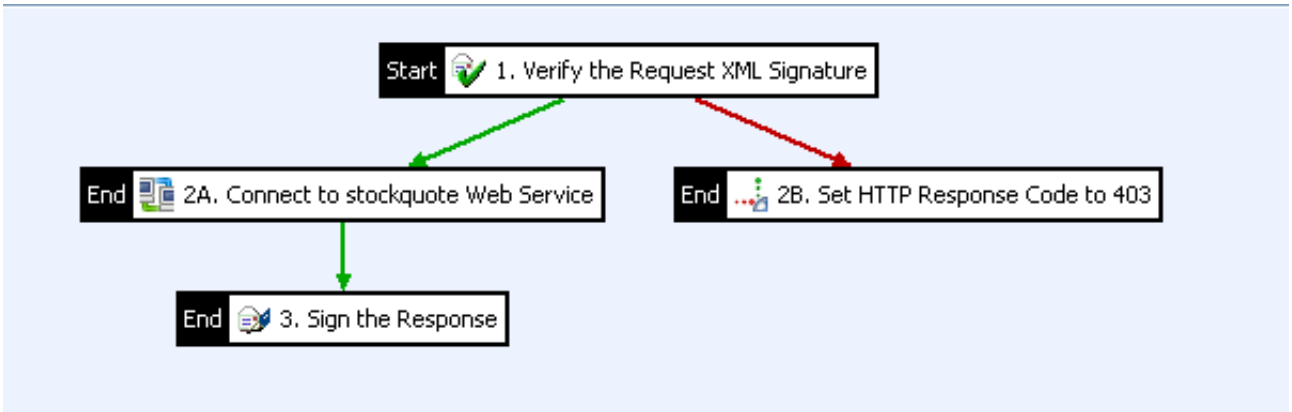
A filter is an executable rule that performs a specific type of processing on a message. For example, the **Message Size** filter rejects messages that are greater or less than a specified size. There are many categories of message filters available with the API Gateway, including authentication, authorization, 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. The next section shows some example filters.

Policies

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 shows an example policy that performs XML signature verification:

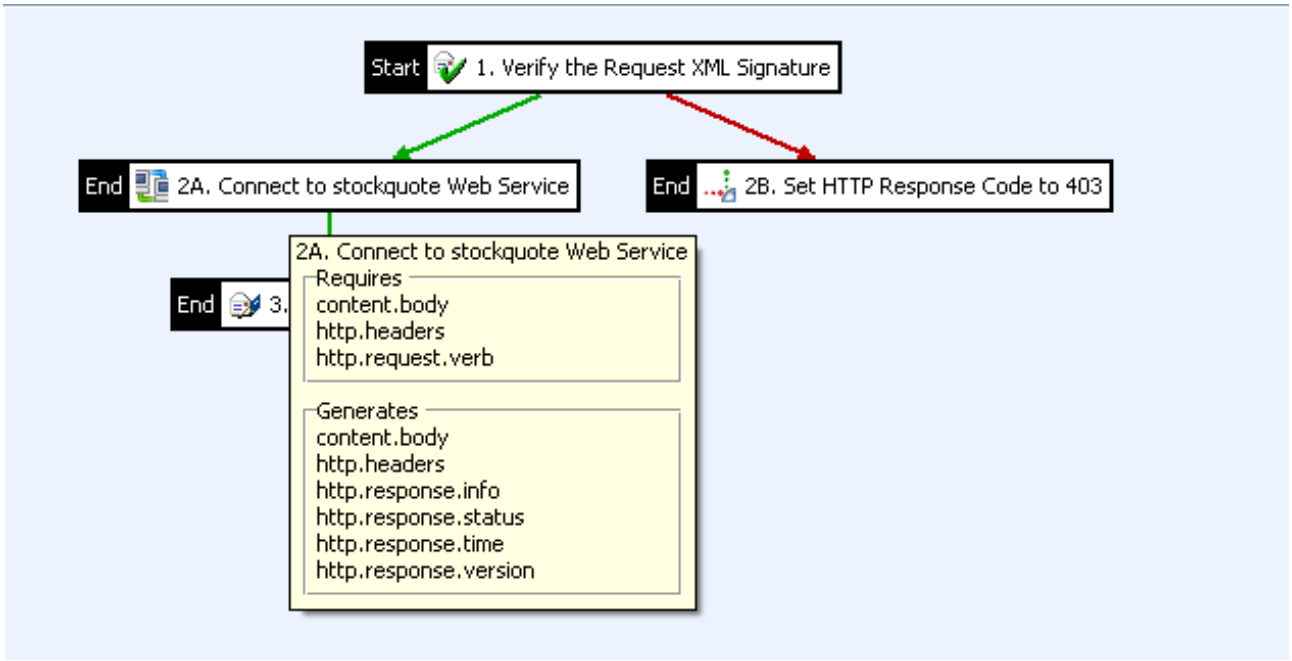


A policy must have a **Start** filter (in this case, **Verify the Request XML Signature**). Filters labeled **End** stop the execution of the policy (for example, the filter execution fails). A filter labeled **Start/End** indicates that the policy execution starts there, and that the policy stops executing 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 runtime.

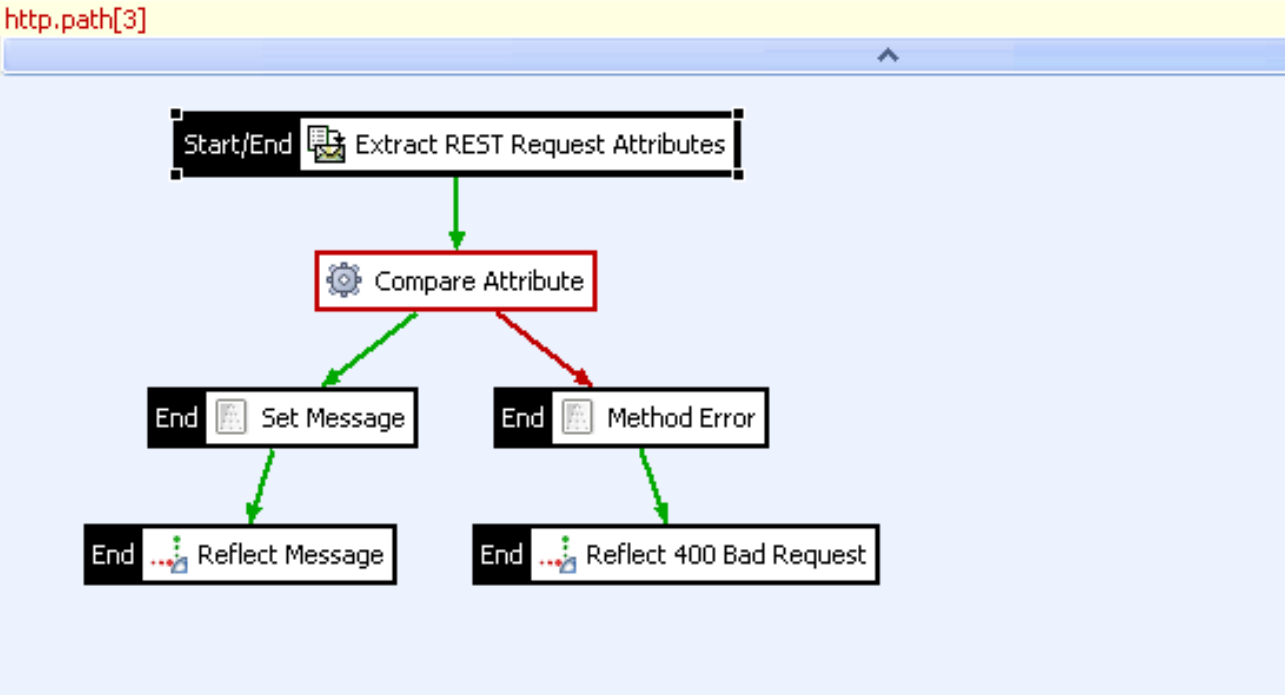
The message attribute *white board* refers to the list of attributes that are available to a particular filter at runtime of the API Gateway during the processing of requests and responses. The following example shows the attributes displayed when hovering over a **Connect to URL** filter at design time in Policy Studio:



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 shows a required attribute:



Selectors

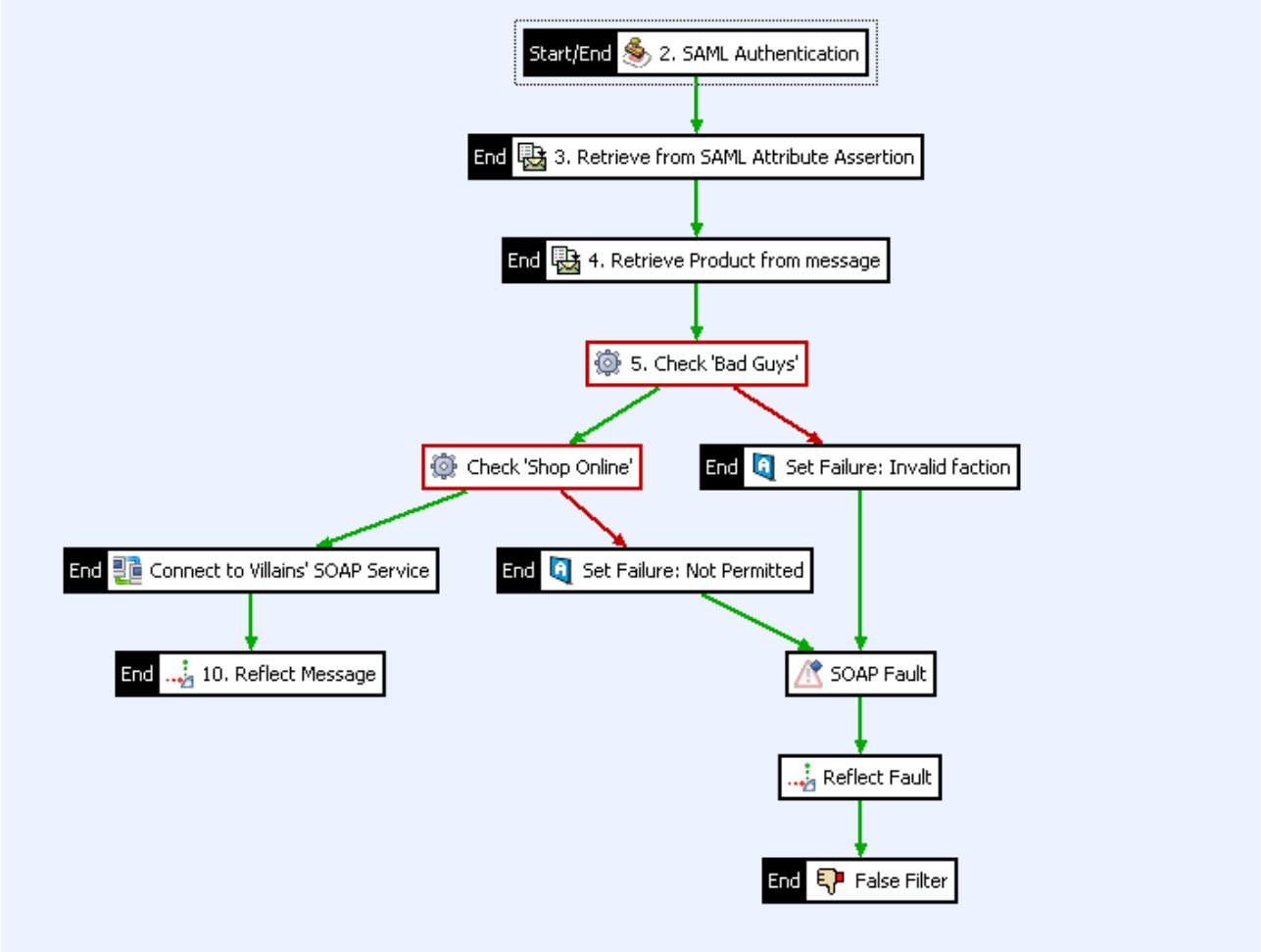
A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata (for example, in message attributes, a Key Property Store (KPS), or environment variables). For example, the following selector returns the value of a message attribute:

```
${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 and errors

When a transaction fails, you can use a fault to return error information to the client application. The API Gateway provides the **SOAP Fault**, **JSON Error**, and **Generic Error** filters. By default, the API Gateway returns a very basic fault to the client when a message filter fails. You can add a specific fault filter to a policy to return more detailed error information to the client. For example, the following screen shows an authentication policy that includes a **SOAP Fault**:



Policy shortcuts

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.

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 containers

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 contexts

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.

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 Rendezvous 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.

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).

Start the API Gateway tools

Overview

This topic describes the prerequisites and preliminary steps. It explains how to start the API Gateway Manager administrator tool and the Policy Studio developer tool.

Before you begin

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

Install the API Gateway and Policy Studio
If you have not already done so, see the *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 and an API Gateway instance are running.

Launch API Gateway Manager

To access the web-based API Gateway Manager administration 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**: System health, traffic summary, and topology (domain, hosts, API Gateways, and groups).
- **Traffic**: Message log and performance statistics on the traffic processed by the API Gateway. For example, all HTTP, HTTPS, JMS, File Transfer, and Directory messages processed by the API Gateway.
- **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.
- **Logs**: API Gateway trace log, transaction log, and access log files.
- **Events**: API Gateway transaction log points, alerts, and SLA alerts.

- **Settings**: Enables you to configure dynamic API Gateway logging, user roles, and credentials.

Start Policy Studio

To start the Policy Studio tool used to create and manage policies, 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 defaults.
4. The **API Gateway** instance is displayed in the **Topology** view.
5. In the **Topology** view, double-click the **API Gateway** instance 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 develop 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 *Manage API Gateway deployments*.

Configuring the Sample Policies

Overview

This topic introduces and explains how to set up 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 to get familiar with the main concepts and basic steps:

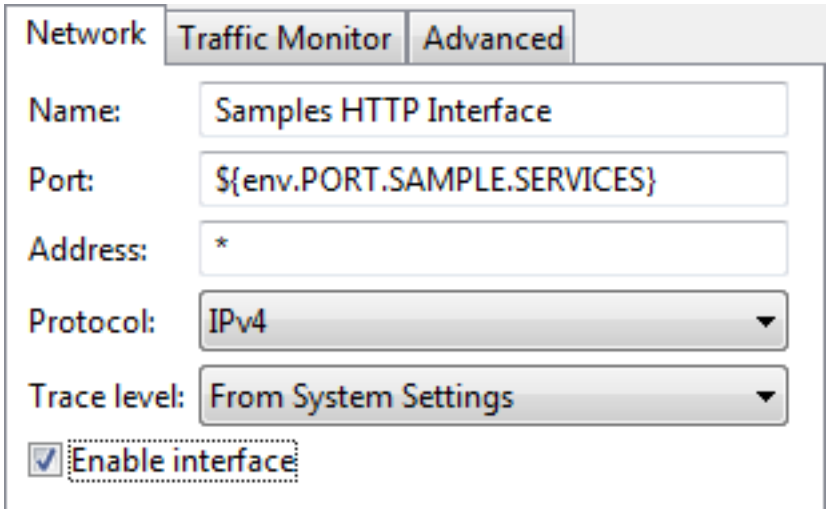
- *API Gateway Concepts Guide*
- *API Gateway management with Policy Studio*
- *Start the API Gateway tools*

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** > **API Gateway** > **Sample Services** > ***:${env.PORT_SAMPLE_SERVICES}**.
2. Right-click, and select **Edit** to display the **Configure HTTP Interface** dialog.
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 for API Gateway instances, see the *API Gateway Deployment and Promotion Guide*.

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.

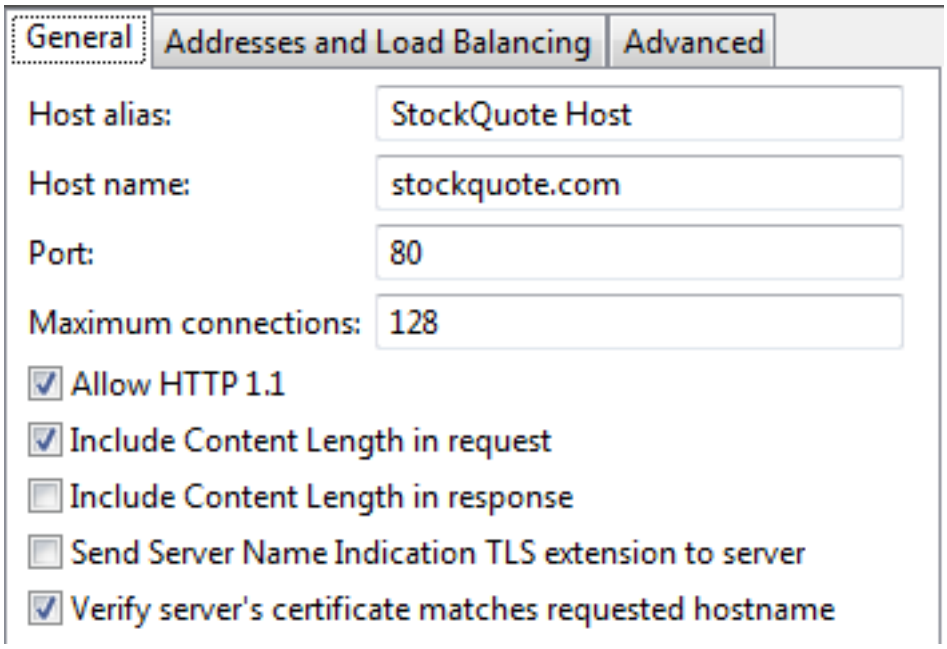
Sample Service URL
A **Connect to URL** filter with the following URL is used to invoke the `StockQuote` service from each of the sample policies:

```
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

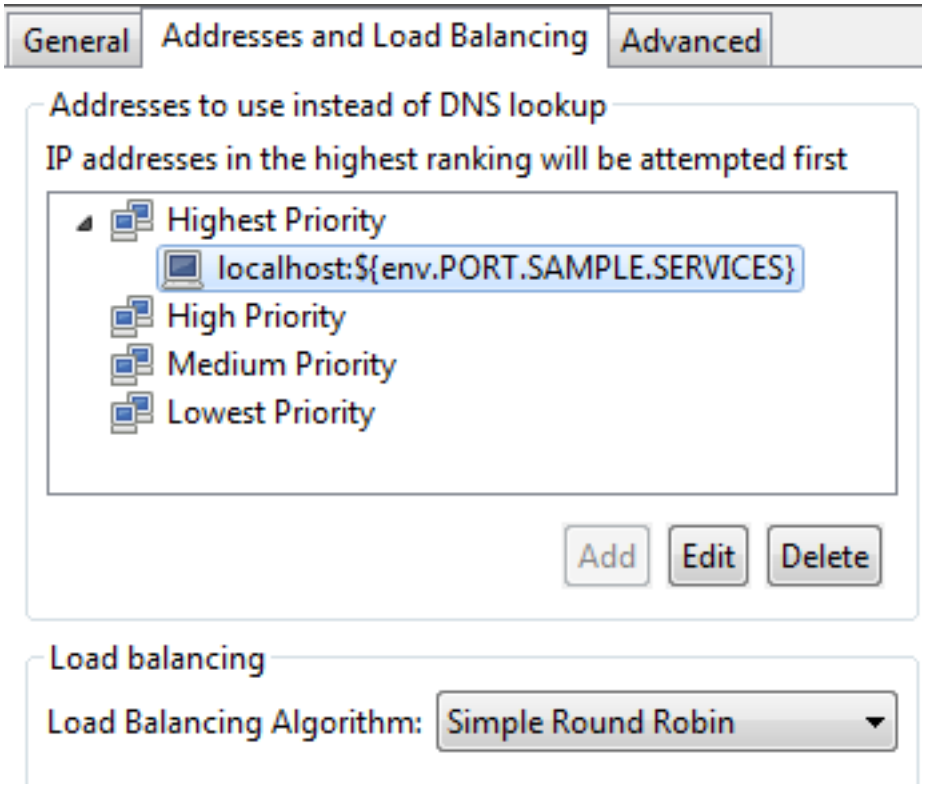
In the Policy Studio, the Remote Host configuration is displayed under the API Gateway instance name (`API Gateway`) in the navigation tree, and is named `stockquote.com:80`. To view its settings, right-click, and select **Edit** to view the **Remote Host Settings** dialog:



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 and Load Balancing** 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 to use instead of DNS lookup**list, and the API Gateway would load balance the requests across servers in the same group using the specified algorithm.



For more details on these settings, see *Configure 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:

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

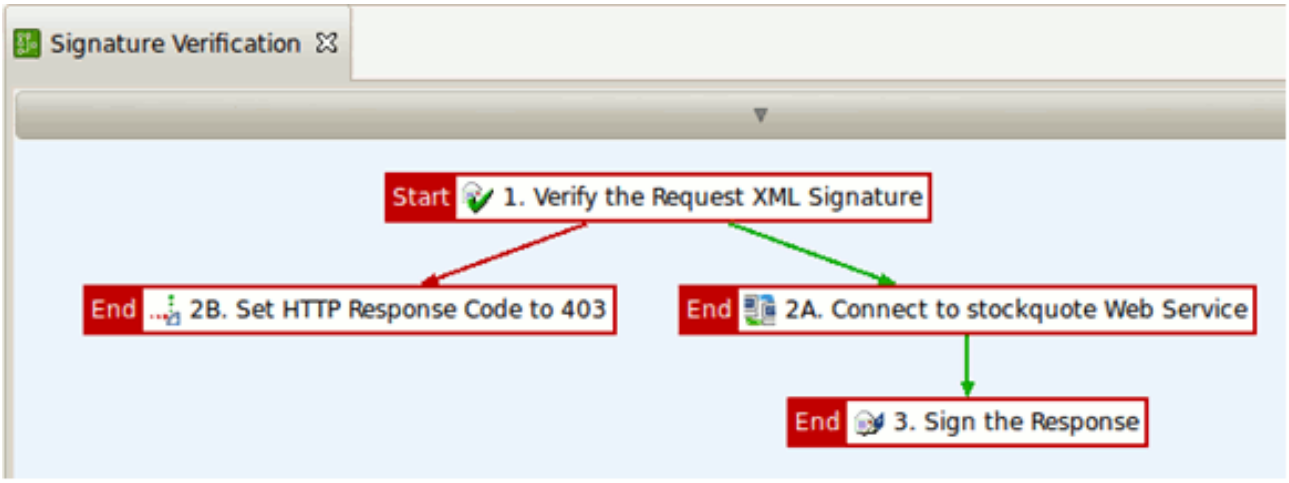
Overview

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:



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
   ```

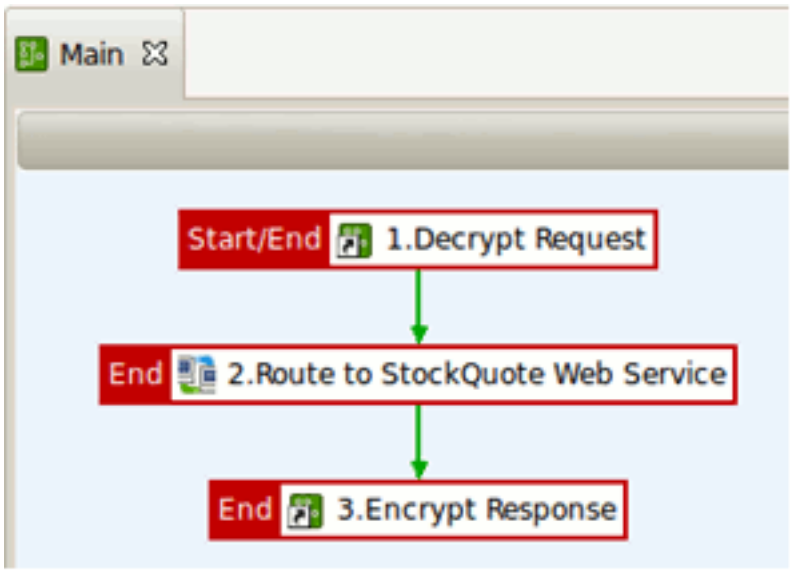
5. Click the Send Request button.

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.

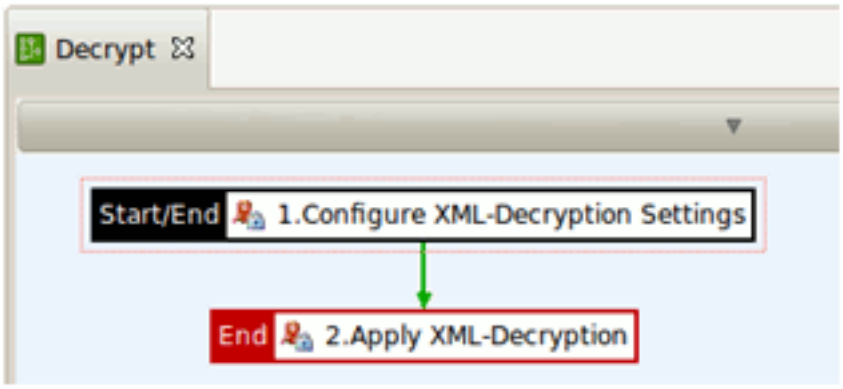
Main Policy
The **Main** policy is as follows:



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.

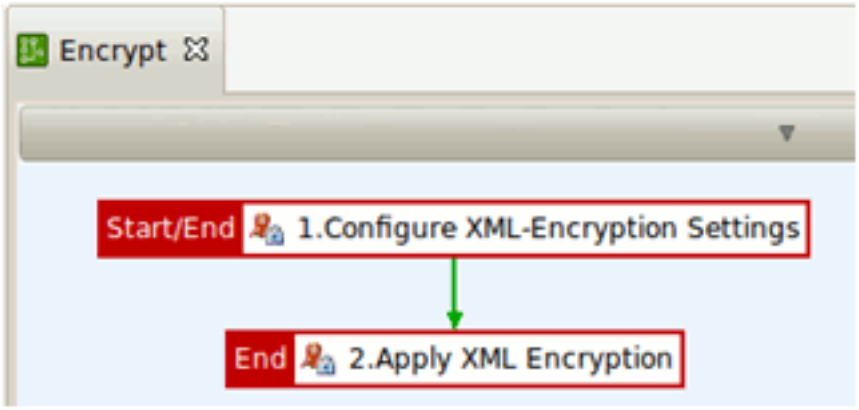
Decrypt Policy
The **Decrypt** policy is as follows:



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

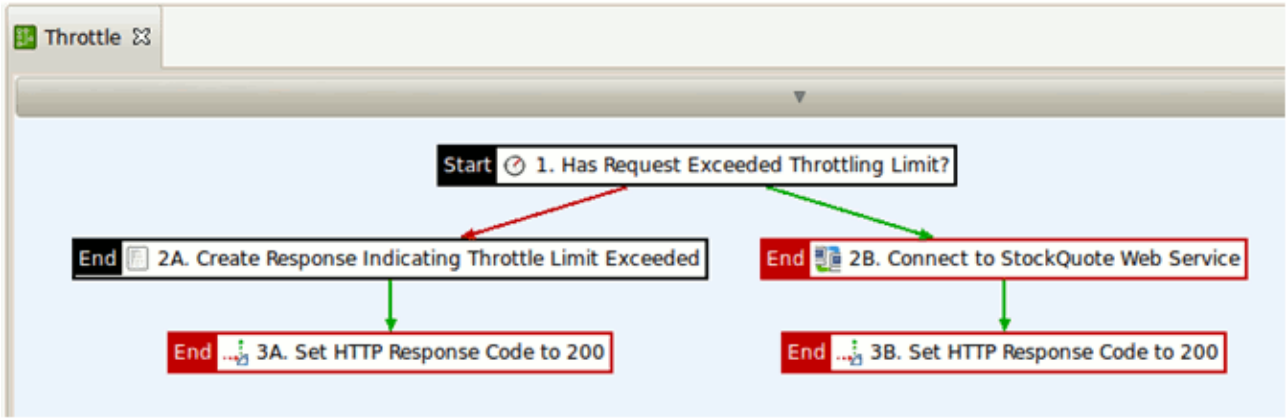
Overview

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:



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.

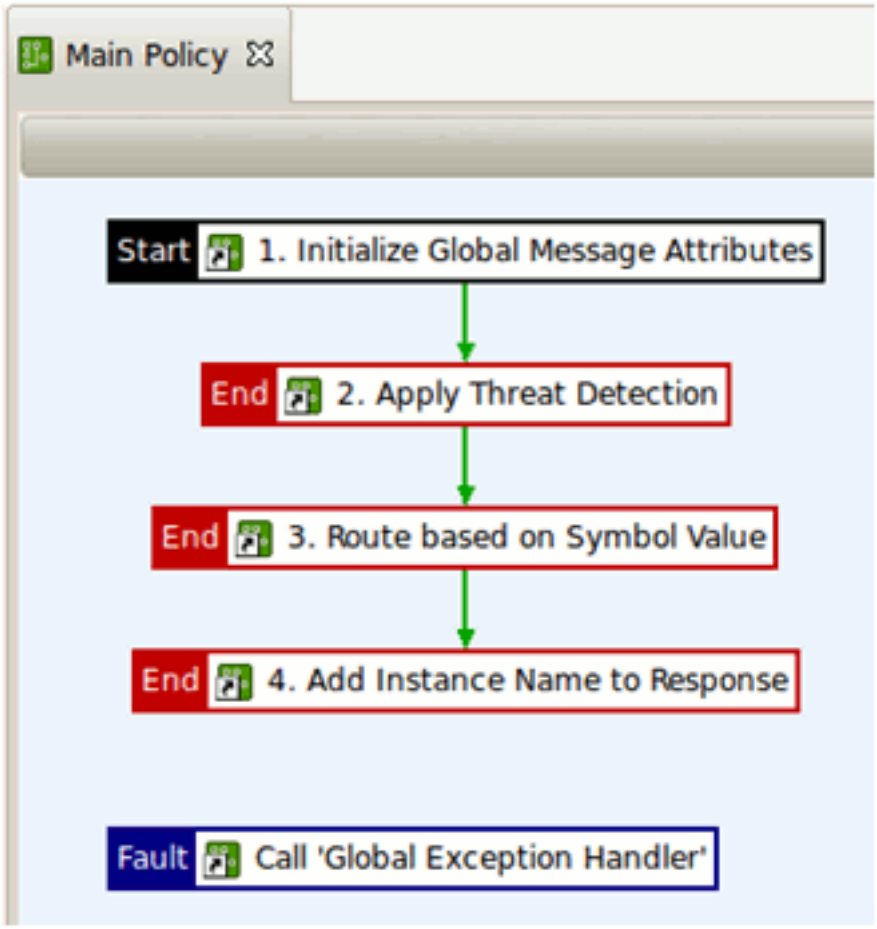
This topic describes the policies displayed in the **Sample Policies** > **Web Services** > **Virtualized StockQuote Service** policy container in the Policy Studio, and explains how to run this sample.

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

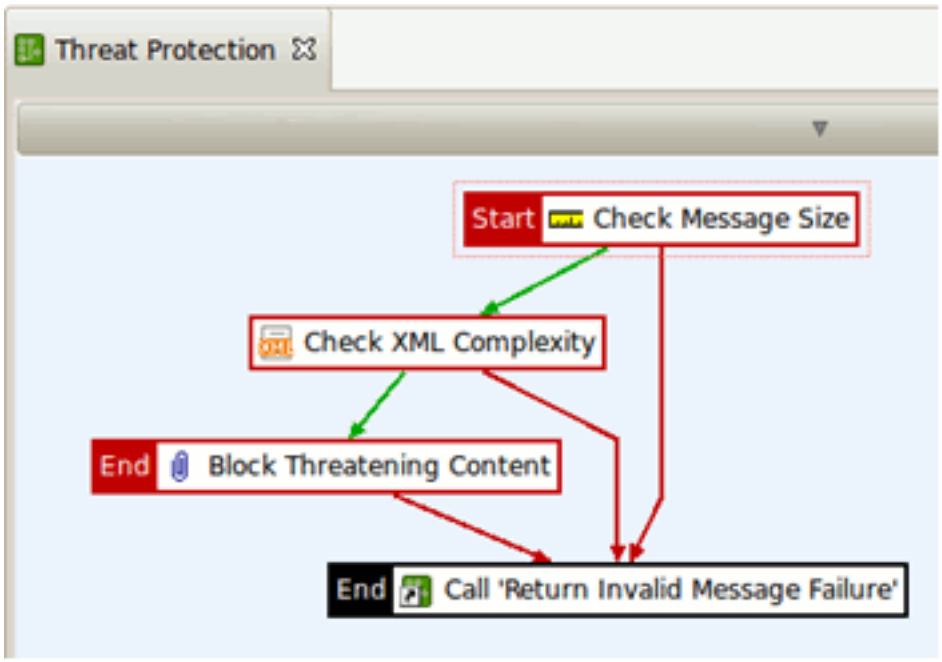
Virtualized Service Main Policy
The **Main Policy** is as follows:



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:



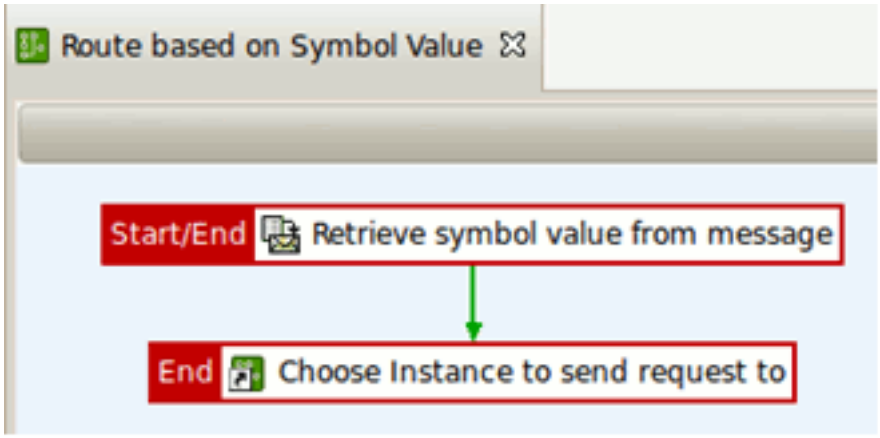
The **Threat Protection** policy performs the following tasks:

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`

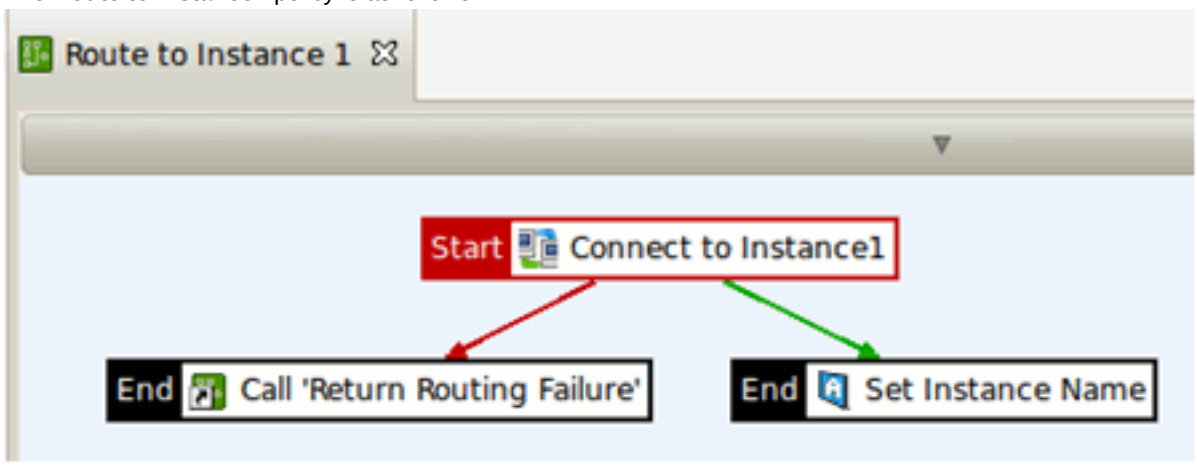
The **Route Based on Symbol Value** policy is as follows:



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:

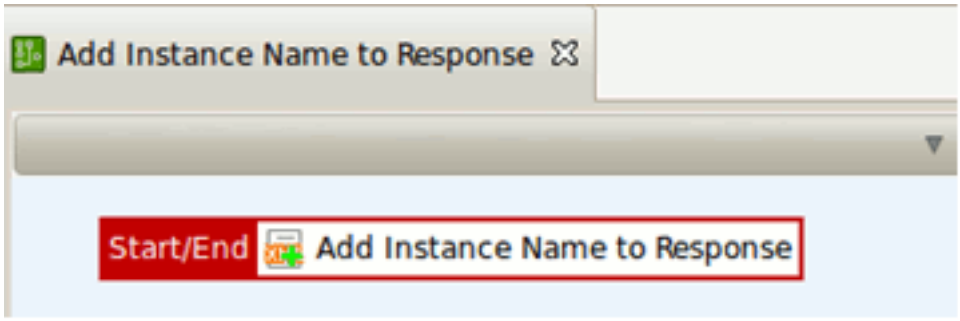


The **Route to Instance1** policy (called from the Switch filter) performs the following tasks:

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://oracle.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
`-f [content-filename]`	File to send as the request
`-h [host]`	Name of destination host
`-i [filename]`	Destination of statistics data

Argument	Description
`-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

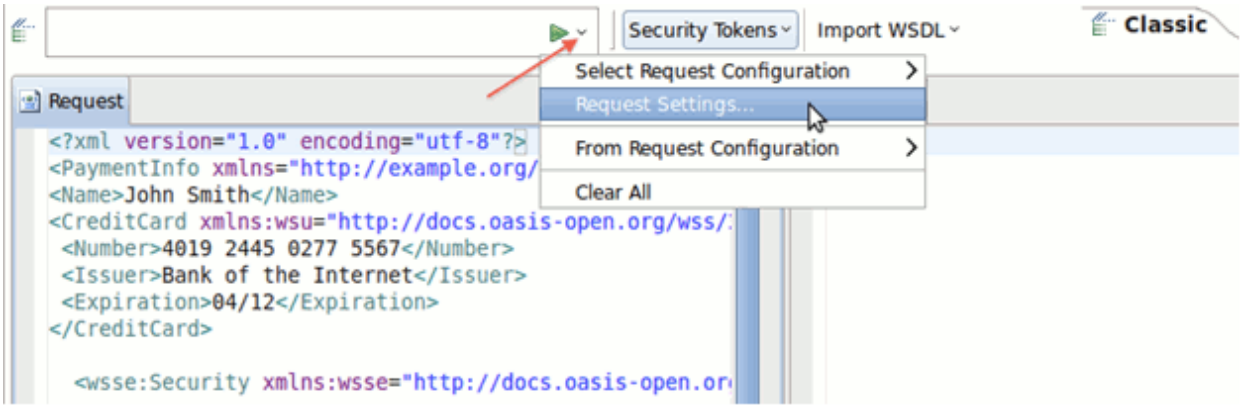
Overview

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.

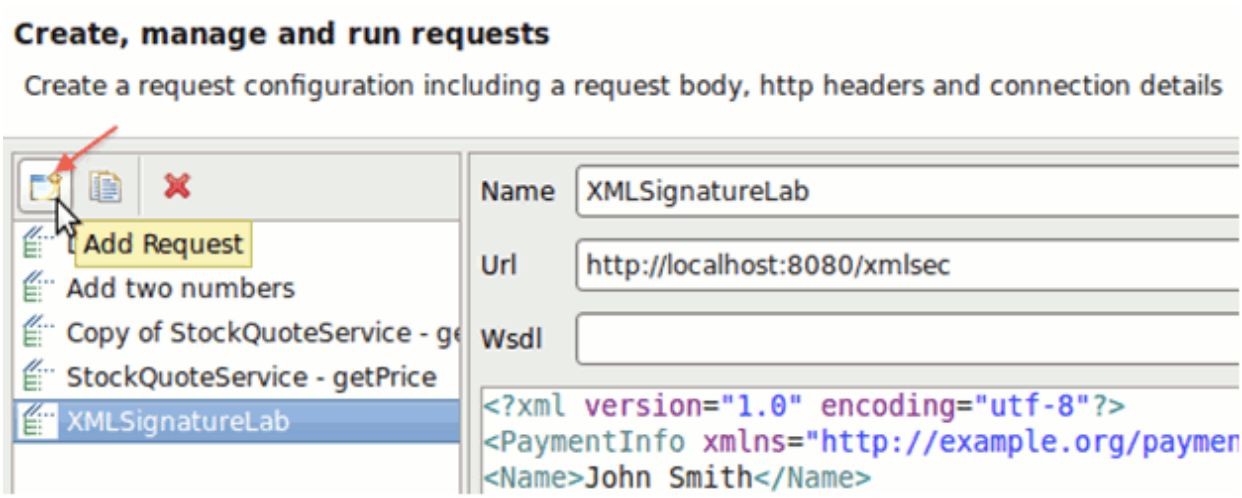
Creating a Request in API Gateway Explorer

To create a request, perform the following steps:

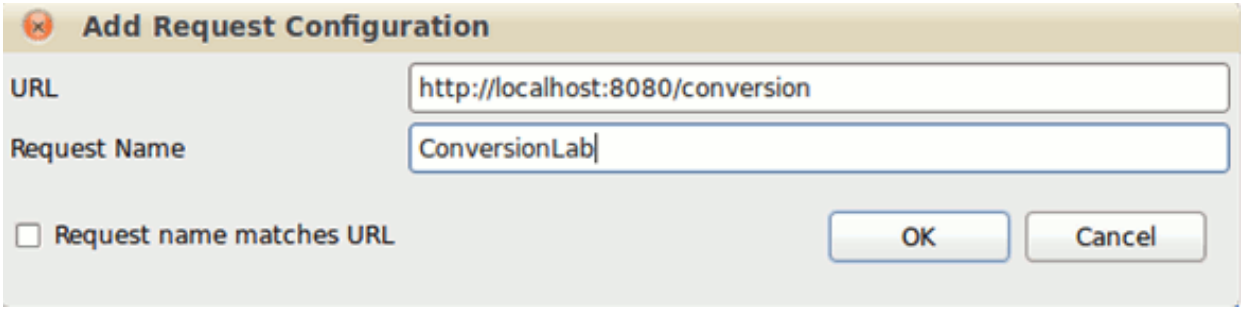
1. Click the down arrow button beside the green triangular Send Request button in the toolbar, and select **Request Settings**:



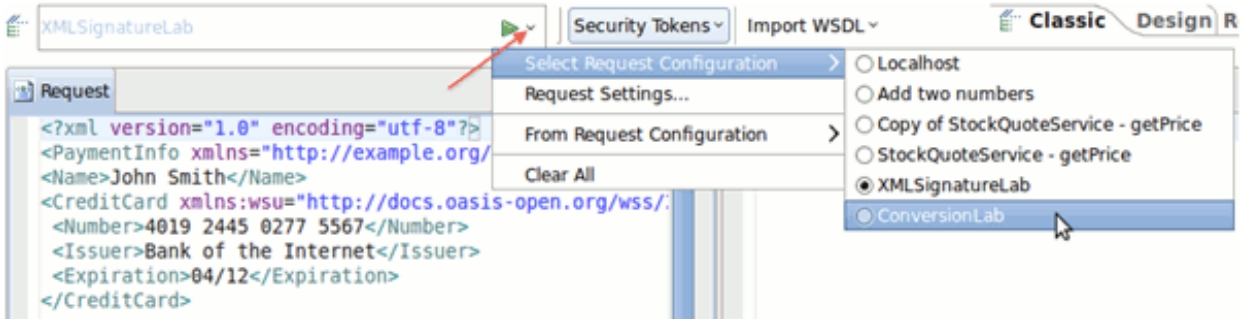
2. In the **Request Settings** dialog, click the **Add Request** button on the left in the toolbar:



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.



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



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.

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.

iconmonstr:
This product includes graphic icons developed by iconmonstr [http://iconmonstr.com/].

Configure policies from WSDL files

Overview

When you import a WSDL file describing a web service into the web service repository, API Gateway exposes a *virtualized* version of the web service. This virtualized service is exposed on a host and port of the machine running API Gateway (for more information on how this is achieved see the section called "Publish the WSDL"). In this way, a client can retrieve the WSDL for the virtualized web service from API Gateway without knowing its real location.

When you import a WSDL file, Policy Studio automatically generates policies that can be used to talk to the back-end web service. For example, a **Service Handler** is created to resolve and validate requests to the web service and responses from the web service. The service handler is automatically configured based on the operation definitions in the WSDL.

A WSDL file can include WS-Policy assertions that define the security policies or security requirements that a client must adhere to in order to communicate with the corresponding service. For example, a web service might require the client to sign sensitive parts of the message and include a WS-Security `UsernameToken` to authenticate to the web service.

If the imported WSDL file contains WS-Policy assertions, API Gateway is responsible for making sure the requests it sends to the web service adhere to the security constraints specified in the policy. In this case API Gateway is considered as the *initiator* of the web service.

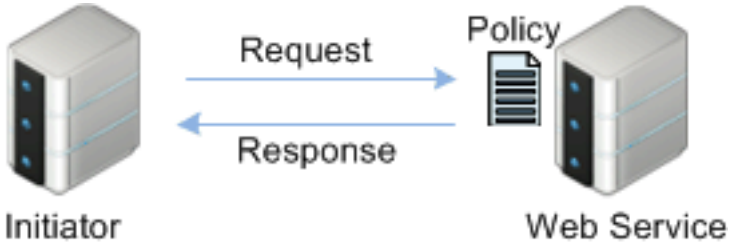
During the WSDL import process, you can select which operations to expose to clients and what path to expose the service on, and you can also specify the WS-Policy assertions that API Gateway enforces on the messages it receives from clients. In this case, API Gateway is considered to be the *recipient* to a consumer (that is, client) of the virtualized web service.

You can use the **Import WSDL** wizard to configure API Gateway as a web service initiator, as a recipient to a consumer of the virtualized web service, or both. The steps involved in the wizard, and the configuration windows displayed, differ according to whether WS-Policy assertions are being enforced between API Gateway and the web service (initiator case) or between a client and API Gateway (recipient case). For more information on the **Import WSDL** wizard, see the section called "Import a WSDL file"

The **Import WSDL** wizard enables you to configure complex policies to *secure and virtualize* web services with only a few clicks and minimal intervention.

API Gateway as the web service initiator

The following figure demonstrates the case where API Gateway is the initiator of the web service:

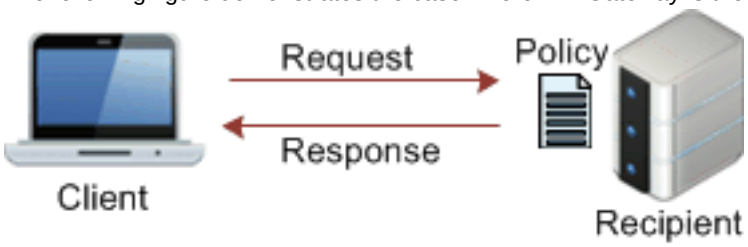


When the imported WSDL describing the web service includes WS-Policy assertions, API Gateway is responsible for making sure that the requests it sends to the web service adhere to the security constraints specified in the policy.

Policy Studio automatically generates the policies required to talk to the web service, and to enforce the security requirements specified by the policy assertions in the WSDL. These policies contain filters, most of which are configured automatically. However, a small number of filters require you to manually configure specific fields. For example, when signing or encrypting a message, you must specify the signing or encrypting key.

API Gateway as the web service recipient

The following figure demonstrates the case where API Gateway is the recipient of the web service:



When importing the WSDL, the **Import WSDL** wizard enables you to select the web service operations to expose to the client, set the path on which to expose the service, and optionally to specify the security policies that API Gateway enforces on the messages it receives from the client.

Policy Studio automatically generates the policies that API Gateway uses to talk to the client, and to enforce the security requirements you specified during WSDL import. These policies contain filters, most of which are configured automatically. However, a small number of filters require you to manually configure specific fields. For example, when configuring the duration of a WS-Security timestamp, you might need to specify a longer or shorter duration.

Import WSDL summary

The steps involved in the **Import WSDL** wizard are summarized as follows:

1. Load the WSDL from a file, URL, or UDDI.
2. Retrieve and validate the WSDL.
3. Select the operations to expose.
4. Configure the WS-Policy settings. If the imported WSDL file includes WS-Policy settings, policies are generated to secure the connection between API Gateway and the web service. You can also secure the virtualized service by configuring WS-Policy settings between the client and API Gateway at this step.
5. Select the path to expose the services on.
6. If you selected to secure the virtualized service using WS-Policy settings, configure the policies that API Gateway will enforce for messages it receives. You can select gateway and message level policies at this step.
7. Configure the recipient security settings. At this step you can manually configure the policies that have been automatically generated by Policy Studio to enforce the WS-Policy settings between the client and API Gateway.
8. Configure initiator security settings. At this step you can manually configure the policies that have been automatically generated by Policy Studio to enforce the WS-Policy settings between API Gateway and the back-end web service.
9. Review the summary information for the virtualized service. At this step you can also override the default validation and routing policies with custom policies, and decide whether or not to publish the WSDL to clients.

Import a WSDL file

To import a WSDL file into the web service repository, complete the following steps:

1. In the Policy Studio tree view, expand the **Business Services > Web Service Repository** node.
2. Right-click the **Web Services** node, and select **Register Web Service** to launch the **Import WSDL** wizard.
3. In the **Load WSDL** window, select the WSDL location from one of the following options:
   - **WSDL File**:
     Click the **Browse** button to select a WSDL file from the file system.
   - **WSDL URL**:
     Enter a URL where the WSDL can be loaded from. This option supports HTTP basic authentication. Under **Authentication** select the **Use HTTP Basic** option and enter a user name and password.

- **WSDL from UDDI**:
  Click the **Browse UDDI** button to select a WSDL file from a Universal Description, Discovery, and Integration (UDDI) registry. For more information on how to retrieve a WSDL file from a UDDI registry, see the *Retrieving WSDL files from a UDDI registry* topic.

Click **Next**.

4. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL. Click **Next** to continue.

Note

If the WSDL fails validation, an error is displayed. For more information, see the section called "XML schema and WSDL document validation" and the section called "XML schema and WSDL document limitations".

5. In the **WSDL Operations** window, select the operations to be exposed on the virtualized service and click **Next**. Alternatively, click **Select All** or **Select None** to select all or none of the operations defined in the WSDL file.
6. On the **WS-Policy Options** window, select **Secure this virtualized service with a WS-Policy** to use a WS-Policy to secure the virtualized service. If you select this option, the **Secure Virtual Service** dialog is displayed at a later step, which enables you to configure WS-Policy settings between the client and API Gateway and generate policies that API Gateway will enforce for messages it receives from clients.
7. On the **WS-Policy Options** window, select **Use the WS-Policy in the WSDL to connect securely to the back-end Web Service** to generate policies to secure the connection between API Gateway and the web service, based on the WS-Policy settings included in the WSDL file. This is enabled (and selected by default) only if the imported WSDL file includes WS-Policy information. Click **Next** to continue.
8. On the **Expose Services** window, select or enter a unique relative path on which to expose the virtualized service (for example, **Default Services**).

Note

If the WSDL file contains multiple services, you must select a relative path for each service before continuing. Select a service from the **Select a service** field, and then select or enter a unique relative path to expose that service on. Repeat for each service.

9. Click **Finish**. The configuration is prepared based on your selections and one or more of the following windows are then displayed:
   - **Secure Virtual Service**
     If you selected to secure the virtualized service with a WS-Policy, this window is displayed. For more information, see the section called "Configure a security policy".
   - **Configure Recipient Security Settings**
     This window is displayed if you selected to secure the virtualized service with a recipient WS-Policy. For more information, see the section called "Configure recipient security settings".
   - **Configure Initiator Security Settings**
     This window is displayed if selected to use the WS-Policy in the WSDL to secure messages sent from API Gateway (as initiator) to the web service. For more information, see the section called "Configure initiator security settings".
10. After you have completed configuring the security settings, the **Summary** window is displayed. This window summarizes the details for the imported WSDL file. It includes the location of the WSDL file, and a tab for each web service virtualized by API Gateway. Each tab includes the following:
    - The name of the virtualized service.
    - The path that the WSDL for the virtualized service is exposed on.
    - **Validation**:
      API Gateway uses the WSDL to validate incoming messages by default. Alternatively, you can select the check box and click the browse button to use a dedicated validation policy for all messages sent to the web service. For example, this enables you to delegate to a custom validation policy used by multiple web services.

- **Routing**:
  API Gateway routes to the displayed URL by default. Alternatively, you can select the check box and click the browse button to use a dedicated routing policy to send all messages on to the web service. For example, this enables you to delegate to a custom routing policy used by multiple web services.
- **WSDL Access Options**:
  Select the **Allow the Gateway to publish WSDL to clients** option to make the WSDL for this web service available to clients. This option is selected by default. The published WSDL represents a virtualized view of the web service. Clients can retrieve the WSDL from API Gateway, generate SOAP requests, and send them to API Gateway, which routes them on to the web service. For more details, see the section called “Publish the WSDL”.

These options enable you to configure the underlying auto-generated **Service Handler** without having to navigate to it in the **Policies** tree. Review the information on each tab and click **OK** to close the wizard.

Configure a security policy

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

1. Select a policy from the **Gateway Policy** field. 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. Select a message-level request policy from the **Request Policy** field. The available policies are as follows:
   - Encrypt SOAP Body
   - Sign SOAP Body
   - Sign and Encrypt SOAP Body
3. Select a message-level response policy from the **Response Policy** field. The available policies are the same as for **Request Policy**.
4. Click **OK**.

Configure recipient security settings

If API Gateway is configured as a recipient for the client, the **Configure Recipient Security Settings** window is displayed. This window enables you to configure the filters used to implement the rules required by the WS-Policy settings you selected in the **Secure Virtual Service** window. The exact sequence of filters differs depending on the policies that you selected.

For example, if a policy with a SAML token is selected, the **Validate SAML Authentication Assertion** filter is displayed, whereas if a WS-Policy requiring a WS-Security `UsernameToken` is selected, the **Validate WS-Security Username Token** filter is displayed. The effort required to configure these filters is minimal because most of the fields are populated automatically from the WS-Policy assertions. For example, the layout, signing, encryption, and key wrapping algorithms, key referencing method, user name digest, and clear password are all automatically populated from the WS-Policy assertions. This means that you only need to configure a small number of settings, such as the signing key, encryption certificate, and timestamp validity period.

Configure initiator security settings

If API Gateway is configured as an initiator of the web service, the **Configure Initiator Security Settings** window is displayed. This window enables you to configure the filters used to implement the rules required by the WS-Policy settings specified in the imported WSDL file. The exact sequence of filters differs depending on the WS-Policy assertions contained in the WSDL.

For example, if an `sp:SamlToken` assertion is specified, the wizard contains a window for the **Insert SAML Authentication Assertion** filter. The effort required to configure these filters is minimal because most of the fields are populated automatically from the WS-Policy assertions 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.

Configure recipient policy filters

The following tables describe the filters that can be created when API Gateway is configured as a recipient. You can configure these filters in the **Configure Recipient Security Settings** window. For simplicity, only filters that require manual input are shown, and only the fields that might require manual input are detailed in the tables.

Insert Timestamp Filter

Field Name	Description
Expires In	You can 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 can specify a drift time value to allow for a time differential between the

Field Name	Description
	clock on the machine hosting API Gateway and the machine hosting the 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 recipient policy, you must configure an **Insert MTOM Content** filter. Configure an XPath expression that points to the base64-encoded element content to insert and create an MTOM type attachment for.

Configure initiator policy filters

The following tables describe the filters that can be created when API Gateway is configured as an initiator. You can configure these filters in the **Configure Initiator Security Settings** window. For simplicity, only filters that require manual input are shown, and only the fields that might require manual input are detailed in the tables.

Insert WS-Security Timestamp

Field Name	Description
Expires In	You can 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 user name 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, you can enter any value 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. 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 can specify a drift time value to allow for a time differential between the clock on the machine hosting API Gateway and the machine hosting the web service.
Issuer Name	Select the alias of the certificate from the certificate store 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

Field Name	Description
	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 an initiator policy, you must configure an **Extract MTOM Content** filter. Configure an XPath expression that points to the base64-encoded element content to extract and create an MTOM type attachment for.

Edit the recipient or initiator WS-Policy

To edit a previously configured WS-Policy (for example, to change the signing key in the auto-generated policy), right-click the web service in the Policy Studio tree, and select **WS-Policy > Edit Recipient WS-Policy** or **WS-Policy > Edit Initiator WS-Policy**. You can also add or remove a recipient WS-Policy after import by selecting the **WS-Policy > Add Recipient WS-Policy** or **WS-Policy > Remove Recipient WS-Policy** options. These options are described as follows:

Edit Recipient WS-Policy
If you configured a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), you can edit its filters using this option. If you did not configure a recipient WS-Policy during WSDL import, this option is disabled.

Add Recipient WS-Policy
If you did not configure a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), this option enables you to add a recipient WS-Policy after import. The **Secure Virtual Service** window is displayed when you select this option followed by the **Configure Recipient Security Settings** window. This enables you to select and configure a WS-Policy to enforce between the client and API Gateway.

Remove Recipient WS-Policy
If you configured a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), you can remove it using this option. If you did not configure a recipient WS-Policy during WSDL import, this option is disabled.

Edit Initiator WS-Policy
If the imported WSDL file included WS-Policy assertions, you can edit the filters used to implement the WS-Policy assertions in the WSDL using this option. However, if there was no WS-Policy in the imported WSDL file, you cannot use this option. You cannot add an initiator WS-Policy after WSDL import because that would break the contract between 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 can reimport the WSDL.

Configure a recipient WCF WS-Policy

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 API Gateway, the `svcutil` .NET web services utility can consume the WS-Policy and auto-generate clients that communicate securely with 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 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`		

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

Important

If you intend to consume the WS-Policy exposed by 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 API Gateway can be consumed by `svcutil`, you need to make additional configuration changes to the auto-generated WCF client to communicate securely with API Gateway. For more details on making any necessary configuration changes, see your WCF documentation.

Remove 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. To view the configured policy, double-click the **Service Handler**, and select 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 API Gateway, and is determined by the WS-Policy selected in the **Secure Virtual Service** window.
- *Initiator security context*: This is between API Gateway and the back-end web service, and is determined by the WS-Policy contained in the imported WSDL file 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-side WS-Policy only
In this case, a recipient WS-Policy is configured in the **Secure Virtual Service** window to protect a virtual service exposed by API Gateway. The recipient WS-Policy defines the security contract between the client and API Gateway. Any security tokens sent by the client are intended for consumption by API Gateway. They are *not* intended for the back-end web service.

For example, the web service might not understand SAML, WS-Security, XML Signature, and so on, which might 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 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 WS-Policy only
In this case, the WS-Policy is contained in the imported WSDL file. The WS-Policy defines the security contract between API Gateway and the back-end web service defined in the WSDL. On the request side, any security tokens sent by the client to API Gateway, which are out of scope of the initiator WS-Policy between API Gateway and web service, are removed before 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 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 API Gateway, which might contradict the rules in the initiator contract (between API Gateway and the web service) must be stripped out before 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 API Gateway. Therefore, any tokens returned by the web service are only intended for use by API Gateway. They are *not* intended for consumption by the client. In other words, the security context is only between API Gateway and the web service. If the web service returns a `UsernameToken`, it is consumed by 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** window (recipient case). This scenario includes both the *recipient security context* between the client and API Gateway, and the *initiator security context* between 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 API Gateway requires a `UsernameToken`, but the initiator contract between 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.

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.

Related information

For more information on manually configuring policies to protect web services (for example, if a WSDL file is not available), see the *Configure policies manually* topic.

The **Web Service** filter is the main filter generated when a WSDL file is imported into the web service 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. For more information, see the *Web service filter* topic.

For more information on the web service repository, see the *Manage the web service repository* topic.

For more information on the global cache of WSDL and XML schema documents, see the *Manage WSDL and XML schema documents* topic.

Configure policies manually

Overview

This topic describes how to use Policy Studio to configure an API Gateway policy manually. It also applies to cases where a Web service 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 topic 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 *Configure policies from WSDL files* topic.

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 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 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 API Gateway instance, which maps incoming requests on a particular URI to the new policy. Complete the following steps:

1. In the tree view of Policy Studio, right-click the **Default Services** node, which can be found under the **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 API Gateway
Before the new configuration changes can take effect, you must deploy them to API Gateway. You can do this by clicking the **Deploy** button on the right of the toolbar. Alternatively, press the F6 key.

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 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 window to find out more about the configuration fields that are available on each window.

Configure 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 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:

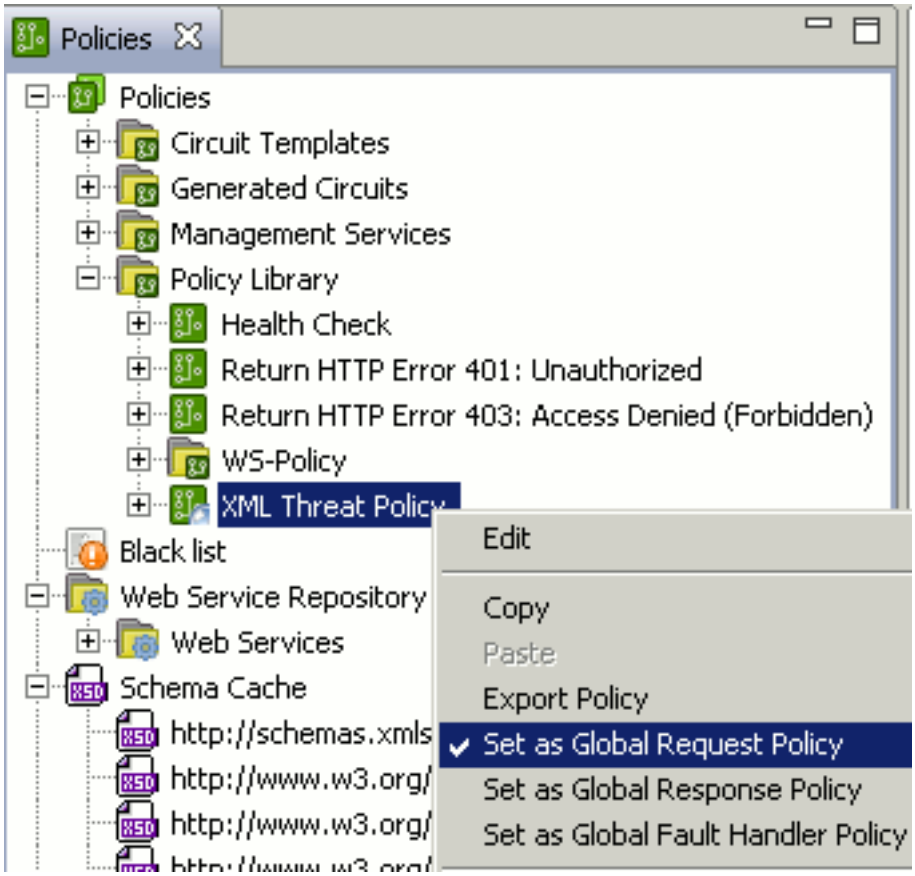
Role	Label	Description
Global Request Policy	`system.policy.request`	Executed globally for all messages passing through the API Gateway on the request path.
Global Response Policy	`system.policy.response`	Executed globally for all messages passing through the API Gateway on the response path.
Global 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.

Select 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 API Gateway.

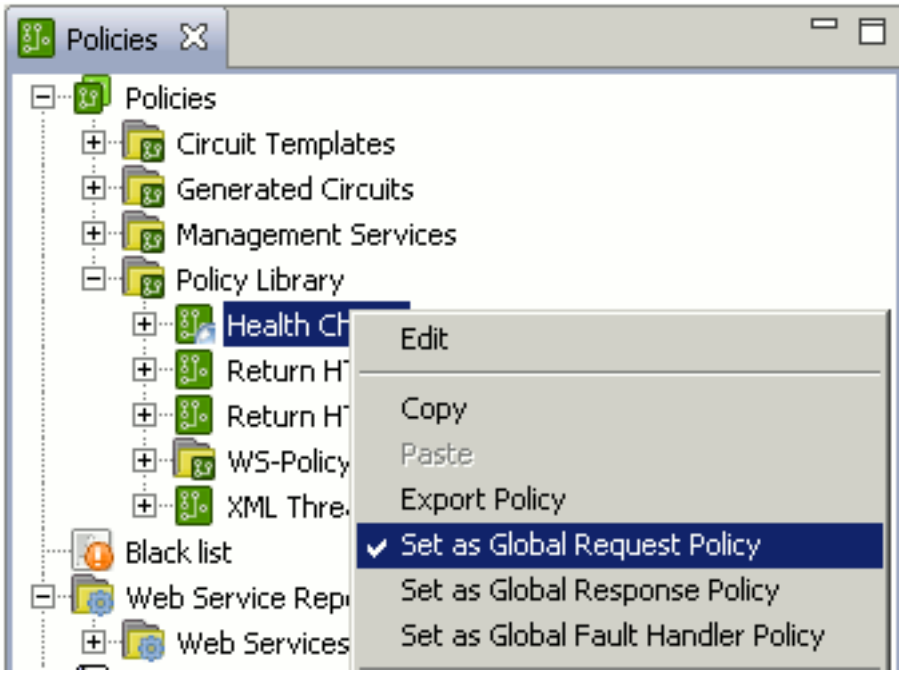
The following example shows the **XML Threat Policy** set as the **Global Request Policy**. The policy node labeled for the specific role is displayed with a globe icon:



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:

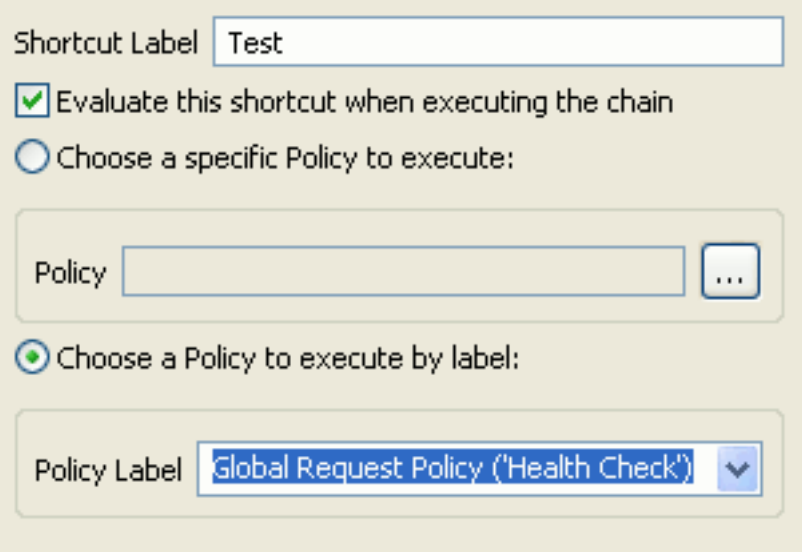


Important

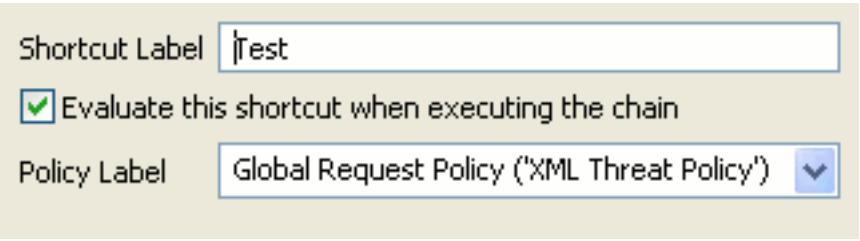
You can not select a policy for a specific role if, in doing so, you create a loop in the policies. For example, if a **Policy Shortcut Chain** filter has a reference to a labeled policy, and the filter's parent policy is marked as the labeled policy, the filter would call back to itself in a loop. This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

Configure 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:



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)**.

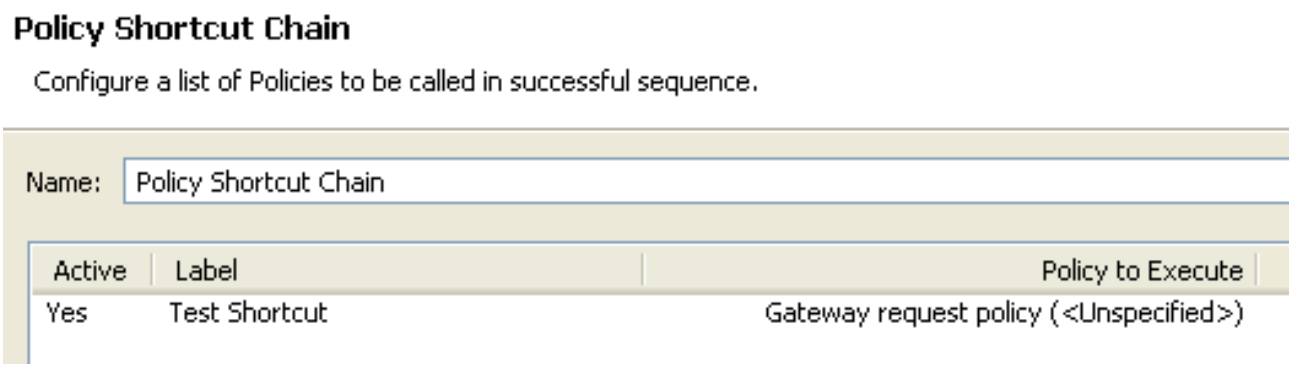


For more details on configuring these windows, see the *Policy Shortcut Chain* topic.

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** check box 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 shortcut, the link is not evaluated.

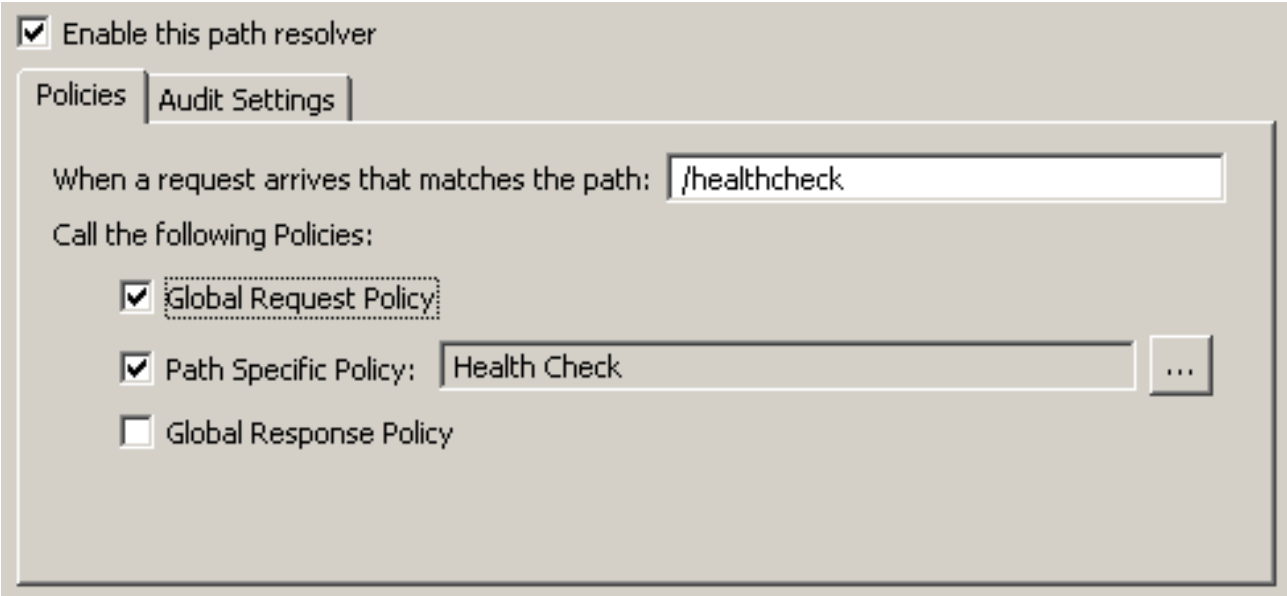
In this example, the table shows that the shortcut is configured to point to the labeled policy, but the label does not resolve to a policy (for example, it is unspecified because there is no policy in the specified role):



Configure 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 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**.

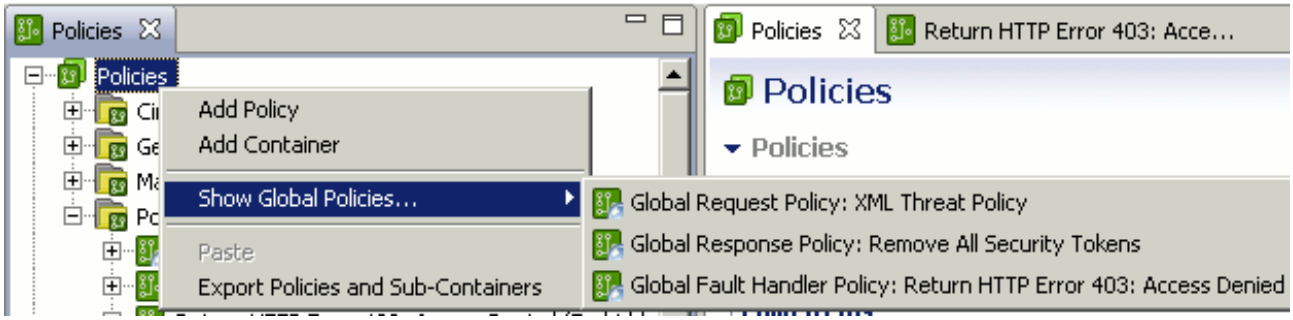
You can configure a global policy at the service level to run as part of a policy chain invoked when incoming messages are received by API Gateway. The following example shows the **Global Request Policy** configured to run first on the `/healthcheck` relative path:



For more details, see the *Configure relative paths* topic.

Show 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:



Note

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

Configure policy assemblies

Overview

In certain cases, you may need to convert a policy into a modular reusable piece of functionality that can be called from other policies. For example, you have a complex policy that creates a WS-Security `UsernameToken`, inserts it into the message, and subsequently creates an XML Signature over the token and SOAP body. Depending on the message recipient, the content may need to be signed using slightly different settings. One service might require a `<sp:Basic128/>` algorithm suite, while another might require `<sp:Basic256/>`).

Similarly, subtle differences in security requirements may require the token and signature to be generated differently. 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 for such cases, interoperating with different vendor services can become arduous. This involves creating several complicated policies that might only differ in one field in each filter. To avoid this duplication, you can create a *policy assembly* 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>`, or signing key to use), the policy assembly can use selectors for these fields, which are configured dynamically at runtime. For more details, see *Selecting configuration values at runtime*.

The policy assembly advertises that it requires configuration details to be called generically from other policies. For example, it requires the key to sign the message. By templating the signing policy as a policy assembly, and making it available to call from other policies like any other filter, the caller must set the signing key for the policy assembly. In this way, different policies that require a signed `UsernameToken` can call the same policy assembly. 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 assembly, it is displayed in the Policy Studio filter palette, and you can drag and drop it into any policy that requires the functionality in the assembly. You must configure any fields required by the policy assembly when it is dragged and dropped on to the canvas of another policy.

Configure a policy assembly

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

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

5. When finished, click **OK**.
6. Click the **Deploy** button in the toolbar to deploy the newly created policy assembly to API Gateway.

Apply a policy assembly

When a policy is configured as a policy assembly, it is available for reuse in the Policy Studio filter palette. Draging and droping the policy assembly on to the policy canvas displays the **Policy Activation Filter** window for that policy assembly. 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 assembly (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 attribute value or selector (for example, `${http.headers}`).

Specifying monitoring settings
The **Traffic Monitor** tab enables you to set the following filter-level monitoring settings:

- **Record outbound transactions**:
  Select whether to record outbound message transactions sent from 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.

Manage API Gateway deployments

Overview

When connected to the Admin Node Manager server in Policy Studio, you can deploy configurations to API Gateway instances running in groups in a domain. In Policy Studio, the **Group / API Gateway** 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 enable policy developers and administrators to centrally manage the policies that are enforced at all nodes throughout the network.

In addition, Policy Studio enables you to compare and merge differences between versions of the same policy. Policies can be merged, and deployed to any running instance that is managed by Policy Studio. One of the most powerful uses of this centralized management capability is in transitioning from a staging environment to a production environment. For example, policies can be developed and tested on the staging environment, and when ready, they can be deployed to all instances deployed in the production environment.

Connect to a server in 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.

When 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,

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 **Group / API Gateway** topology view is displayed. This displays the list of server instances currently managed by the Admin Node Manager in Policy Studio, and enables you to manage the configuration for server instances.

Edit a server configuration in Policy Studio

The **Group / API Gateway** 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 Ser-**

vices, **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**). You can also deploy configuration packages in the **Group / API Gateway** topology view. For more details, see *Deploy API Gateway configuration*.

Manage deployments in 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 *Start the API Gateway tools*.

Compare and merge configurations in Policy Studio

You can compare and merge differences between the currently loaded API Gateway configuration with a configuration stored in a deployment package (.fed file). Click the **Compare** button on the Policy Studio toolbar to select a `.fed` file to compare the current configuration against. The results are displayed in the **Compare/Merge** tab.

For example, you can view the differences made to particular fields in an Authentication filter that occurs in both configurations. When a difference is located, you can merge the differences, and thereby update the fields in the Authentication filter in the current configuration with the field values for the same Authentication filter in the deployment package.

For more details,

Manage Admin users in API Gateway Manager

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

For more details on role-based access,

Configure policies in 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.

Deploy API Gateway configuration

Overview

You can edit API Gateway configuration offline, and then deploy later to a specified API Gateway instance using API Gateway configuration packages. A *deployment package* is a `.fed` file that contains all API Gateway configuration. This includes policies, listeners, external connections, users, certificates, and environment settings. A *policy package* is a `.pol` file that contains policies, listeners, external connections, and environment settings. While an *environment package* is an `.env` file that contains users, certificates, and environment settings. The content of the `.fed` file is equivalent to the combined contents of the `.pol` and `.env` files.

A *package property* is a name-value pair that applies to a specific configuration package (`.fed`, `.pol`, or `.env`). Specifying a property associates metadata with the configuration in that package. For example, the **Name** property with a value of `Default Factory Configuration` is associated with a default installation. For more details on configuration packages and properties, see the *API Gateway Deployment and Promotion Guide*.

You can use Policy Studio to create packages (`.fed`, `.pol`, or `.env`). You can also use Policy Studio to deploy an existing package or factory configuration on selected API Gateway instances, or to deploy a currently loaded configuration. You can use the API Gateway Manager console to deploy a package in a browser. Alternatively, you can use the `managedomain` script to create and deploy deployment packages (`.fed` files) on the command line.

Create a package in Policy Studio

You can create an API Gateway configuration package for a currently loaded configuration, or for a selected server instance in the **Group / API Gateway** topology view.

Currently loaded configuration
To create a package (`.fed`, `.pol`, or `.env`) for a currently loaded API Gateway configuration, perform the following steps:

1. In the main menu, select **File** > **Save** followed by the appropriate option:
   - **Deployment Package**
   - **Policy Package**
   - **Environment Package**
2. Enter a filename, and click **Save**.

Group / API Gateway view
To create a deployment package (`.fed`) in the API Gateway **Group / API Gateway** topology view, perform the following steps:

1. Right-click a server instance in the tree, and select **Save Deployment Package**.
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**.

Configure package properties in Policy Studio

You can view or modify API Gateway configuration package properties for a currently loaded configuration, or for a selected server instance in the **Group / API Gateway** topology view.

Currently loaded configuration
To view and modify configuration properties for a currently loaded API Gateway configuration, perform the following

steps:

1. In the Policy Studio tree, and select **Package Properties** > **Policy** or **Environment**.
2. Enter values for the appropriate configuration properties (for example, **Name**, **Description**, or **Version**).
3. If you wish to create any additional properties (for example, **Department**), click the green (+) button on the right, and enter a property value (for example, `Engineering`).
4. If you wish to remove a property, click the red (x) button on the right of the property.
5. Click **Save** on the top right of the screen.

Group / API Gateway view
To view and modify configuration properties for a selected server instance in the API Gateway **Group / API Gateway** topology view, perform the following steps:

1. Right-click a server in the tree, and select **View/Modify Properties**.
2. Select the **Policy Properties** or **Environment Properties** tab.
3. Enter values for the appropriate configuration properties (**Name**, **Description**, and **Version**).
4. If you wish to create any additional properties (for example, **Department**), click the green (+) button on the right, and enter a property value (for example, `Engineering`).
5. If you wish to remove a property, click the red (x) button on the right of the property.
6. Click **Update Configuration Properties**.
7. Click **OK**.

For details on customizing the default package properties displayed,

Deploy packages in Policy Studio

You can use the Policy Studio to deploy configuration packages to selected API Gateway instances in the **Group / API Gateway** topology view.

Deploy a deployment package
To deploy an existing deployment package (`.fed` file) in the **Group / API Gateway** 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 **Group** 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 configuration contained in a single Deployment Package**.
4. In the **Deployment Package** field, click **Browse for .fed**, and select the `.fed` file.
5. Click **Deploy** to upload the package to the Admin Node Manager and deploy to the selected server(s).
6. When the package has deployed, click **Finish**.

Deploy policy and environment packages
To deploy an existing policy package (`.pol` file) and environment package (`.env` file) in the **Group / API Gateway** 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 **Group** 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 configuration contained in Policy Package and Environment Package**.
4. In the **Policy Package** field, click **Browse for .pol**, and select the `.pol` file.
5. In the **Environment Package** field, click **Browse for .env**, and select the `.env` file.

6. Click **Deploy** to upload these packages to the Admin Node Manager and deploy to the selected server(s).
7. When the package has deployed, click **Finish**.

Deploy a factory configuration in Policy Studio

To deploy a default factory configuration in the **Group / API Gateway** 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 **Group** 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).

Deploy currently loaded configuration in Policy Studio

You can also deploy updates to a currently loaded configuration in Policy Studio when editing the configuration. To deploy a currently loaded configuration, 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 **Group** list, and select the server instance(s) in the box below.
3. Click **Deploy**, and wait for the deployment to complete.
4. Click **Finish**.

Push configuration to a group in Policy Studio

When there is more than one API Gateway instance in a group, and configuration becomes out of sync between instances, you can select which configuration to push to the group. Perform the following steps in Policy Studio:

1. In the **Group / API Gateway** topology view, right-click the API Gateway instance configuration that you want to deploy to other instances in the group.
2. Select **Push this API Gateway's configuration to the group**.
3. In the wizard, select the API Gateway instances in the group that you wish to deploy to.
4. Click **Deploy** to deploy to the selected instances in the group.
5. Click **Finish**.

View deployment results in Policy Studio

When you click **Deploy**, the **Deployment Results** screen is displayed, 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**.

Cancel 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 instance. The corresponding Console **Deployment Log** is also available in the **Console** view.

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

Deploy on the command line

You can create and deploy a deployment package (`.fed`) 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
```

Deploy packages in API Gateway Manager

You can also use the API Gateway Manager web console to deploy configuration packages to a group of API Gateway instances. This functionality is available on the default **Dashboard** tab. For more details,

Compare and merge API Gateway configurations

Overview

In the Policy Studio, you can compare the currently loaded API Gateway configuration with a configuration stored in a deployment package (`.fed` file). You can also merge any differences between the configurations.

Differences between configurations are identified as additions, deletions, or conflicts. When merging configurations, you can choose which differences to merge.

Note

The currently loaded configuration can only be compared with a configuration stored in a deployment package (`.fed`) or in a server configuration file (`.xml`). You cannot compare against a policy package (`.pol`) or environment package (`.env`). For more information on configuration packages, see the *Deploy API Gateway configuration* topic.

Compare and merge configurations

To compare the currently loaded configuration against the configuration in a `.fed` file, follow these steps:

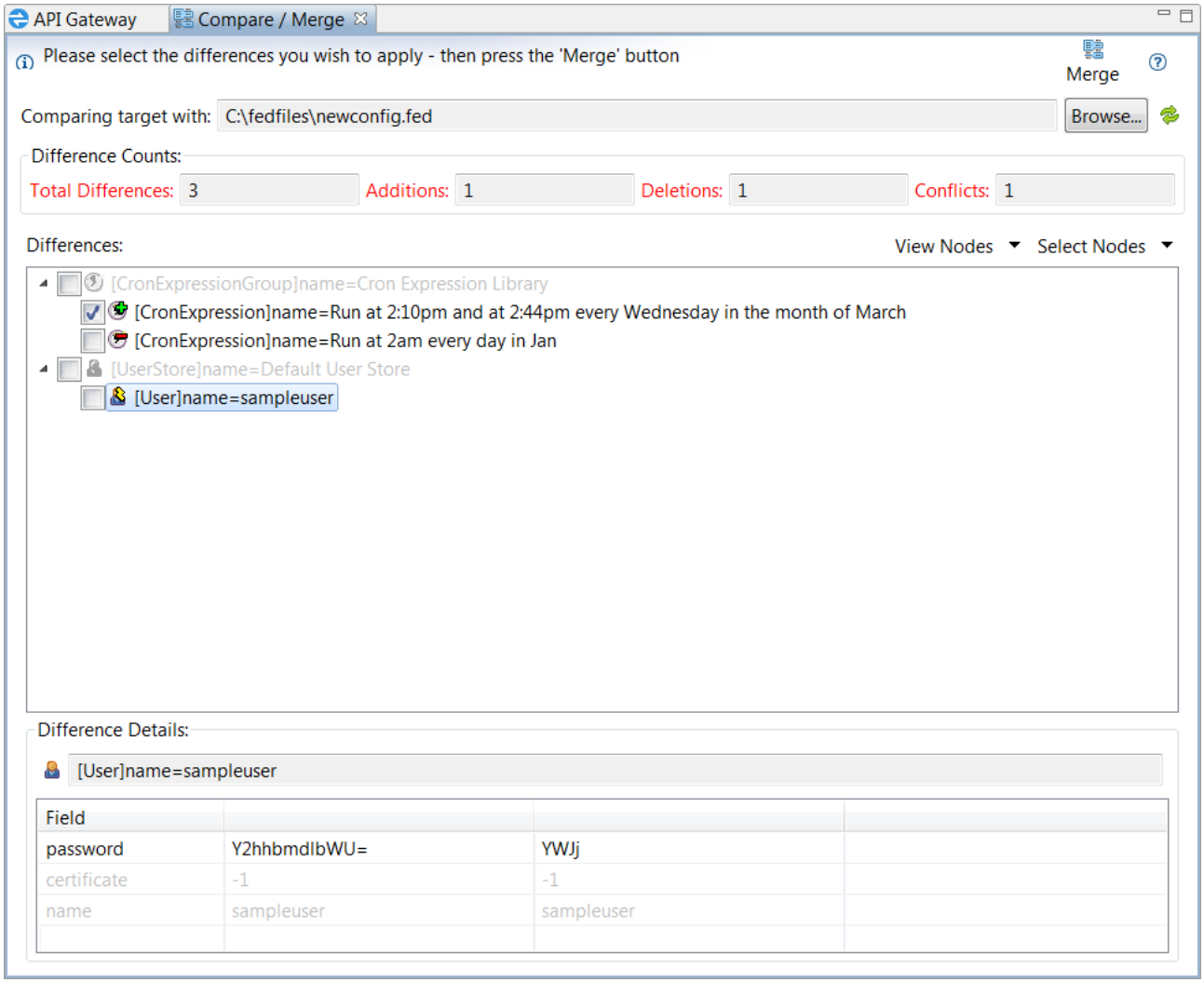
1. Click the **Compare** button on the Policy Studio toolbar.
2. In the **Comparing target with** field, click the **Browse** button to choose a `.fed` file to compare the configuration with.
3. Enter the passphrase for the configuration, if one has been set, and click **OK**. The configurations are compared and the results are displayed in a tree view. Only entities with differences are shown.
4. To see detailed differences, click a configuration entity in the tree. The differences for that entity are displayed in the **Difference Details** pane.
5. To merge differences into the currently loaded configuration, select the check box next to each difference to be merged, and click the **Merge** button at the top right of the window.

Note

If you modify the currently loaded configuration after the **Compare and Merge** tab is opened, click the **Refresh** button to refresh the comparison and show any new differences.

Comparison results

The following figure shows the result of a comparison:



The **Difference Counts** pane shows the number of differences in total, the number of additions, the number of deletions, and the number of conflicts.

The **Differences** tree view shows all of the differences in the configuration entities:

- Entities with green plus icon are additions. These entities exist in the `.fed` file but not in the currently loaded configuration.
- Entities with a red minus icon are deletions. These entities exist in the currently loaded configuration but not in the `.fed` file.
- Entities with a yellow warning icon are conflicts. These entities exist in both configurations but are not the same.

The **Difference Details** pane shows the values of the fields in each configuration when you click on an entity in the tree view. The second column shows the values of the fields in the `.fed` file, and the third column shows the values of the fields in the currently loaded configuration. The fields that are different in each configuration are highlighted.

In the preceding figure:

- The cron expression `Run at 2am every day in Jan` is a deletion.
- The cron expression `Run at 2:10pm and at 2:44pm every Wednesday in the month of March` is an addition.
- The user `sampleuser` is a conflict, because the password field has a different value in each configuration.

Some configuration entities contain references to other entities. In this case, an icon is displayed for the field in the **Difference Details** pane. Double-click a row with an icon to view the differences for those entities.

Filter differences

To filter nodes from the **Differences** tree view based on their type, click **View Nodes**, and select from the following options:

- **Additions**
- **Deletions**
- **Conflicts**

All differences are shown by default.

Select differences for merging

To select nodes for merging from the **Differences** tree view based on their type, click **View Nodes**, and select from the following options:

- **Additions**
- **Deletions**
- **Conflicts**

Additions are selected by default.

Manage 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. To manage Admin users, click 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. However, *API Gateway users* provide access to the messages and services protected by the API Gateway. For more details, see *Manage API Gateway users*.

Admin user privileges

After installation, a single Admin user is defined in the API Gateway Manager with a user name 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.

Remove the default Admin user
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 Gateway Administrator`

- `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 Gateway Administrator`	API Gateway Manager	Read/write access to API Gateway Manager.
`API Gateway Operator`	API Gateway Manager	Read-only access to API Gateway Manager.
`Deployer`	Deployment scripts	Deploy a new configuration.
`KPS Administrator`	KPS Web UI	Perform create, read, update, delete (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 Gateway Administrator`

Add 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 Gateway Administrator`).
7. Click **Create**.

Remove an Admin user

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

Reset 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**.

Manage 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 to enable for this Admin user in the dialog (for example, `Policy Developer` and/or `API Gateway Administrator`).
4. Click **OK**.

Edit 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 more details on role-based access, see the *API Gateway Administrator Guide*.

Manage 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.

Connect 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.

Important

To manage API Gateways in your network, you must connect to the Admin Node Manager server URL.

Connect 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 *Deploy API Gateway 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.

Unlock 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 "Connect 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, 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 Policy Studio, each of which are discussed briefly in the sections below:

- API Gateway Settings
- Web Service Repository
- API Gateway Instances
- Policies
- Certificates and Keys
- API Gateway Users
- Alerts
- External Connections
- Caches
- Black list
- White list
- Document Bundles
- Scripts
- Stylesheets

API Gateway settings

You can configure the underlying configuration settings for API Gateway using the **Server Settings** node in the Policy Studio tree. This includes the following settings:

- General
- Logging
- Messaging
- Monitoring
- Security

For more details, see the *API Gateway Administrator Guide*.

Web service repository

The easiest way to secure a Web service with API Gateway is to import the Web Services Description Language (WSDL) file for the service using 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 service repository, making sure to update the URL of the Web service to point at the machine on which API Gateway is running instead of that on which the Web service is running. Consumers of the Web service can then query API Gateway for the WSDL file for the Web service. The consumer then knows to route messages to 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 service repository offers a very simple way of securing a Web service with minimal impact on consumers of that service. Because of this, the Web service repository should be used as the primary method of setting up policies within Policy Studio. For more information on using the repository to register a Web service, see the *Manage the web service repository* topic.

API Gateway instances

A single running instance of API Gateway 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, you might need to add several HTTP interfaces. For example, an HTTP interface and an SSL-enabled HTTPS interface.

Furthermore, you can add features such as the following at the API Gateway instance 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 API Gateway can read messages from HTTP, SMTP, FTP, JMS, or a directory, this enables it to perform protocol translation. For example, API Gateway can read a message from a JMS queue, and then route it on over HTTP to a Web service. Similarly, 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 API Gateway instances, see the *Configure API Gateway instances* topic.

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 API Gateway are found in the **Policy Library** policy container. This container is prepopulated 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

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 *Manage certificates and keys* topic.

API Gateway user store

Users are mainly used for authentication purposes in 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 *Manage API Gateway users* topic contains more details on how to create users, user groups, and attributes.

System alerts

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 API Gateway to send these alerts, see the *Configure system alerts* topic.

External connections

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, API Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, 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 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
- Proxy Servers
- Radius Clients
- SiteMinder Connections
- TIBCO Connections
- Tivoli Connections
- XKMS Connections

You can also use external connections to configure a group of related URLs. This is most useful to round-robin between a number of related URLs to ensure high availability. When 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 API Gateway to cache responses from a back-end Web service. For example, if 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. 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 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 API Gateway, it is added to all other caches.

For more details on how to configure 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 prepopulated 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 selector expression*
- *Threatening content*

WSDL and XML schema document bundles

The WSDL documents and XML schemas that API Gateway can use to validate incoming requests against are stored in a global cache. 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 cache of WSDL documents or XML schema documents by selecting **Resources > WSDL Document Bundles** or **Resources > XML Schema Document Bundles**. Select a child node to view its contents. To add a schema, right-click the **XML Schema Document Bundles** node, and select **Add Schema**. For more details on adding XML schemas to the cache see the *Manage WSDL and XML schema docu-*

ments topic.

When you have imported your XML schemas, see the *Schema validation* topic 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 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 API Gateway to use scripts, see the topic on the *Scripting Language Filter*.

Stylesheets

The **Stylesheets** library contains the XSLT style sheets that API Gateway can use to transform incoming request messages. The **XSLT Transformation** filter enables you convert the contents of a message using these style sheets. 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 style sheet library by selecting **Resources > Stylesheets**. Select a child node to view or edit its contents. To add a style sheet, right-click the **Stylesheets** node, and select **Add Stylesheet**.

For more details on using the **Stylesheet Library** dialog to add style sheets, and on configuring API Gateway to use XSLT style sheets, 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 Policy Studio by right-clicking an item, and selecting **Show All References**. References are displayed in a tab at the bottom of the window.

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.

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.

Environmentalization setting

Environmentalization refers to configuring environment-specific settings for a particular target environment (for example, users, certificates, and external connections for a development environment). You can enable Policy Studio to display settings that have been environmentalized by selecting the **Indicate if configuration settings have been marked for environmentalization** option.

When this option is selected, you can environmentalize a selected field (for example, database URL) by clicking the globe icon to the right of the field. Alternatively, press Ctrl-E. When you have selected settings to be environmentalized, the field is disabled, and the globe icon is displayed on the right. You can manage settings that have been environmentalized under the **Environment Settings** node in the Policy Studio tree. For more details, see the *API Gateway Deployment and Promotion Guide*.

Management services

The Admin Node Manager and 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.

The **Management Services** server process, interfaces, and policies are displayed in the Policy Studio tree. The **Management Services** policy container is displayed in the tree under the **Policies** node. The **Management Services** HTTP interfaces are displayed under the **Listeners** node under the server instance. For more details, see the section called "Management services" in *Configure HTTP services*.

Important

You should only modify **Management Services** under strict advice and supervision from the Oracle Support team.

Policy color settings

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.
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 Policy Studio and the server for these changes to take effect. You should restart Policy Studio using the `policystudio -clean` command.

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 Password**.

Status bar setting

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.

Topology screen settings

The **Topology Screen** settings enable you to customize how configuration *package properties* are displayed in the **Topology View** table in Policy Studio. These are user-entered properties contained in deployment packages (`.fed`), policy packages (`.pol`), and environment packages (`.fed`). The **Group / API Gateway** and **Deployed by** columns in the **Topology View** are read only. You can customize all other columns to show package property values.

Specify default column values
You can specify the following default package property values in the **Column Value** column:

- `${manifest.policy.Name}`
- `${manifest.policy.Description}`
- `${manifest.policy.Version}`
- `${manifest.policy.VersionComment}`
- `${manifest.env.Name}`
- `${manifest.env.Description}`
- `${manifest.env.Version}`
- `${manifest.env.VersionComment}`
- `${manifest.root.Id}`
- `${manifest.root.Timestamp}`

Add custom properties
You can also add custom package properties. For example, to add a custom policy package property to be displayed in the **Topology View**, perform the following steps:

1. Click **New**.
2. Double-click the value in **Column Header**, and enter `MyCustomPolicyField`.
3. Double-click the value in **Column Value**, and enter `${manifest.policy.MyCustomPolicyField}`.

Similarly, to add a custom environment package property, add a property with a **Column Header** of `MyCustomEnvField`, and a **Column Value** of `${manifest.env.MyCustomEnvField}`.

Customize the topology view table
You can add, edit, remove, or reorder the columns displayed in the **Topology View** using the **Topology Screen** settings. You can also specify the **Column Width** displayed.

For more details on configuring package properites, see the *API Gateway Deployment and Promotion Guide*.

Trace level setting

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 settings

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 service 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 check box (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).

WS-I Setting	Description
Assertion Description	Specify whether the description of each test assertion should be included in the report using the check box (unselected by default).
Verbose Output	Specify whether verbose output is displayed in the Policy Studio console window using the check box (unselected by default). To view the console window, select **Window** > **Show Console** from the Policy Studio main menu.

Important

On Linux/UNIX, when you download WS-I Testing Tools v1.1, you must run `dos2unix` on `/java/bin/Analyzer.sh` and `/java/bin/setenv.sh`. This is because both files do not have executable privileges set and have Windows line endings, so the shell interpreter is unable to use them.

For details on running the WS-I testing tools, see the *Manage WSDL and XML schema documents* topic.

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.

Filter 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**).

Configure 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.

Configure 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.

Manage the web service repository

Overview

The **Web Service Repository** stores information about web services whose definitions have been imported using Policy Studio. 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 API Gateway.

The web service repository enables you to register web services by importing a WSDL file, and to group related web services into groups. When you register a web service in the web service repository, Policy Studio auto-generates a **Service Handler** that is used to control and validate requests to the web service and responses from the web service. If a web service is updated after initial import, you can resynchronize with it to import new versions of the WSDL and XML schemas, and the **Service Handler** is regenerated to reflect the updates.

This topic describes how to manage web services and groups, and explains what happens when you import a WSDL file to register a web service. This topic also describes how to export a registered web service, how to update (or resynchronize) an existing web service, and how to expose additional operations on a web service.

Manage web services and groups

The **Web Service 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. To import a WSDL file into the default group, right-click the **Web Services** node, and select **Register Web Service**.

Alternatively, to add a new web service group, right-click the default **Web Services** group, or the **Web Service Repository** node, and select **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**.

Register a web service

Registering a web service involves importing the WSDL file that contains the definitions for the web service. Policy Studio provides an **Import WSDL** wizard to make registering a web service a simple process that requires minimal manual intervention. For more information on registering a web service by importing a WSDL file, see the *Configure policies from WSDL files* topic.

The **Import WSDL** wizard auto-generates policies and service handlers for each web service imported. These are automatically configured wherever possible, based on the imported WSDL. This means that only a small number of fields need to be configured manually.

After a web service has been registered using the **Import WSDL** wizard, the WSDL for the service is *published* by API Gateway. 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 "Publish the WSDL".

Tip

You can also register a web service using a script. Sample scripts are provided in the `INSTALL_DIR/apigateway/samples/scripts/ws` directory.

Results of registering a web service

Service handlers and policies are auto-generated when you register a web service by importing a WSDL file in Policy Studio. The specific policies that are created depends on whether you configured a policy to enforce security between

the client and API Gateway, and whether the imported WSDL file contained any WS-Policy assertions. The following list summarizes what is created by the wizard:

Web service
A new web service tree node is created in the **Web Service Repository** tree for each web service that you selected to expose operations from in the imported WSDL. The web service is created in the **Web Services** group by default.

Click the new web service node to list the WSDL file for the service and any imported WSDL files and XML schemas. To view the source for a WSDL or XML schema file, click the file and a read-only view of the source is displayed.

Note

It is important to realize that the WSDL displayed in this view represents the WSDL that is exposed to the client. Therefore, it contains only those operations that were selected during the import process, together with any recipient WS-Policy that has been applied to the virtualized web service. You can view the WSDL in its original form under the **Resources > WSDL Document Bundles** node.

You can add, remove, or edit recipient WS-Policies for the service from this node. For example, to edit an existing recipient WS-Policy, right-click the web service node, and select **WS-Policy > Edit Recipient WS-Policy**. If the imported WSDL file included WS-Policy assertions, you can also edit the initiator WS-Policy (for example, to change the signing key). Right-click the web service node and select **WS-Policy > Edit Initiator WS-Policy**. For more information, see the section called “Edit the recipient or initiator WS-Policy”.

WSDL resources
A new WSDL document bundle is created for the imported WSDL file under the **Resources > WSDL Document Bundles** tree node. This document bundle contains the original WSDL document for the back-end web service, as retrieved at the time of the WSDL import. If multiple versions of this WSDL have been imported, each version is available here. The document bundle also contains the original versions of any resources imported by the WSDL, such as other WSDL files or XML schemas.

Click the WSDL document bundle to list the versions that have been imported. Click a particular version to list the WSDL file and any imported WSDL files and XML schemas, as they appeared for that version. To view the source for a WSDL or XML schema file, click the file and a read-only view of that version of the source is displayed.

For more information about WSDL document bundles, see *Manage WSDL and XML schema documents*.

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 is the name of the service (for example, **HelloWorldService**). This policy contains the **Service Handler** for the service. The service handler is an auto-configured **Web Service Filter**.

Service handler
The **Service Handler** 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.

Security policies
If you configured a WS-policy to enforce security between the client and API Gateway (as described in the section called "Configure a security policy"), or if the imported WSDL file contained WS-Policy assertions, a number of additional policies are automatically created in a generated policy container named **WSPolicy**. Any recipient policies are created in a container named **Recipient** and any initiator policies are created in a container named **Initiator**. These generated policies include the filters required to generate and validate the relevant security tokens (for example, SAML tokens, WS-Security `UsernameToken` elements, 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.

Export a web service

To export a web service, right-click on the web service node under the **Web Service Repository** and select **Export Web Service**.

The following items are exported:

- All circuit containers for the web service, including policies and containers that were generated as part of WS-Policy configuration.
- The current version of the WSDL and its schemas.

Note

If multiple versions of the WSDL are available, only the current version is exported. The complete history of the WSDL is *not* exported.

- Optionally, other configuration used by the policies can be exported (for example, remote host configuration).

Update a web service

Over the lifetime of a web service, the service definition for the web service can change. You can manage the lifecycle of a web service easily, by using the **Resynchronize Web Service** option in Policy Studio. This enables you to update the web service definition for a service by revisiting the location of the WSDL.

To update a web service, right-click the web service node, and select **Resynchronize Web Service**. You are prompted for the location of the latest version of the WSDL. This defaults to the location from which the WSDL was originally loaded. API Gateway compares the contents of the WSDL and any of its imported schemas to those stored in its cache. If the contents are different, it creates a new version of the WSDL.

Examples of possible updates to the WSDL or XML schemas that trigger creation of a new version are as follows:

- **Service name (local part)**
  If the service name has changed in the WSDL, the import exits with the error `"Couldn't find target Service in this WSDL"`. You can import this WSDL definition as a new web service using the **Register Web Service** option.
- **WSDL namespace**
  If the namespace has changed in the WSDL, the import continues with a warning.

Note

Namespace changes result in a new node being created under the **Resources > WSDL Document Bundles** tree. Any subsequent resynchronizations of the service will result in new versions being added to this node (assuming no further namespace changes).

- **WSDL 2.0**
  If the WSDL has changed to use the WSDL 2.0 specification, import exits with an error. API Gateway supports WSDL 1.1 only, it does not support WSDL 2.0.
- **Operation added**
  If a new `operation` has been added to a `portType` in the WSDL, import succeeds. API Gateway regenerates the configuration, using older operations as a template for the WS-Policy settings, if attached. The service handler for the web service is updated to include a resolver for the new operation.
- **Operation modified**
  If an `operation` has been modified on a `portType` in the WSDL (for example, changes to the `input`, `output`, or `fault`), import succeeds. API Gateway regenerates the configuration and updates the service handler for the web service.
- **SOAPAction**
  If the `SOAPAction` has changed for an `operation` in the WSDL, import succeeds. API Gateway regenerates the configuration and updates the service handler for the web service.
- **Operation removed**
  If an `operation` has been removed from a `portType` in the WSDL, import succeeds. API Gateway removes the operation from the configuration. The service handler for the web service is updated to remove the resolver for the removed operation.
- **XML schema (when WSDL validation is in use)**
  If you are using the WSDL to validate incoming messages (this is the default option when you import a WSDL file), and the schemas in the WSDL have been updated, API Gateway retrieves the updated schemas.

Updating a web service is *not* possible for the following types of WSDL or XML schema changes:

- Port, binding, or operation changes that require user intervention for WS-Policy configuration.
- Port change (new endpoint location).
- Port added (SOAP flavor).
- Port change (new binding).
- Binding style or use change.
- WS-Policy added.
- WS-Policy modified.
- WS-Policy removed.
- Operation message part changes (for example, reference to an element changes to a type, part name changes, and so on).
- XML schema changes (when custom schema validation is in use).

Change the operations exposed by a web service

When you register a web service by importing a WSDL file, you are prompted to select the operations that are to be exposed to the client. You can change the operations that are exposed after import by using the **Select Exposed Operations** option in Policy Studio. For example, if you imported a web service containing two operations, `foo` and `bar`, and you only selected the `foo` operation at import time, you can use this feature to also expose the `bar` operation after import.

To change the operations exposed by a web service, right-click the web service node, and select **Select Exposed Operations**. All of the operations defined in the WSDL file are displayed, and the operations that are currently exposed are selected. Select the operations to be exposed and deselect the operations that are not to be exposed and click **OK**.

API Gateway regenerates the configuration. The service handler for the web service is updated to include resolvers for any newly exposed operations, and to remove resolvers for any removed operations. The WSDL exposed to clients of the virtualized service is updated to reflect the changes.

Publish the WSDL

When the WSDL has been imported into the web service 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 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 API Gateway, this URL is no longer accessible to consumers of the service. Because of this, clients must send SOAP messages through API Gateway to access the service. In other words, they must now address the machine hosting API Gateway instead of that directly hosting the service.

When returning the WSDL to the client, 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 API Gateway resides. API Gateway is responsible for routing messages on to the machine hosting the service.

Assuming that 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 client retrieves this modified WSDL file, it routes messages to the machine hosting API Gateway instead of attempting to directly access the web service.

Access the WSDL
For the client to access this modified WSDL file, Policy Studio provides a WSDL retrieval facility whereby clients can query the web service repository for the WSDL file for a particular web service. To do this, the client must specify `WSDL` on a request 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 API Gateway rather than to the web service directly.

Publish to UDDI
For details on how to publish a WSDL file registered in the web service repository to a UDDI registry, see the *Publishing WSDL files to a UDDI registry* topic.

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 instance 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 settings

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*

Kerberos configuration

Overview

The **Kerberos Configuration** screen enables you to configure API Gateway instance-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 check boxes 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 instance-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.

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

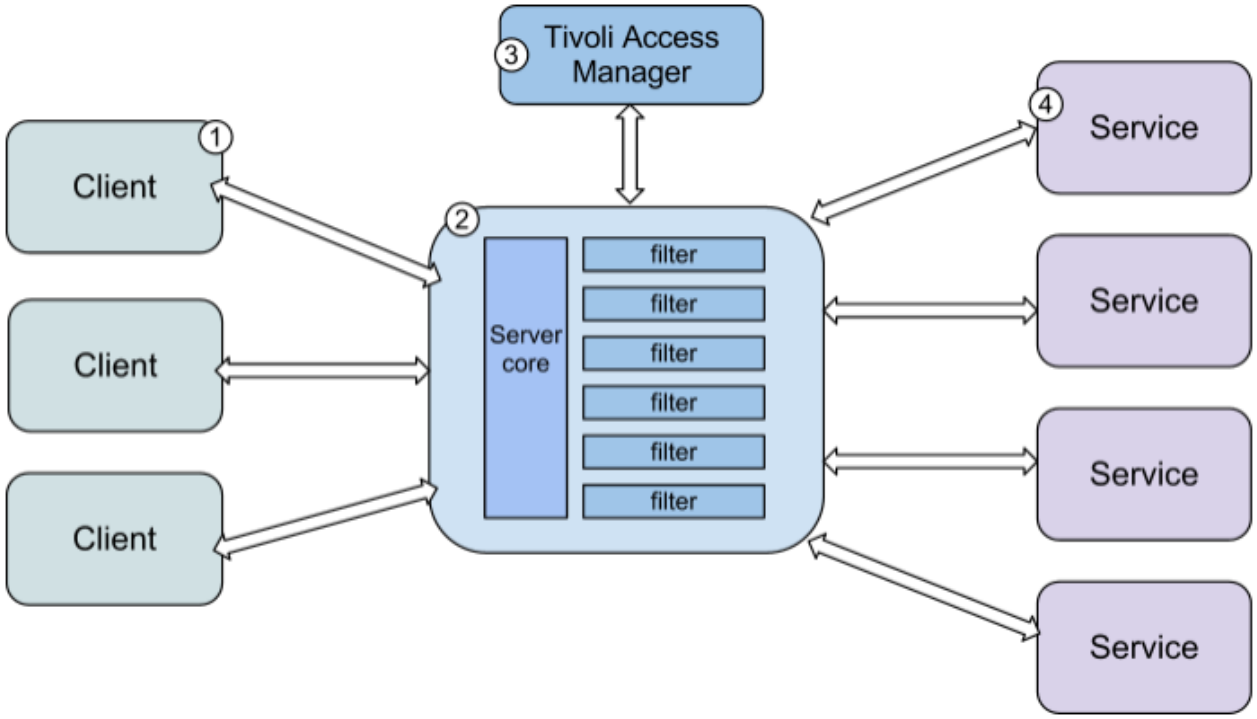
The API Gateway has been built to integrate with Tivoli Access Manager 6.0.

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
Integration with the IBM Tivoli Access Manager requires the IBM Tivoli Access Manager for e-business Authorization C API. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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 available arguments are described as follows:

Argument	Description
`-config`	Creates the configuration files required for the API Gateway to communicate with Tivoli.
`-f`	Name of the main Tivoli configuration file. This file is generated by the command.
`-d`	Name of the directory that is to contain the SSL key file (`.kdb`) for the server. This command generates the key file.
`-n`	Name of the application connecting to Tivoli (the API Gateway).
`-s`	Mode in which the application (the API Gateway) runs. The most likely scenario is that the API Gateway runs remotely.
`-P`	Administrator's password.
`-S`	Password for the API Gateway.
`-r`	Listening port for the API Gateway..
`-h`	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 available arguments are described as follows:

Argument	Description
`-add_replica`	Adds a Tivoli authorization server replica. The API Gateway contacts this server to make authorization decisions.
`-f`	Name of the main Tivoli configuration file. This file is generated by the command.
`-h`	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.

Create 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:

- **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 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.

Export 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. For example, this is useful in a development environment if you wish to share and test configuration with other developers. By exporting configuration data from one API Gateway installation, and importing into another API Gateway installation, you can effectively share your API Gateway configuration in a development environment. This also enables you to manage differences and references between configuration components.

For details on importing configuration data, see *Import API Gateway configuration*.

Note

For details on migrating API Gateway configuration between development, testing, and production environments, see the *Deployment and Promotion Guide*.

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.

Export 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.

Export 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.

Import API Gateway configuration

Overview

You can import configuration data into your API Gateway configuration (for example, policies, certificates, and users). For example, this is useful in a development environment if you wish to share and test configuration with other developers. By exporting configuration data from one API Gateway installation, and importing into another API Gateway installation, you can effectively share your API Gateway configuration in a development environment. This also enables you to manage differences and references between configuration components.

For details on exporting configuration data, see *Export API Gateway configuration*.

Note

For details on migrating API Gateway configuration between development, testing, and production environments, see the *Deployment and Promotion Guide*.

Import 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.

View 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.

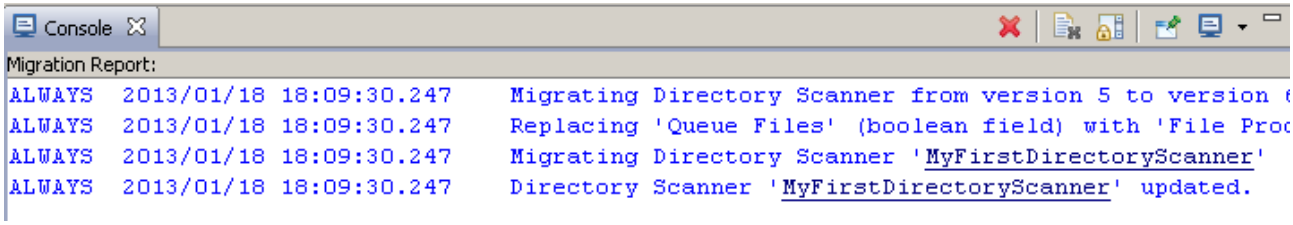
Important

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.

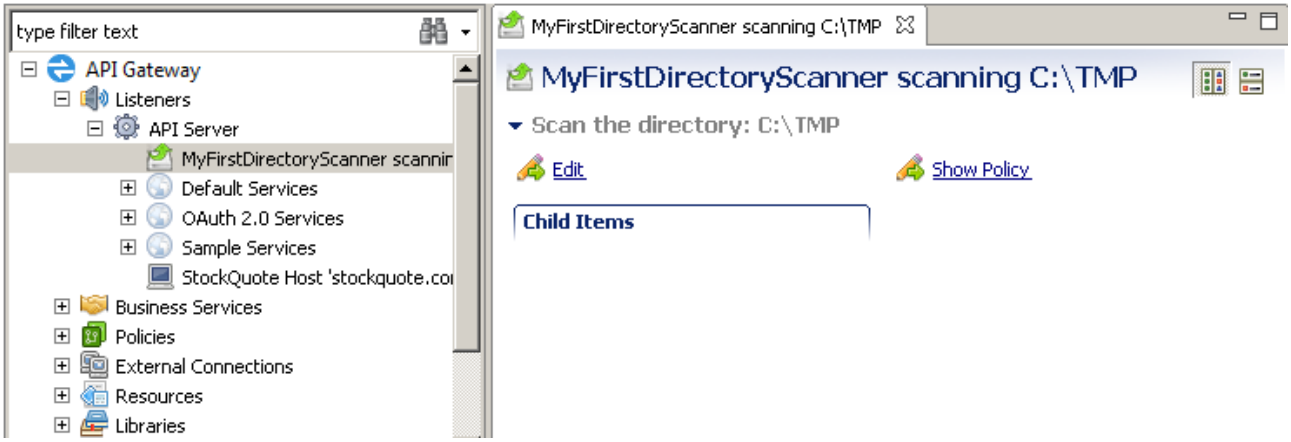
Import configuration from a previous version

If you import configuration created using a previous version of the API Gateway, the configuration is automatically upgraded to the current API Gateway version configuration. This results in the migration of the configuration entities present in the `.xml` file that is being imported.

The **Migration Report** trace console at the bottom of the screen displays the migration report output that is generated when the configuration is upgraded. For example:



The **Migration Report** console also displays links that navigate to the appropriate upgraded configuration entity. For example, the following screen is displayed when the `MyFirstDirectoryScanner` link is clicked:


type filter text
API Gateway
Listeners
API Server
MyFirstDirectoryScanner scannir
Default Services
OAuth 2.0 Services
Sample Services
StockQuote Host 'stockquote.co
Business Services
Policies
External Connections
Resources
Libraries
MyFirstDirectoryScanner scanning C:\TMP
MyFirstDirectoryScanner scanning C:\TMP
Scan the directory: C:\TMP
Edit
Show Policy
Child Items

Configure 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 hosts

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 *Configure 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 *Configure 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 *Configure 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 *Configure a file transfer service* topic explains how to configure the API Gateway as a file transfer service.

Add policy execution scheduling

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 preconfigured 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.

Configure JMS 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 pro-

cessing 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 *JMS Services* topic.

Add Amazon SQS queue listener

The **Amazon SQS Queue Listener** enables you to poll an Amazon SQS queue for messages at a specified rate. When messages are retrieved from the queue, they can be passed to a specified policy for processing. For more details, see the *Configure Amazon SQS queue listener* topic.

Add 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 *Configure an FTP poller* topic.

Add 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 transferred by FTP 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 *Configure directory scanner* topic.

Add 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 *Configure a POP client* topic.

Configure TIBCO

You can configure a TIBCO Rendezvous® Listener. For more details, see *TIBCO Rendezvous listener*.

API Gateway settings

You can configure per-instance global configuration settings by clicking the **Server Settings** node in the Policy Studio tree. For example, these include settings for timeouts, caches, logging, monitoring, security, and so on. For more details on configuring API Gateway instance settings, see *API Gateway Administrator Guide*.

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.

Configure HTTP services

Overview

The API Gateway uses *HTTP Services* to handle traffic from various HTTP-based sources. The available HTTP Services are as follows:

HTTP Interfaces
HTTP interfaces define the ports and IP addresses on which the API Gateway instance listens for incoming requests. You can also add *HTTPS interfaces* to specify SSL certificates to authenticate to clients, and certificates considered trusted for establishing SSL connections with clients.

Relative Path
You can configure *relative paths* so that when a request is received on a specific relative path, the API Gateway instance can map it to a specific policy, or chain of policies. For more details, see *Configure relative paths*.

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. For more details, see *Configure relative paths*

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. For more details, see *Configure relative paths*

Packet Sniffer
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, you could 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 preconfigured 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 preconfigured service groups, you can add new HTTP services groups 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, perform the following steps:

1. Right-click the API Gateway instance, and select **Add HTTP Services**.
2. Enter a name for the group in the **HTTP Services** dialog.
3. To monitor traffic processed by this Service Group using API Gateway Analytics, select **Include in real time monitoring**. For details on configuring the API Gateway to store real-time monitoring data used to generate graphical reports in a web-based interface, see the *API Gateway Administrator Guide*.
4. To enable Cross Origin Resource Sharing (CORS) for this HTTP service, select **CORS** tab, and click the button on the right to select a preconfigured CORS profile. By default, no profile is selected, which means that CORS is disabled. For details on CORS, see *Cross-Origin Resource Sharing*.
5. In the **Select CORS Profile** dialog, if no profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see the section called “Adding a CORS Profile”.

When an HTTP service group has been created, you can configure it with the HTTP services described in the sections that follow.

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 relevant options. For more details, see the *API Gateway Administrator Guide*.

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 in bytes that the API Gateway instance allocates to each request. For more details, see General settings in the *API Gateway Administrator Guide*.

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 *Configure conditions for HTTP interfaces* topic.

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 host names 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 host name 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 host name 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 host name 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 host name 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 host name.

To specify an SNI, perform the following steps:

1. Click the **Add** button to configure a server host name in the **SSL Server Name Identifier (SNI)** dialog.
2. Specify the server host name 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]

Management services

The **Management Services** group exposes a number of services used by the Admin Node Manager and API Gateway Analytics for remote configuration and monitoring. The **Management Services** server process, interfaces, and policies are displayed in the Policy Studio tree. The **Management Services** policy container is displayed in the tree under the **Policies** node. The **Management Services** HTTP interfaces are displayed under the **Listeners** node under the server instance.

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 API Gateway Analytics exposes all its management services on port `8040`. For more details, see the section called “Change 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.

Change 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. Under the **Listeners** node in the Policy Studio tree, right-click the **Management Services** HTTP Interface, and select **Edit**.
2. Specify an updated value in the **Port** field (for example, `8091`), and click **OK**.
3. Click the **Deploy** button in the Policy Studio toolbar, or press F6 to deploy the update.
4. Restart Policy Studio. You must restart Policy Studio when Management Services are updated.
5. 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 API Gateway Analytics only. You should only modify **Management Services** under strict advice and supervision from the Oracle Support team.

Configure relative paths

Overview

A *relative path* binds policies to a specific relative path location (for example `/test/path`). When the API Gateway receives a request on the specified relative path, it invokes the specified policy or policy chain. This topic explains how to configure relative path resolvers. For details on how to configure policies, see Chapter 2, Managing Policies.

You can use a *Static Content Provider* or *Static File Provider* to serve static HTTP content or files from a particular directory. In this case, the API Gateway instance acts as a Web server. The API Gateway instance can also act as a *Servlet Application* container, which enables you to drop servlets into the HTTP Services configuration. This should only be used by developers with specific requirements under strict advice from the Oracle Support team.

Relative paths can have nested child resolvers of the following type:

- Relative path
- Static content provider
- Static file provider
- Servlet application

This topic explains how to use the Policy Studio to configure each of these relative path resolver types. For details on configuring HTTP Services Groups and HTTP Interfaces, see *Configure HTTP services*.

Configure a relative path

To configure a relative path for a specific HTTP Service Group (for example, **Default Services**), perform the following steps:

1. In the Policy Studio tree, select **Listeners** > **API Gateway** -> **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** > **Relative Path**. You can also click the **Add** button in the screen on the right.
3. In the **Configure Relative Path** dialog, specify whether to enable listening on the specified path using the **Enable this path resolver** setting, which is set by default.

Alternatively, when editing a policy, you can click the **Add Relative Path** button at the bottom of the policy canvas beside the **Context** drop-down list.

The next four sections explain how to configure the settings on the **Configure Relative Path** dialog.

Policies settings

Use the **Policies** tab to 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/path`) for the selected HTTP Services 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 *Configure global policies*.
Path Specific Policy	To configure a path-specific policy, select this setting, and browse 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 *Configure 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.

Logging Level
You can configure the following 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 *Set transaction log level and log message* topic.

Payload Level
You can configure the following 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 *Log message payload*.

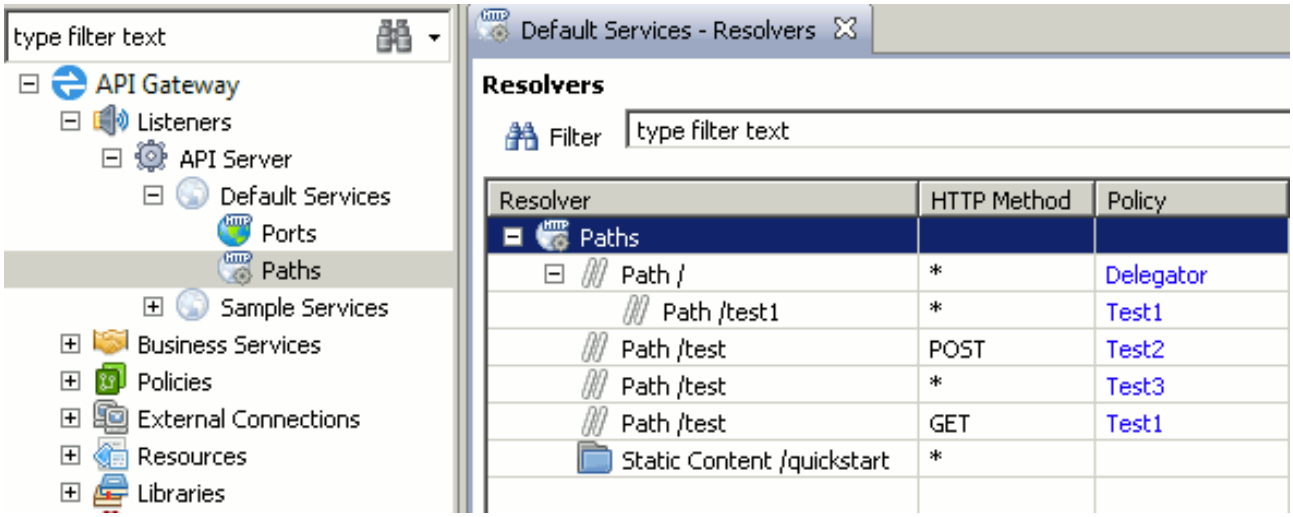
Access Log
Select the **Include in server access log records** setting if you wish to add this relative path to the API Gateway Access

Log. This enables the Access Log at the service level. This setting is not selected by default. For more details, see the *API Gateway Administrator Guide*.

HTTP method settings

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 also 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:



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.

For details on the subpath shown in this example (`Path/Test1`), see the section called “Nested relative paths”.

Advanced settings

On the **Advanced** tab, select whether to resolve this relative path using an **Exact path match**. The default behavior is to use a longest path match (explained in the next section). This setting enables you to further restrict the match to an exact path match (for example, `test1/helloService`). This setting is not selected by default.

CORS settings

On the **CORS** tab, you can configure settings for Cross Origin Resource Sharing (CORS). For details on CORS, see *Cross-Origin Resource Sharing*. By default, the CORS profile set at the HTTP service level is used for all child resolvers of the HTTP service. However, you can override this at the relative path level as follows:

1. In the **CORS Usage** field, select **Override CORS using the following profile**. By default, no CORS profile is selected, and the parent settings are used.

Note

Relative paths can act as HTTP Services, and can accommodate child resolvers. This means that when a Relative Path has children, and has a CORS profile configured, by default, the children use the parent profile (unless a child overrides it).

2. In the **CORS Profile** field, click the button on the right to select a preconfigured CORS Profile.
3. If no CORS Profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see the section called “Adding a CORS Profile”.

Nested relative paths

Using the example shown in the section called “HTTP method settings”, when you have a path `/` that has a child subpath (in this case `/test1`), the following occurs when a request arrives at the API Gateway:

1. Incoming request with path `/test1` is received.
2. Request is resolved against `/` (using the longest path match algorithm).
3. API Gateway checks if there are any children (in this case, there is `/test1`).
4. API Gateway checks if any children can process the request (`/test1` is a match).
5. Path resolution is successful.

From a policy execution point of view, when the request is being processed, each policy associated with a matching path is executed. In the above example, both `/` and `/test1` make up the match. This means that the policy associated with `/` is executed first, and if that passes, the policy associated with `/test1` is executed. The parent policy uses the **Call Internal Service** filter, which enables child resolvers to be invoked.

Nested paths are generally used as a protection mechanism. For example, a system might be configured with `/` as the only root path, with a number of children. `/` could have a Role-Based Access Control (RBAC) policy associated with it protecting all children. If the RBAC policy succeeds, access is granted, and the child policy is executed. If it fails, access is denied. This mechanism is implemented in the Admin Node Manager. You can view this in Policy Studio by opening the Admin Node Manager configuration.

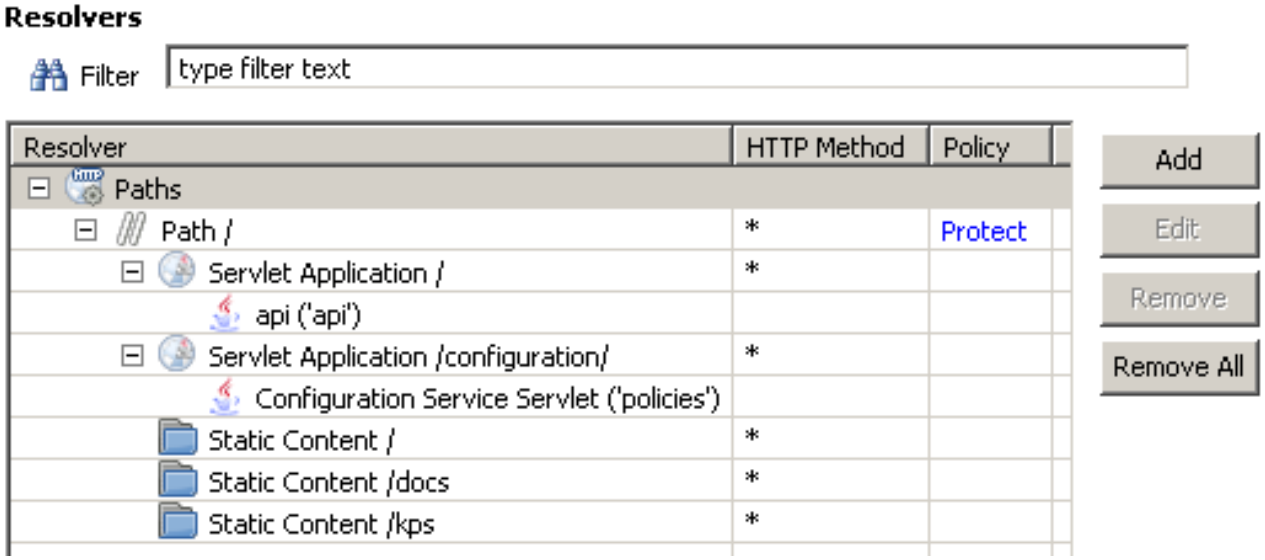
Note

The difference between `/` having a child `/test1`, and just having a root relative path of `/test1` is due to path resolution and policy execution. If you wish to protect `/test1` using RBAC, or run a prerequisite policy prior to running the policy associated with `/test1`, you should use subpaths. You could also achieve this using a **Policy Shortcut** filter in the policy associated with `/test1` (as a root path). This may be sufficient for a small number of root policies. But when a large number of policies need to be protected, subpaths are a more elegant solution. Children no longer need to have a **Policy Shortcut** filter with the policy associated with the parent path as protector.

Adding a Nested Relative Path
To add a child node to a relative path, right-click the appropriate relative path node in the **Resolvers** screen, and select the node type (for example, **Add** > **Relative Path**). You can add child resolver nodes of the following type:

- Relative path
- Static content provider
- Servlet application

In the following example, there is one main relative path configured to `/`, which calls the **Protect** policy. Content is served by the underlying Servlet Application and Static Content resolvers only when the **Protect** policy succeeds:



Important

The parent policy (in this case, **Protect**) must use the **Call Internal Service** filter. This acts as a loopback and enables child resolvers to be invoked. When this prerequisite is met, you can add nested relative paths as required.

Order of Resolution
The order of resolution for nested relative paths is to first resolve at the parent level. If resolution is successful, and there are children present, then attempt to resolve at the child level. There is no precedence between resolver node types (relative path, static content provider, and servlet application). Path resolution is performed using the longest path match algorithm by default, regardless of whether nested subpaths are used. For details on configuring an exact path match, see the section called “Advanced settings”.

Example Nested Path Resolution
Using the example relative path in the screen above, consider the following inbound requests to the Node Manager:

```
https://testpc:8090/api/deployment/domain/deployments
https://testpc:8090/common/themes/blue/images/server_icon.png
```

The `/api/deployment/domain/deployments` request is resolved as follows:

- Matches the root relative path `/` as a longest path match
- Matches Servlet Application `/` as a longest path match (1 char)
- Matches Servlet `/api` as a longest path match (4 char)
- Matches Static Content `/` as a longest path match (1 char)
- Does not match Servlet Application `/configuration`
- Does not match Static Content `/docs`
- Does not match Static Content `/kps`

In this example, there are two matches, the `api` Servlet under the Servlet Application on `/`, and the Static Content on `/`. Because the API Gateway uses the longest path match, the `api` Servlet wins, and the request is routed to that resolver. There is no precedence between resolvers, all resolvers are queried for a match.

The `/common/themes/blue/images/server_icon.png` request is resolved as follows:

- Matches the root relative path `/` as a longest path match
- Matches servlet application `/` as a longest path match (1 char)
- Does not match servlet `/api`
- Matches static content `/` as a longest path match (1 char)
- Does not match servlet application `/configuration`
- Does not match static content `/docs`
- Does not match static content `/kps`

In this example, there is only one match, the Static Content on `/`. In this case, the Servlet Application on `/` is not considered because none of its children can resolve the request path.

Note

If there are two resolver matches, and each matches on the same path length, the last resolver visited during path resolution is used, in the order in which the resolver was read and loaded from the configuration.

Static content providers

A *Static Content Provider* can be used 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 way. For example, you can configure a Static Content Provider to serve content from the `c:\docs` folder on Windows when it receives requests on the relative path `/docs`.

Adding a Static Content Provider
To add a Static Content Provider to an HTTP Services group (for example, **Default Services**), perform the following steps:

1. Select **Listeners** > **API Gateway** -> **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** -> **Static Content Provider**.
3. 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 instead of 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 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. You can turn this 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. For more details, see the section called “HTTP method settings”.

Static file providers

A *Static File Provider* can be used with an HTTP Interface to serve a static file from a specified directory. A relative path is associated with each Static File Provider so that requests received on this path are dispatched directly to the file provider and are not mapped to a policy in the usual way. For example, you can configure a Static File Provider to serve `c:\my_brand\favicon.ico` on Windows when it receives requests on the `/favicon.ico` relative path.

Adding a Static File Provider
To add a Static File Provider to an HTTP Services group (for example, **Default Services**), perform the following steps:

1. Select **Listeners** > **API Gateway** > **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** > **Static File Provider**.
3. Complete the following fields on the **General** tab:

Relative Path:
Enter the path that you want to receive requests for the static file on (for example, `/favicon.ico`).

File:
Enter or browse to the location of static file that you want to serve (for example, `$VDISTDIR/webapps/emc/favicon.ico`, where `$VDISTDIR` specifies the directory in which the API Gateway is installed).

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method for the static file. The default is `GET`, which means that only HTTP GET calls are accepted. You can override the default behavior, and select a different HTTP method for this resolver from the list. For more details, see the section called “HTTP method settings”.

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.

When editing Admin Node Manager or API Gateway Analytics configuration, there are some default Servlet Applications available under the **Management Services** group. By default, this HTTP 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 the default Servlet Applications may prevent the API Gateway from functioning correctly. You should only delete default Servlet Applications under strict supervision of the Oracle Support team.

Adding a Servlet Application
To add a Servlet Application to an HTTP Services Group (for example, **Default Services**), perform the following steps:

1. Select **Listeners** > **API Gateway** -> **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** -> **Servlet Application**.
3. Complete the following fields on the **General** tab:

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. For more details, see the section called “HTTP method settings”.

Adding a Servlet
The new Servlet Application now appears in the **Resolvers** screen. 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. You can add this class to the server runtime by adding the JAR, class file, or package hierarchy to the `[VDISTDIR]/ext/lib` folder. `VDISTDIR` is your API Gateway distribution directory, which is the location where the API Gateway is installed.

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.

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 in the **Web Service Repository**, a new web service resolver node is created for each imported web service under the **Paths** for the relevant HTTP services group (for example, Default Services). You can edit the web service resolver settings by right-clicking its 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.

Policies:
On the **Policies** 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 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 *Configure global policies*.
Path Specific Policy	To configure a path-specific policy, select this setting, and browse 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 *Configure 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 Settings:
The **Audit Settings** 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 relative path. For more details, see the section called "Audit 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 from the list. The **HTTP Method** settings are the same as those already described for the relative path. For more details, see the section called "HTTP method settings".

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.

Configure virtual hosts

Overview

A virtual host is a server, or pool of servers, that can host multiple domain names (for example, `company1.example.com` and `company2.example.com`). This enables you to run more than one website, or set of REST APIs, on a single host machine (for example, `192.0.2.11`). Each domain name can have its own hostname, paths, APIs, and so on. For example:

```
https://amex.api.example.com:8080/api/v1/test
https://visa.api.example.com:8080/api/v2/test
https://mastercard.api.example.com:8080/api/v2/test
```

The API Gateway implements *name-based* virtual hosting, in which the client HTTP `Host` header is used as the routing criteria during path resolution (for example, `Host amex.api.example.com`). This means that you can have multiple domains running on the same hardware (IP address), while this is not apparent to the client or end user.

For example, the following URL invokes the `visa.api.example.com` virtual host:

```
https://visa.api.example.com/api/v1/test
```

This results in the following message at runtime:

```
POST /api/v1/test HTTP/1.1
Host: visa.api.example.com
Content-Type: application/x-www-form-urlencoded

client_id=SampleConfidentialApp&client_secret=......
```

Important

To support name-based virtual hosts, you must first ensure that your Domain Name System (DNS) server has been updated to map each hostname to the correct IP address (for example, `*.example.com` is mappedd to `192.0.2.11`). For more details, see the topic on "Configuring a DNS service with wildcards for virtual hosting" in the *API Gateway Administrator Guide*.

When your DNS server has been updated to map each hostname to the correct IP address, you can then configure the API Gateway for virtual hosting.

Configure virtual hosts for HTTP services

You can configure virtual hosts at the HTTP service level. This means that these settings are applied to the HTTP service and to any child resolvers. To configure a virtual host at the HTTP service level, perform the following steps:

1. In the Policy Studio tree, select an HTTP service (for example, **Listeners** > **API Gateway** > **Default Services** > **Virtual Hosts**).
2. Right-click, and select **Add a Virtual Host**.
3. Configure the following settings in the **Virtual Host** dialog:
   - **Name**:
     Enter a unique name of the virtual host.
   - **Enabled**:
     Select whether the virtual host processing is enabled. This is enabled by default.
   - **Hosts**:

Specify the list of domains that you wish to host under this HTTP service. To add a host, click **Add** at the bottom right, and enter the domain name (for example `amex.api.example.com`).

You can also specify domain names using wildcards already configured in your DNS (for example `*.example.com:/8080` or `visa.api.example.com.*`). For details on configuring DNS wildcards, see the *API Gateway Administrator Guide*.

Configure child resolvers

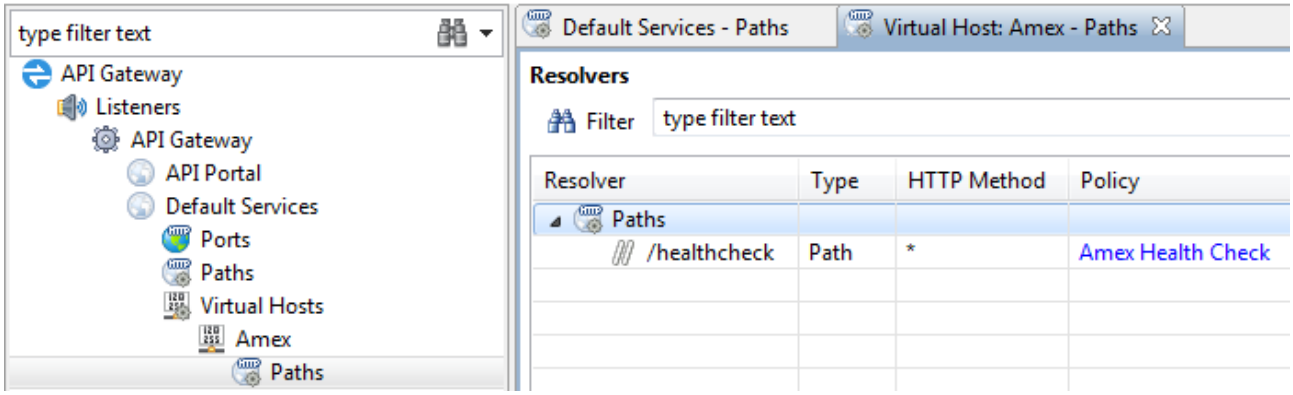
When you have configured a virtual host at the HTTP service level, you can also configure the following child resolvers:

- Relative path
- Static content provider
- Static file provider
- Servlet application

To configure a child resolver, perform the following steps:

1. In the Policy Studio tree, select the **Paths** node under the virtual host, and right-click to add a resolver (for example, **Add relative path**).
2. In the **Resolve path to policies** dialog, click the browse button beside the **Path Specific Policy** field, and select a policy to run on this path.

For example, if an inbound request to `/healthcheck` matches on `amex.api.*`, the **Amex Health Check** policy is executed. Otherwise, the global **Health Check** policy for the HTTP service is executed (in this case, **Default Services**).



Configure 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.

Add 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 an API Gateway instance node (for example, the default **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 sec-

	tion 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 web-based API Gateway Manager and API Gateway Analytics monitoring consoles.
8. **Click OK**. This creates a tree node for the SMTP service under the selected instance in the **Services** tree.

Add 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 instance.
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 *Configure HTTP services* topic.
4. **Click OK**.

Configure 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.

Add 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 instance.
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.

Add 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 instance.
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.

Message Attribute	Description
authentica- tion.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.

Add 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 instance.
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.

Add 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 instance.
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`.

Add 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 instance.
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
8	Yes	Yes	Yes	Yes	No	Yes

These authentication scenarios are described as follows:

1. No authentication user name 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 user name 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 user name and password are specified so the API Gateway authenticates with the server. The mail server advertises AUTH, but because a user name 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 user name 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 user name 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.

Multipart MIME content, generally used for sending attachments in SMTP, is also supported. Each separate body in the multipart 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 **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 vian 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 vian 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
```

Configure a 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.

Tip

For details on how to use the API Gateway to poll a remote file server, to query and retrieve files to be processed, see *Configure an FTP poller*.

General settings

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 settings

For all file transfer protocols, you must specify a **File Upload** policy to be invoked with the file data. 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`, where `VINSTDIR` is the location of the running API Gateway instance. 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**.

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 in 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
You must specify a **File Upload** policy to be invoked with the raw file data. 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 settings

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 *Manage certificates and keys*. Alternatively, you can specify a certificate to bind to at runtime using an environment variable selector (for example, `${env.serverCertificate}`). For details on setting external environment variables for API Gateway instances, see the *API Gateway Deployment and Promotion Guide*.

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 *Manage certificates and keys*.

Command settings

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-project/ftpserver_commands.html

Access control settings

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.

Message settings

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 settings

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.

Audit settings

The **Audit Settings** tab enables you to configure the logging level for the file transfer service, and to configure when message payloads are logged.

Logging Level
You can configure the following settings on all filters executed on the file transfer service:

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 *Set transaction log level and log message*.

Payload Level
You can configure the following settings for the file transfer service payload:

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 *Log message payload*.

Include in real time monitoring
Select whether to monitor traffic for the file transfer service. This means that traffic for this service is monitored in the API Gateway Manager and API Gateway Analytics web-based interfaces. For more details, see the *API Gateway Administrator Guide*. This setting is selected by default.

Traffic monitor settings

The **Traffic Monitor** tab enables you to configure traffic monitoring settings for the file transfer service. To override the system-level traffic monitoring settings, select **Override system-level settings**, and configure the relevant options. For more details, see the *API Gateway Administrator Guide*.

Policy execution scheduling

Overview

You can configure a policy execution scheduler at the level of the API Gateway instance. 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 field, `0/15` means minutes 0, 15, 30, and 45, while `5/15` means minutes `5`, `20`, `35`, and `50`). Spe-

Special Character	Description
	cifying * 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, 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`).

Add 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.

Add 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 instance node (for example, **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**.

Configure Amazon SQS queue listener

Overview

Amazon Simple Queue Service (SQS) is a hosted message queuing service for distributing messages amongst machines. You can configure API Gateway to poll an Amazon SQS queue at a set rate. Any message found on the SQS queue in this interval can be sent to a policy for processing.

For more information on Amazon SQS, go to http://aws.amazon.com/sqs/.

To add a new Amazon SQS queue listener, in the Policy Studio tree view, under the **Listeners** node, right-click the instance name (for example, `API Gateway`), and select **Amazon Web Services > Add SQS Queue Listener**.

General settings

You can configure the following general settings for the Amazon SQS queue listener:

Name:
Enter a suitable name for this SQS listener.

AWS settings

AWS Credential:
Click the browse button to select your AWS security credentials (API key and secret) to be used by API Gateway when connecting to Amazon SQS.

Region:
Select the region appropriate for your deployment. You can choose from the following options:

- US East (Northern Virginia)
- US West (Oregon)
- US West (Northern California)
- EU (Ireland)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- South America (Sao Paulo)
- US GovCloud

Client settings:
Click the browse button to select the AWS client configuration to be used by API Gateway when connecting to Amazon SQS. For more information on configuring client settings, see the section called “Configure AWS client settings”.

Poll settings

Poll the queue <queueName> every <pollRate> milliseconds:
Enter the name of the queue to be polled and the rate at which the queue is to be polled, in milliseconds. The default queue name is `requestQueue` and the default poll rate is `10000` milliseconds (10 seconds).

Call policy:
Click the browse button to select a policy to send the message to for processing. When a message has been consumed from the queue it is passed to the selected policy for processing.

Process the retrieved messages as body with following Content Type:

Select this option to create a body from the message, and enter a content type. The default content type is `text/xml`.

Store the content of the retrieved messages in the following attribute:
Select this option to store the content of the message in an attribute, and enter the attribute name. The default attribute name is `sqs.body`.

Delete message on completion:
Select this option to delete the message from the queue when processing is complete. This option is selected by default.

Maximum number of messages to read from queue:
Enter the maximum number of messages to return. Amazon SQS never returns more messages than this value but it might return less. The default value is `10`.

Visibility timeout for other consumers:
Enter the duration (in seconds) that the received messages are hidden from subsequent retrieve requests after being retrieved by API Gateway. The default value is `1000` seconds.

Response settings

Write a response message:
Select this option to write the contents of an attribute to a response queue. This option is not selected by default.

Response queue name:
If you selected the **Write a response message** option, enter the name of the response queue in this field. The default name is `responseQueue`.

Use content body as the response:
Select this option to use the content body as the response.

Use the following attribute as the response:
Select this option to use an attribute as the response, and enter an attribute selector for the attribute containing the response in this field (for example, `${sqs.body}`). For more details on selectors, see *Selecting configuration values at runtime*.

Configure AWS client settings

API Gateway is a client of AWS, and as such you can define a client profile by which API Gateway connects to AWS.

When you click the browse button on the **Client settings** field for an Amazon SQS queue listener, a **Send to Amazon SQS** filter, an **Upload to Amazon S3** filter, or an Amazon Simple Notification Service (SNS) alert destination, you can edit the configuration of the AWS client profile.

To edit the configuration, right-click the **Default AWS Client Configuration** and select **Edit**. Configure the following settings on the dialog:

Name:
Enter a suitable name for this client configuration.

Connection settings

Maximum number of open HTTP connections:
Enter the maximum number of open HTTP connections. The default value is `50`.

Socket timeout:
Enter the amount of time to wait (in milliseconds) for data to be transferred over an established, open connection before the connection is timed out. The default value is `50000` milliseconds. A value of 0 means infinity, and is not recommended.

Connection timeout:

Enter the amount of time to wait (in milliseconds) when initially establishing a connection before giving up and timing out. The default value is `50000` milliseconds. A value of 0 means infinity, and is not recommended.

Maximum number of retries:
Enter the maximum number of retry attempts for failed requests (requests that can be retried). The default value is 3.

User agent:
Enter the HTTP user agent header to send with all requests.

Protocol:
Select the protocol (HTTP or HTTPS) to use when connecting to AWS.

Proxy settings

You can optionally configure the following proxy settings:

Proxy Host:
Enter the proxy host to connect through.

Proxy port:
Enter the port on the proxy host to connect through.

User name:
Enter the user name to use when connecting through a proxy.

Password:
Enter the password to use when connecting through a proxy.

NTLM Proxy support:
To configure Windows NT LAN Manager (NTLM) proxy support, enter the Windows domain name in the **Windows domain** field and enter the Windows workstation name in the **Windows workstation name** field.

Advanced settings

You can optionally configure these settings to tune low level TCP parameters to try and improve performance.

Note

These settings are for advanced users only.

Size hint (in bytes) for the low level TCP send buffer:
Enter the size hint (in bytes) for the low level TCP send buffer.

Size hint (in bytes) for the low level TCP receive buffer:
Enter the size hint (in bytes) for the low level TCP receive buffer.

Further information

For more detailed information on Amazon Web Services integration, see the *AWS Integration Guide* available from Axway Support.

Configure an 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 instance name (for example, `API Gateway`), and select **FTP Poller** > **Add**. This topic describes how to configure the fields on the **FTP Poller Settings** dialog.

Tip

For details on how to configure the API Gateway to act as a file transfer service that listens on a port for remote clients, see *Configure a file transfer service*.

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 user name 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.

Limit the number of files to be processed:
Select this option to limit the number of files that the FTP poller will process on each poll of the FTP server. This option is not selected by default.

Specify the max number of files to be processed:
Enter the maximum number of files to be processed on each poll of the FTP server. The default is `100`.

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 *Manage 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`).

Configure directory scanner

Overview

The Directory Scanner enables you to scan a specified directory on the file system for files containing messages (for example, in XML or JSON format). When the messages have been read, they can be passed into the core message pipeline, where the full range of message processing filters can act on them. The Directory Scanner is typically used in cases where an external application is dropping files (perhaps by FTP) on to the file system so that they can be validated, modified, and potentially routed on over HTTP or JMS. Alternatively, they can 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. For example, instead of making drastic changes to the legacy system by adding an HTTP engine, the API Gateway can pull the files from the file system, and route them on over HTTP to another back-end system. The added benefit is that messages can be exposed to the full range of message processing filters in 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 API Gateway instance (for example, `API Gateway`), and select the **Directory Scanner** > **Add** menu option. This topic describes how to configure the fields on the **Directory Scanner Settings** dialog.

General settings

Configure the following setting:

Name:
Enter a name for this Directory Scanner. This setting is required.

Input settings

The fields in this section configure where to scan for files, what files to scan, and when to scan.

Input Directory:
Enter or browse to the input directory that the API Gateway scans for files. This setting is required.

Match File:
Select one of the following options to specify how inbound files are filtered:

- **By Name**:
  You can specify a comma-separated list of specific names, wildcarded names, or leave this field blank to accept all files by name. For example, a value of `"*.xml,*.xsl"` only accepts XML and XSL files in the input directory.
- **By Extension**:
  You can specify a comma-separated list of file extensions (excluding the `.` character), or leave this field blank to accept all files by extension. For example, `"xml,xsl"` only accepts XML and XSL files in the input directory.
- **By Pattern (regex)**:
  If you wish to scan only for files based on some pattern, you can specify this 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 for these files only. For example, the following regular expression scans only for 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 (ms):
Specifies the amount of time (in milliseconds) between each scan of the **Input Directory** for new files. Defaults to `60000`.

Processing settings

The fields configured in this section determine what processing is performed on the input files, and where files are placed before and after processing.

Processing Directory:
Enter or browse to the directory to which the input file is copied prior to processing. This field is optional. If it is not specified, the input file is moved to the processing directory specified by the processing policy.

Response Directory:
Enter or browse to the directory to which the response file is copied. This field is optional. If this is not specified, the response file is not written to disk.

Processing Policy
Select the policy executed on the input file. For example, the policy could perform message validation, routing, virus checking, or XSLT transformation. This setting is required.

File Type:
Specifies how the input file is interpreted. Select one of the following options:

- **Raw**:
  Assumes a content-type of `application/octet-stream`. This is the default.
- **Treat as HTTP Message (including headers)**:
  Assumes the inbound file contains an HTTP request (optionally with HTTP headers).
- **Infer content-type from extension**:
  Performs a lookup on configured MIME/DIME types to determine the content-type of the file based on its extension.
- **Use Content-type**:
  Enables you to specify a content-type in the textbox.

Sort Files:
Select whether files are sorted, and select one of the following sort types from the list:

- `Date last modified`
- `Extension`
- `Extension (case-sensitive)`
- `Name`
- `Name (case-sensitive)`
- `Size`

The default is to sort by `Name`.

Select one of the following sort directions:

- `Ascending`
- `Descending`

The default is `Ascending`.

File Processing:
Select whether to **Process files in parallel** or **Process files in sequence** (queued). This option is useful if you need to process very large files, or process files in a particular order (when used in conjunction with the **Sort Files** option).

On completion settings

You can specify what to do when the file processing has completed. Select one of the following options:

- **Do Nothing**:
  The input file remains in the **Input Directory** or **Processing Directory**. This is the default.
- **Delete Input File**:
  The input file is deleted from the **Input Directory** or **Processing Directory**.
- **Move Input File**:
  The input file is moved (archived) to the directory specified in the **To Directory** field. You can also specify an optional **File Prefix** or **File Suffix** for the archived file.

Traffic monitor settings

The **Traffic Monitor** tab enables you to configure traffic monitoring settings for the directory scanner. To override the system-level traffic monitoring settings, select **Override system-level settings**, and configure the relevant options. For more details, see the *API Gateway Administrator Guide*.

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 API Gateway instance, selecting the **Add HTTP Services** menu option, and entering `Packet Sniffer Group` on the **HTTP Services** dialog.

You 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.

Configure 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-instance* by right-clicking the API Gateway instance 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 Alias:
The human readable alias name for the remote host (for example, `StockQuote Host`). This setting is required.

Host Name:
The host name or IP address of the Remote Host to connect to (for example `stockquote.com`). 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 and Load Balancing** tab, which override the default network DNS server mappings, if configured. This setting is required.

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 instance 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.

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 avail-

able 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*.

Configure 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 *Configure HTTP watchdog*. For more information on adding Conditions to an HTTP Interface, see *Configure conditions for HTTP interfaces*.

Configure WebSocket connections

WebSocket protocol overview

The WebSocket protocol provides an extension to the HTTP 1.1 protocol to establish persistent, bidirectional communication between a client and a server.

The WebSocket protocol can be summarized as follows:

1. To establish a communication channel between a client and a server, the client needs to send an HTTP `Upgrade` request to the server. This is known as the WebSocket protocol handshake.
2. If the server is capable and willing to upgrade the connection, it sends a HTTP `101` response to the requesting client. At this point the handshake is considered successful and the connection between the server and the client is upgraded to the WebSocket protocol.

Note

As soon as the client receives the HTTP `101` response the connection is no longer considered an HTTP connection.

3. Messages can now flow bidirectionally between the server and the client over the WebSocket connection.
4. Any participant in the data exchange can request the WebSocket connection be terminated by sending a `Close` request to the other participant.

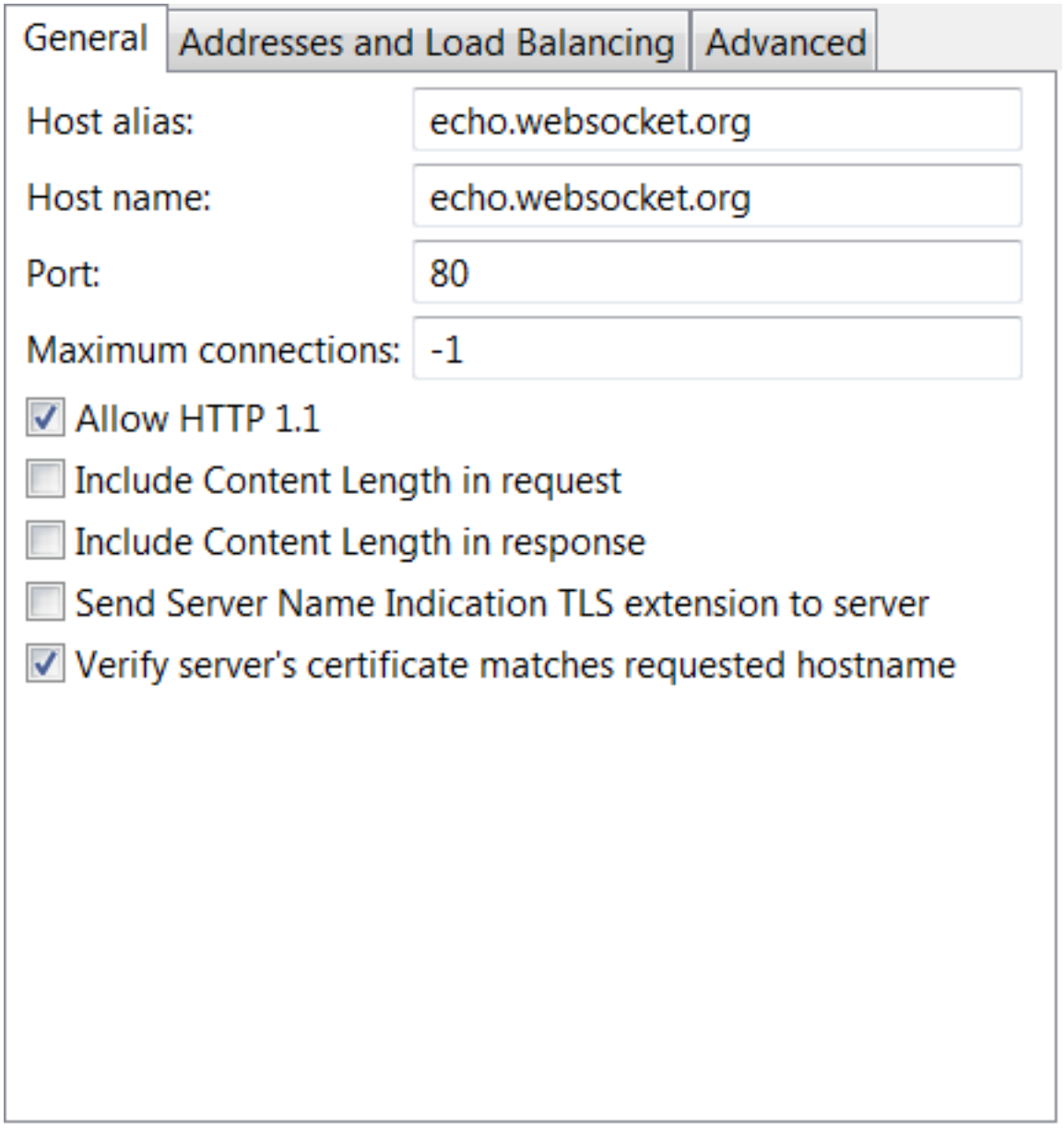
For a detailed description of the protocol, see RFC 6455 [http://tools.ietf.org/html/rfc6455].

Configure a WebSocket connection

API Gateway can act as a WebSocket proxy, whereby it is deployed in front of a WebSocket capable web server (for example, Jetty or Apache Tomcat) and provides governance (security, monitoring, and so on) on the WebSocket traffic flowing between the client, API Gateway, and the web server.

Each WebSocket server that API Gateway is routing to must be defined as an HTTP/1.1 remote host. To add a remote host to an API Gateway instance, right-click the API Gateway instance in the Policy Studio tree view, and select **Add Remote Host**. For more information, see *Configure remote host settings*.

For example, if API Gateway is proxying WebSocket traffic to the URL `http://echo.websocket.org/`, then you would configure a remote host as shown in the following figure:



To enable API Gateway to accept an HTTP `Upgrade` request from a client you must add a WebSocket handler to your API Gateway configuration and configure it with the HTTP path that the upgrade can be expected on.

To add a WebSocket handler, follow these steps:

1. In the Policy Studio tree, select a list of relative paths (for example, **Listeners > API Gateway > Default Services > Paths**).
2. In the **Resolvers** window on the right, click **Add > WebSocket** to display the **WebSocket configuration** dialog.

WebSocket configuration settings

Configure the following fields on the **WebSocket configuration** dialog:

Enable this path resolver:
Select or deselect the check box to enable or disable the WebSocket handler. It is enabled by default.

Policies settings

You can assign specific policies on this tab to specific URIs that define the WebSocket endpoints. For example, you might need to handle frames being exchanged between a client and `ws://example.org/echo` differently to frames being exchanged between a client and `ws://example.org/voip`.

Note

In the above scenario, different sets of policies need to be defined for each URI (`/echo` and `/voip`). This requires different relative paths. For more information on relative paths, see *Configure relative paths*.

When a request arrives that matches the path:
Enter the path on which WebSocket connections are to be accepted. This defines the URI of the WebSocket endpoint. A relative path resolver for this path must already exist.

Default MIME type for message body:
Enter the MIME type of the messages. The default is `application/json`.

When messages are flowing bidirectionally between a WebSocket server and client, they are no longer HTTP messages and as such they do not contain a Content type header. For API Gateway to process the content of the message it needs to know what type of content the message is. This field enables you to specify the type of message being exchanged between the server and the client.

On Upgrade request from client:
Click the browse button to select a policy to be used by API Gateway when an `Upgrade` request is received.

This policy is executed when a connection is being upgraded from HTTP to WebSocket. For example, you might statically route all WebSocket requests to `ws://example.org` to `wss://example.com` by using a policy. A policy for an HTTP connection must be provided and this policy must provide a mechanism to connect to the remote server.

For example, to route all requests to `ws://example.org` to `wss://example.com`, you can use the **Static Router** filter. Similarly, for dynamic routing you can use the **Dynamic Router** or **Connect to URL** filters. In the latter case, the remote host that the client is attempting to connect to can be extracted from the Host header of the request. This value can then be passed to the **Connect to URL** filter and used by that filter to establish a connection to the remote host.

The on upgrade policy is also a good place to perform authentication and authorization of the requesting client.

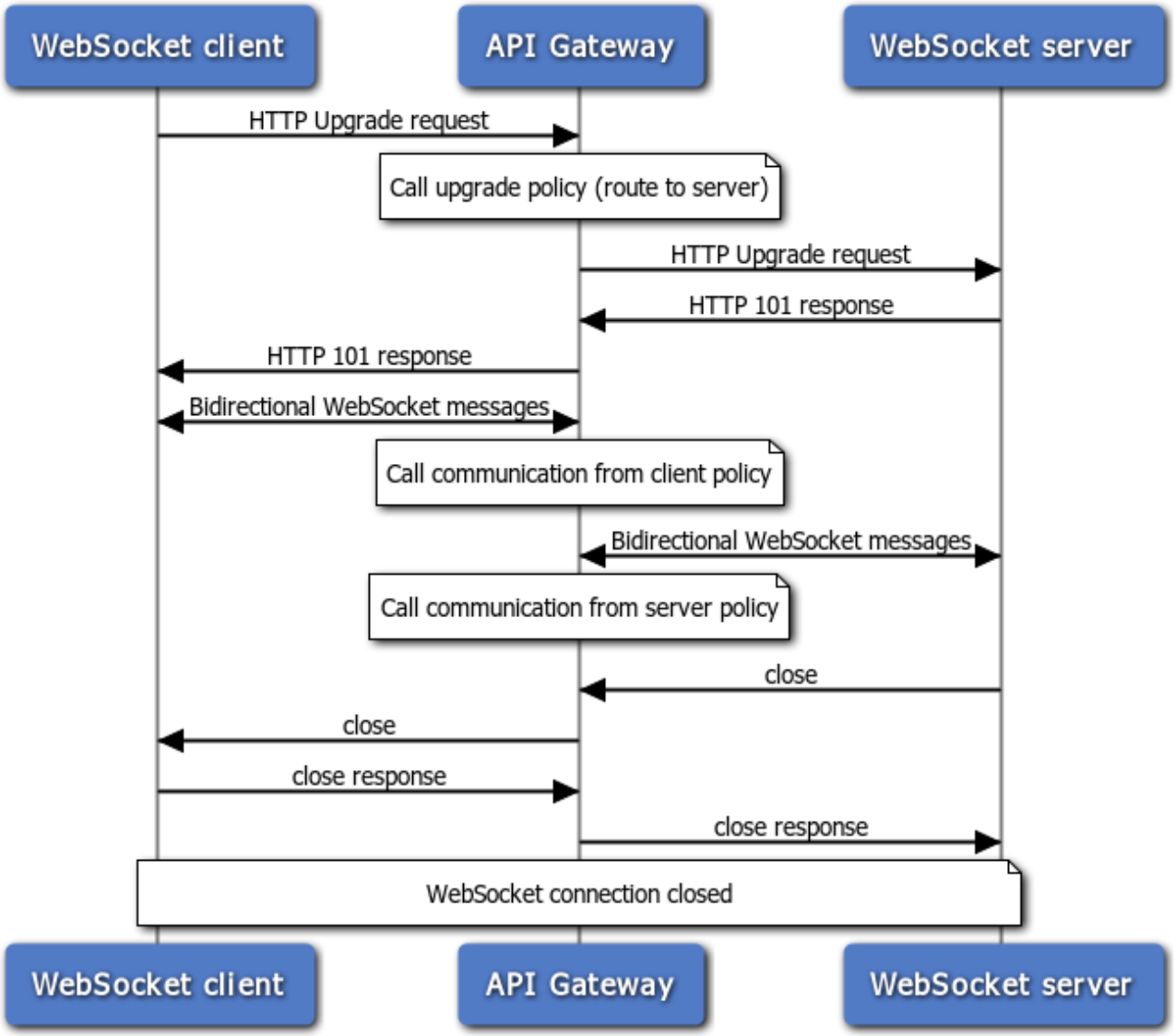
On WebSocket communication from client:
Click the browse button to select a policy to be used by API Gateway when frames are received from the WebSocket client.

On WebSocket communication from server:
Click the browse button to select a policy to be used by API Gateway when frames are received from the WebSocket server.

Note

After a successful upgrade request, the context used to upgrade the connection from HTTP to WebSocket protocol is accessible to the policies called for specific frames. This context is not shared between different WebSocket connections and it is destroyed when the connection is closed. The context contains all the message attributes (including authorization and authentication data) used by the upgrade request. This context should be accessed by the server and client policies in read only mode.

The following figure shows the flow of messages between the client, API Gateway, and the server in a typical WebSocket communication. It also shows at which point each of the API Gateway policies are called.



Connection expire time:
Enter a numerical value and choose the units from the list. The available options are: seconds, minutes, hours, and days. The default value is `0`, which means unlimited.

This is the absolute time for the connection to be active. For example, if this value is set to 1 hour, then after 1 hour the connection is dropped by API Gateway even if the connection is still active (frames are being sent).

Tip

You can define an idle timeout for the connection as a part of the remote host configuration.

Advanced settings

For details on the fields on this tab, see the section called “Advanced settings” in the *Configure relative paths* topic.

CORS settings

For details on the fields on this tab, see the section called “CORS settings” in the *Configure relative paths* topic.

Monitor a WebSocket connection

You can use the API Gateway Manager web console to monitor WebSocket traffic. You can view the initial HTTP `Upgrade` request and response on the **Traffic > HTTP** tab. You can view all the WebSocket frames processed by API Gateway on the **Traffic > WebSockets** tab.

To view all the WebSocket frames processed by API Gateway in API Gateway Manager, follow these steps:

1. Click the **Traffic** button at the top of the window.
2. Click the **WebSockets** tab.

   A view of all WebSocket frames sent from or received by API Gateway is displayed.

   Tip

   A message might consist of one or more frames.
3. Click a message frame to see a detailed view of the content. For example, if you click a frame sent from the client to the server, the origin, opcode (interpretation of the payload data), duration, length of the data, key used to mask the data, and payload itself are shown.

For more information on traffic monitoring using the API Gateway Manager web console, see the *API Gateway Administrator Guide*.

Configure 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, see *Configure conditions for HTTP interfaces*.

To configure an HTTP Watchdog, right-click a previously configured Remote Host in the Policy Studio tree, select **Watchdog** > **Add**, and configure the settings in the 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 (for example, `200`).

HTTP Request for Polling:
The fields in this section enable you to configure the type and URI of the HTTP request to poll the Remote Host with. The default is 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**, and specify 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 select **Poll if up**. In this case, the Watchdog uses real HTTP requests to poll the Remote Host as long as it sends them, but starts sending test poll requests when it is not sending HTTP requests to the Remote Host 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).

Configure 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 API Gateway instance 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.

Configure 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 (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 *Configure remote host settings* and *Configure 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.

Configure 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.

Configure a POP client

Overview

The API Gateway 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 an API Gateway instance 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 instance 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 instance. 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 Rendezvous. The API Gateway can both produce and consume messages for TIBCO Rendezvous. This topic describes how to integrate TIBCO Rendezvous.

TIBCO Rendezvous integration

The API Gateway can act as a producer and a consumer of TIBCO Rendezvous messages. In both cases, a Rendezvous daemon must be configured. This is responsible for communicating with other Rendezvous programs on the network.

Producing TIBCO Rendezvous Messages:
You must configure the following steps to produce messages and send them to another Rendezvous program:

- *TIBCO Rendezvous Daemon*
- *TIBCO Rendezvous Routing*

Consuming TIBCO Rendezvous Messages:
A TIBCO Rendezvous Listener can be configured at the API Gateway instance level to consume Rendezvous messages. You must configure the following steps to consume Rendezvous messages:

- *TIBCO Rendezvous Daemon*
- *TIBCO Rendezvous listener*

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 instance level in the API Gateway. To configure the API Gateway instance to use an OpenSSL engine instead of the default OpenSSL implementation, right-click the instance 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 dialog 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 dialogs are therefore associated with the cryptographic engine.

Each dialog 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 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 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.

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 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 them in both filters.

You can also use external connections to configure a group of related URLs. This allows you 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.

To configure external connections, right-click 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 *Authentication Repository*.

Client credentials

Client credentials allow you to configure client authentication settings at a global level. You can configure a client credential profile for the following authentication options:

- API keys as a client
- HTTP basic
- Kerberos
- OAuth 2.0 as a client

After configuring a client credential profile globally, you can select that profile for use at the filter level (for example, in the **Connection** or **Connect To URL** filters).

To add a client credential profile for a particular authentication mechanism, right-click the appropriate node under the **Client Credentials** node under the **External Connections** node. For more details, see *Configuring client credentials*.

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 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 *Configuring Connection Groups*.

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 *Database Connection*.

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 *Configuring ICAP Servers*.

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 *JMS Services*.

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 user name 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 *Configuring LDAP Directories*.

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 *Proxy Servers*.

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*.

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 *SiteMinder/SOA Security Manager Connection*.

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 *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 *SiteMinder/SOA Security Manager Connection*.

Syslog servers

You can configure Syslog servers globally, and then select them as a customized logging destination for an API Gateway instance. 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 to log to.

For details on how to configure the API Gateway instance to log to this remote Syslog Server, see the *API Gateway Administrator Guide*.

TIBCO

You can add a connection to TIBCO Rendezvous Daemon. To add a connection, 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 *TIBCO Rendezvous Daemon*.

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 *Tivoli integration*.

URL connection sets

URL connection sets are used by API Gateway filters to round-robin between groups of external servers (for example, Entrust GetAccess, 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 **XKMS URL Sets** node 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 Con-**

nections > URL Connection Sets node in the tree. Select the **Add a URL Set** option to display the **URL Group** dialog. For more details, see *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 *XKMS certificate validation*.

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.

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 *Manage 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.

Axway PassPort Repositories

The API Gateway can integrate with Axway PassPort to authenticate and authorize users for resources. To authenticate users against an Axway PassPort repository, right-click **Axway PassPort Repositories**, and select **Add a new Repository**.

For more details on configuring a connection to an Axway PassPort repository, see *Axway PassPort Authentication Repository*. For details on integrating with Axway PassPort using an authorization filter, see *Axway PassPort Authorization*.

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 in *SiteMinder Certificate Authentication*.

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. See 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 *Database Connection*. 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 *Database Query*.

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.

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 *Manage API Gateway users*.

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 *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. For more details, see *Configuring LDAP Directories*.

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 represents 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.

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 21, Oracle Access Manager Filters.

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 *Oracle Security Service Module settings (10g)*.

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 at-

tributes 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 *RSA Access Manager Authorization*.

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 details on how to configure the API Gateway to communicate with a Tivoli server, see *Tivoli integration*.

Axway PassPort Authentication Repository

Overview

Axway PassPort provides a central repository, identity broker, and security audit point for your Axway Business-to-Business Integration (B2Bi) or Managed File Transfer (MFT) solutions. Axway PassPort centralizes and simplifies provisioning and management for your entire online ecosystem, enabling secure collaboration between applications, divisions, customers, suppliers, and regulatory bodies.

To authenticate users against an Axway PassPort repository, in the Policy Studio tree, select **External Connections** > **Authentication Repository Profiles**, right-click **Axway PassPort Repositories**, and select **Add a new Repository**. This topic explains how to configure the Axway PassPort repository settings, and provides details on Axway PassPort repository registration.

Configuration

Complete the following fields to configure an Axway PassPort repository:

Repository Name:
Enter a descriptive name for this repository.

Hostname:
Enter the hostname or IP address of the server running PassPort.

Shared Secret:
Enter the PassPort shared secret. This is specified during the PassPort installation.

CSD Name:
Enter the name of the Axway Component Security Descriptor (CSD) file to use when registering with PassPort. Defaults to `csd.xml`.

Note

The CSD file must be deployed in the API Gateway group's `conf` directory in your API Gateway installation:

```
<install_dir>/apigateway/groups/group-<n>/conf
```

PassPort Certificates—HTTPS:
Select the certificate used for PassPort SSL communication. To export this certificate from PassPort, perform the following steps:

1. In the PassPort user interface, click **Administration** > **Server SecuritySettings**.
2. Note the certificate used for **Default_HTTPS**.
3. Click **Security** > **Certificates**.
4. Select the certificate noted in step 2 (defaults to `CN=PassPortSecured, O=Axway,C=FR`), and click **Export Certificate**.
5. In the **Export Certificate** dialog, select a **File Extension** of `.cer`.
6. Click **OK**, and select a location to save the certificate.

To import this certificate into the API Gateway, perform the following steps:

1. In the API Gateway **Authentication Repository** dialog, click **Select**.

2. In the **Select Certificate** dialog, click **Create/Import**.
3. In the **Configure Certificate and Private Key** dialog, click **Import Certificate**.
4. Select the certificate that was exported from PassPort.
5. Give the certificate an **Alias Name** manually, or click **Use Subject**.
6. Click **OK**.
7. Select the certificate from the list, and click **OK**.

PassPort Certificates—HTTPS Client Authentication (Optional):
You can configure PassPort to use a different certificate for its client authentication protocol. To do this, repeat the steps for the HTTPS certificate, except when exporting from PassPort in step 2, make a note of the **Default_HTTPS_Client_Auth** certificate.

Ports—HTTPS:
Enter the HTTPS port that PassPort is using. In PassPort, this is found under **Administration** > **Server Ports Configuration**. Defaults to `6453`.

Ports—HTTPS Client Authentication:
Enter the HTTPS client authentication port that PassPort is using. In PassPort, this is found under **Administration** > **Server Ports Configuration**. Defaults to `6666`.

Authentication—Domain:
Enter the PassPort domain that this repository is using for authentication and authorization. Defaults to `Synchrony`.

Axway PassPort Repository Registration

The external connection to Axway PassPort requires that communication with the Axway PassPort server is performed over a secure connection using two-way SSL authentication. This means that the PassPort server must be able to identify and trust the client connection, and this trust is established by registration.

The first connection from the API Gateway to PassPort initiates registration. A public-private key pair is created and a Certificate Signing Request (CSR) is submitted to PassPort. This is where the **Shared Secret** and **HTTPS** port values are used. While the CSR is pending, the repository is unable to process any requests. However, registration is a once off event, and when complete, it does not need to be repeated.

When registration is complete, the key and signed certificate are stored in a Java Key Store file in the following directory of your API Gateway installation:

```
<install_dir>/apigateway/groups/group-<n>/conf
```

Troubleshooting registration issues
In the unlikely event that automatic registration fails, you should check the following:

- Ensure the time on the API Gateway is synchronized with the time on the PassPort machine. When PassPort processes the CSR, it sets the **Valid From** date to the current time. If the PassPort time is ahead of the API Gateway time, the API Gateway is unable to use the certificate because it is not yet valid. The error in the trace log is as follows:

  ```
  java.security.cert.CertificateNotYetValidException:java.security.cert.
  CertificateNotYetValidException
  ```

- By default, PassPort blocks for up to 2 seconds waiting for the CSR to be processed. You can configure this value in the PassPort administration user interface under **Administration** > **System Properties** (`am.registration.cert.signature.wait.time`). If the signing request takes longer than this, one of the following errors may be logged:

  ```
  Authentication exception when authenticating system:
  ```

```
com.axway.passport.am.api.v2.service.external.PassportConnectionException: Registration
is still in progress. Certificate Signing Request has not yet been validated, please try
again later. [Status: Waiting Validation]

Authentication exception when authenticating system:
com.axway.passport.am.api.v2.service.external.PassportConnectionException: Registration
is still in progress. Certificate Signing Request has not yet been signed, please try
again later. [Status: Waiting Signing]
```

This is generally a transient error that may be generated when the initial registration is in progress. Resubmitting the request should succeed. If the error persists, check in the PassPort administration user interface for the reason why the signing request has been delayed.

- If the registration request has been refused by the PassPort administrator, the following error is displayed:

```
Registration has been refused. Please contact the PassPort Administrator for further
information. [Status: Validation Refused]
```

- If CSR processing fails for some other reason, the following error is logged:

```
Registration has failed and API Gateway is unable to communicate with PassPort. Please
contact the PassPort Administrator or try manually re-triggering registration. [Status:
...]
```

To resolve this, contact the PassPort administrator to see why the signing request failed. To retry registration, you needs to manually re-trigger registration, as explained in the next subsection.

Re-triggering registration manually

When registration is complete, the API Gateway does not repeat the process, the generated Java Key Store (JKS) is used for all subsequent connection attempts. However, if for any reason the key pair in the JKS is no longer trusted by PassPort, or if registration is not being processed, you can trigger the registration procedure manually.

The simplest way to re-trigger registration is to change the repository name and re-deploy. However, if this is not an option, you can manually remove the keystores.

The format of the JKS file names is as follows:

`keystore_<Repository Name>_<Hostname>_<HTTPS Client Authentication port>.jks`

All non-alphanumerics are replaced with an underscore (_).

For example, given the following repository details:

- **Repository Name**: `PassPort - local`
- **Hostname**: `passport-host`
- **HTTPS Client Authentication**: `6666`

The keystore name is `keystore_PassPort___local_passport_host_6666.jks`.

To trigger registration, perform the following steps:

1. Back up the JKS associated with the Axway PassPort Authentication Repository being reset.
2. Delete this JKS.
3. Restart the API Gateway instance. The deleted file is recreated and registration is initiated.

Note

If registration has not been completed when the registration is being re-triggered, you must delete the following additional temporary files to ensure clean registration:

- `.jks.cs`
  `keystore_<`*`RepositoryName`*`>_<`*`Hostname`*`>_<`*`HTTPSClientAuthenticationPort`*`>rid`
- `.jks.pu`
  `keystore_<`*`RepositoryName`*`>_<`*`Hostname`*`>_<`*`HTTPSClientAuthenticationPort`*`>b`
- `.jks.ke`
  `keystore_<`*`RepositoryName`*`>_<`*`Hostname`*`>_<`*`HTTPSClientAuthenticationPort`*`>y`

In addition, to avoid future confusion, it is good practice to ensure that all redundant certificates and signing requests are removed from PassPort using the PassPort administration user interface.

Configuring client credentials

Overview

Client credentials enable you to globally configure client authentication settings for the following authentication options:

- API keys as a client
- HTTP basic
- Kerberos
- OAuth 2.0 as a client

Note

For more information on configuring OAuth 2.0 client credentials, see the *API Gateway OAuth Guide*.

You can configure settings for client credentials under the **External Connections** node in the Policy Studio tree, which you can then specify at the filter level (for example, in the **Connection** and **Connect To URL** filters).

Configuring API key client credential profiles

API key client credential profiles enable you to globally configure authentication settings for API keys as a client. API keys are supplied by client users and applications calling REST APIs to allow the API service provider to track and control how the APIs are used (for example, to meter access and prevent abuse or malicious attack).

To configure API key client credential profiles, you must first configure a provider. The following Amazon Web Services provider configurations are included in an out-of-the-box installation of API Gateway:

- Amazon AWS V2 Signing
- Amazon AWS V4 Signing

Adding API keys

To add an API key for an existing API key provider, click an API key client credential node (for example, **Amazon AWS V2 Signing**), and click the **Add** button on the **API Key Credential Profiles** tab of the **API Key Credential Profile** window. Complete the following fields on the **Add API Key** dialog:

Name:
Enter a suitable name for this API key.

API Key ID:
Enter an identifier for this API key. This is the Amazon client ID associated with your Amazon account.

API Key Secret:
Enter a secret for this API key. This is the Amazon secret associated with your Amazon account.

Tip

To sort the list view of client credential profiles, click the column heading.

After you have configured your API key client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see the *Connection* and *Connect to URL* topics.

Adding API key providers

To configure a new API key provider, right-click **API Keys**, and select **Add API Key Client Credential**. Complete the following fields on the **API Key Service Provider Configuration** dialog:

Name:
Enter a suitable name for this API key service provider configuration.

Sign the request using:
Select an option to specify how the request is signed. The options are:

- Amazon API Key Signing V2
- Amazon API Key Signing V4
- No Signing

Put the API key in:
Select an option to specify where to put the API key in the request and enter a name in the **named** field. The options are:

- Header – The parameter is added to the header of the request.
- Query String/Form Body – If the incoming request is a GET request, the parameter is added to the query string. If the incoming request is a POST request, the parameter is added to the content body.

For example, to put the API key in the header of the request in a field named `AWSKeyId`, choose **Header** and enter `AWSKeyId`.

Tip
To change the configuration of an existing API key service provider, click the API key client credential node, and edit the settings on the **API Key Configuration** tab of the **API Key Credential Profile** window.

Configuring HTTP Basic/Digest client credential profiles

A client can authenticate to the API service provider with a user name and password combination using HTTP basic authentication or HTTP digest authentication.

To add a HTTP Basic/Digest client credential profile, click the **HTTP Basic** node, and click the **Add** button on the **HTTP Basic/Digest Client Credentials** window. Complete the following fields on the **Add HTTP Authentication Profile** dialog:

Profile Name:
Enter a suitable name for this HTTP authentication profile.

Choose Authentication Type:
Select the **Basic** or **Digest** radio button to specify the type of HTTP authentication to use, and enter a user name and password in the **Username** and **Password** fields.

Automatically send credentials:
Select this option to automatically send credentials in the request. This is the equivalent of **Send token with first request** in Kerberos. This option is selected by default.

After you have configured your HTTP client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see the *Connection* and *Connect to URL* topics.

Configuring Kerberos client credential profiles

A client can 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.

To add a Kerberos client credential profile, click the **Kerberos** node, and click the **Add** button on the **Kerberos Client Credentials** window. Complete the following fields on the **Add Kerberos Profile** dialog:

Profile Name:
Enter a suitable name for this Kerberos authentication profile.

Kerberos Client:
Click the browse button to select a Kerberos client. The selected Kerberos client has two roles. First, it must obtain a Kerberos Ticket Granting Ticket (TGT). 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** field. For more details on configuring Kerberos clients, see the *Kerberos Clients* topic.

Kerberos Service Principal:
Click the browse button to select a Kerberos service principal. The Kerberos client 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 might 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 *always* send the `Authorization` HTTP header that contains the Kerberos service ticket on the first request to the Kerberos service. This option is selected by default.

Send body only after establish context:
Select this option to 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). This option is not selected by default.

Pass when service returns 200 even if context not established:
In rare cases, a Kerberos service might 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 might not contain the `WWW-authenticate` HTTP header. Select this option 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. This option is not selected by default.

After you have configured your Kerberos client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see the *Connection* and *Connect to URL* topics.

Configure Sentinel servers

Sentinel server overview

Axway Sentinel is a Business Activity Monitoring (BAM) product that collects, aggregates, correlates, and reports events from API Gateway and other Axway products, applications, and systems throughout your infrastructure. Sentinel is a separate product that you can buy from Axway or an authorized partner. This topic describes how you can configure API Gateway to send events to Axway Sentinel server.

Note

A complete documentation set for Axway Sentinel is available on the Axway Support website: https://support.axway.com.

To add a new Sentinel server connection, in the Policy Studio tree view, under the **External Connections** node, right-click the **Sentinel Servers** node, and select **Add a Sentinel Server**.

General settings

You can configure the following general settings for the Sentinel server:

Name:
Enter a suitable name for this Sentinel server.

Host:
Enter the host name (FQDN) or IP address of the Sentinel server.

Port:
Enter the port number that the Sentinel server is listening for events on.

Use overflow file:
Select this option to use an overflow file to store API Gateway event data when there is no connection between API Gateway and Sentinel.

Name:
Enter a suitable name for the overflow file.

Size (MB):
Enter the maximum size of the overflow file.

Encoding:
Enter the encoding type to use. The default is `utf-8`.

User agent settings:
By default API Gateway registers events against the API Gateway's name in the topology. To use another name, select the **Or the following name** option and enter an alternative name to use. By default API Gateway registers events against the host name of the machine running the API Gateway. To use another host name, select the **Or the following name** option and enter an alternative host name to use.

Further information

For more detailed information on Sentinel integration, see the *Sentinel Integration Guide* available from Axway Support.

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 table in the section called “Database Connection Pool Settings”.

Prerequisites

Before configuring a database connection, you must add the JDBC driver files for your chosen database to your API Gateway and Policy Studio installations.

API Gateway
To add the third-party JDBC Driver files for your database to the API Gateway, perform the following steps:

1. Add the binary files for your database driver as follows:
   - Add `.jar` files to the *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart Policy Studio.

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;`

Database	Example Connection URL
	`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`

Field Name	DBCP Configuration Property
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 *Send to ICAP* topic. This topic includes example policies that show an **ICAP** filter configured in REQMOD and RESPMOD modes.

JMS Services

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.

Java Message Service (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 JMS API to connect to other messaging systems that expose a JMS interface, including Oracle WebLogic Server, IBM MQSeries, JBoss Messaging, 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 API Gateway instance level, and configure the API Gateway to consume a JMS queue or topic. Right-click the instance 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 **Add**, and entering 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 instance 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 instance 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 instance 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 an instance 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 instance 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”.

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 API Gateway instance-level **Kerberos Configuration** settings. To configure these settings, right-click the instance 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 *Kerberos Principals*.

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 the *API Gateway Administrator Guide*.

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. For more information on configuring the **Keytab Entry** dialog, see the *Kerberos Keytab*.

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 the *API Gateway Administrator Guide*.

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 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 representation of the name (for example, which can be used when searching for the Principal in an Access Control List (ACL)).

Principal Name Type	Explanation
`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 the Oracle support team.

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 in the table by clicking **Load Keytab**, and browsing to the location of an existing Keytab file. You can add a new Keytab Entry by clicking **Add Principal**. For more information on configuring the **Keytab Entry** dialog, see *Kerberos Keytab*.

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 *API Gateway Administrator Guide*.

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 instance-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 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.

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 *Manage 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.

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.

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.

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.

Prerequisites

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.0-sp1-cr005 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

Integration with CA SOA Security Manager requires CA TransactionMinder SDK version 6.0 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

For details on obtaining the required third-party binaries, see your CA product documentation.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart Policy Studio.

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 oes 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, you must use the `smreghost` tool on the API Gateway machine. The tool creates a file called `SmHost.conf`, which you can then upload into the API Gateway configuration using Policy Studio.

To generate a `SmHost.conf` file, perform the following steps:

1. Install the `smreghost` command on the machine on which the API Gateway is installed. For details on installing `smreghost`, see your CA product documentation.
2. Open a command prompt in the directory where you installed `smreghost`, 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 or IP address and port of the CA Policy Server.
3. The `smreghost` tool writes its output to a `SmHost.conf` file in the same directory.

When you have generated a `SmHost.conf` file, perform the following steps:

1. Copy the `SmHost.conf` file to the machine on which you are running Policy Studio.
2. Specify the file location using the browse button at the bottom right of the text area.
3. 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 the *API Gateway Deployment and Promotion Guide*.

After selecting the `SmHost.conf` configuration file, the connection details are displayed in the text area.

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).

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.

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`).

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 *Manage 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.

Manage certificates and keys

Overview

For 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 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. 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).

View 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** window 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.

Configure 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.

Create 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 belonging to a trusted CA whose key pair is stored in the certificate store.

Import 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).

Bind to a certificate at runtime

Alternatively, when configuring an HTTPS interface, you can also specify a certificate to bind to at runtime. In the **Configure HTTPS Interface** dialog, click **X.509 Certficate**, and select **Bind the Certificate at runtime**. For example, you can enter `${env.serverCertificate}`, and enter the certificate as an environment variable in the `envSettings.props` file.

For more details on specifying environment variables, see the *Deployment and Promotion Guide*.

Configure 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,

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.

Manage certificates and keystores

On the main **Certificates** window, 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.

Export certificates to a keystore

You can also export a certificate to a Java keystore. You can do this by clicking the **Keystore** button on the main **Certificates** window. 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** window. On the **Keystore** window, 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** window 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.

Configure key pairs

To configure public-private key pairs in the certificate store, select **Certificates and Keys** > **Key Pairs**. The **Key Pairs** window enables you to add, edit, or delete OpenSSH public-private key pairs, which are required for the Secure Shell (SSH) File Transfer Protocol (SFTP).

Add 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.

Manage 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`

Edit 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.

Delete 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.

Configure 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** window enables you to add, edit, or delete PGP public-private key pairs.

Add 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.

Manage 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`

Edit 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.

Delete 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.

Manage API Gateway users

Overview

The **Users and Groups** node in the Policy Studio tree enables you to manage API Gateway users and groups, which are stored in the API Gateway user store.

By default, the API Gateway user store contains the configuration data for managing API Gateway user information. The API Gateway user store is typically used in a development environment, and is useful for demonstration purposes. In a production environment, user information may be stored in existing user Identity Management repositories such as Microsoft Active Directory, Oracle Access Manager, CA SiteMinder, and so on. For more details, see the *API Gateway Integration Guide*.

Note

API Gateway users provide access to the messages and services protected by API Gateway. However, *Admin users* provide access to the API Gateway configuration management features available in Policy Studio, Configuration Studio, and API Gateway Manager. For more details, see *Manage Admin users*.

API Gateway users

API Gateway users specify the user identity in the API Gateway 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 tree. The users are listed in the table on the main panel. You can find a specific user by entering a search string in the **Filter** field.

Add API Gateway users

You can create API Gateway users on the **Users** page. Click the **Add** button on the right.

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**.

Add user attributes
You can specify optional user attributes on the **Attributes** tab, which is explained in the next section.

API Gateway user 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.

API Gateway user 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 tree. The user groups are listed in the table on the main panel. You can find a specific group by entering a search string the **Filter** field.

Add API Gateway user groups

You can create user groups on the **Groups** page. Click the **Add** button on the right to view the **Add Group** dialog.

Add 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.

Add group attributes
You can specify optional attributes at the group level on the **Attributes** tab. For more details, see the section called “API Gateway user attributes”.

Update API Gateway 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.

Manage WSDL and XML schema documents

Overview

WSDL files often contain XML schemas that define the elements that appear in SOAP messages. When you import a WSDL file to register a web service, the imported WSDL file, and any XML schemas included in the WSDL, are added to a global cache of WSDL and XML schema documents. You can also add XML schemas and WSDL documents to the cache independently.

If you select a cached WSDL file or XML schema in a **Schema Validation** filter, API Gateway can retrieve it 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.

API Gateway can maintain multiple versions of WSDL and XML schema documents in the global cache, and keeps an explicit version history as they change over time. The cache is prepopulated with many of the common XML schema documents that are used in web services, and these are shared across multiple web services.

Structure of the global cache

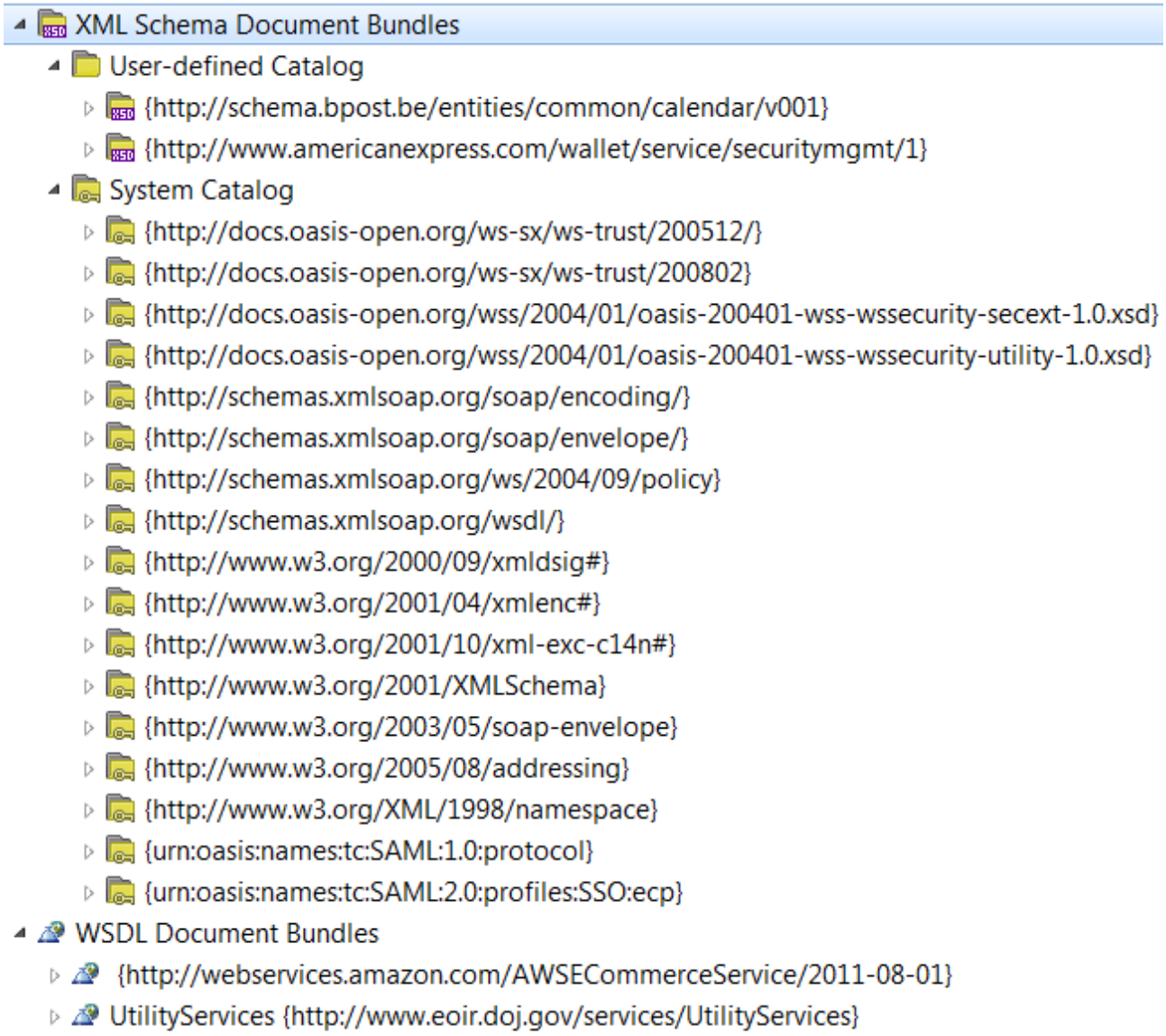
The global cache consists of both WSDL documents and XML schema documents.

The global cache of WSDL documents contains all WSDL documents that have been imported (either directly, or by registering a web service) into API Gateway.

The global cache of schema documents contains:

- User-defined catalog – This contains all user-defined schema documents that have been imported into API Gateway.
- System catalog – This contains all common schemas (for example, the SOAP encoding schema) that are preloaded during API Gateway installation.

The following figure shows the structure.



View cached WSDL or XML schema documents

Policy Studio provides a *read-only* view of cached WSDL and XML schema documents.

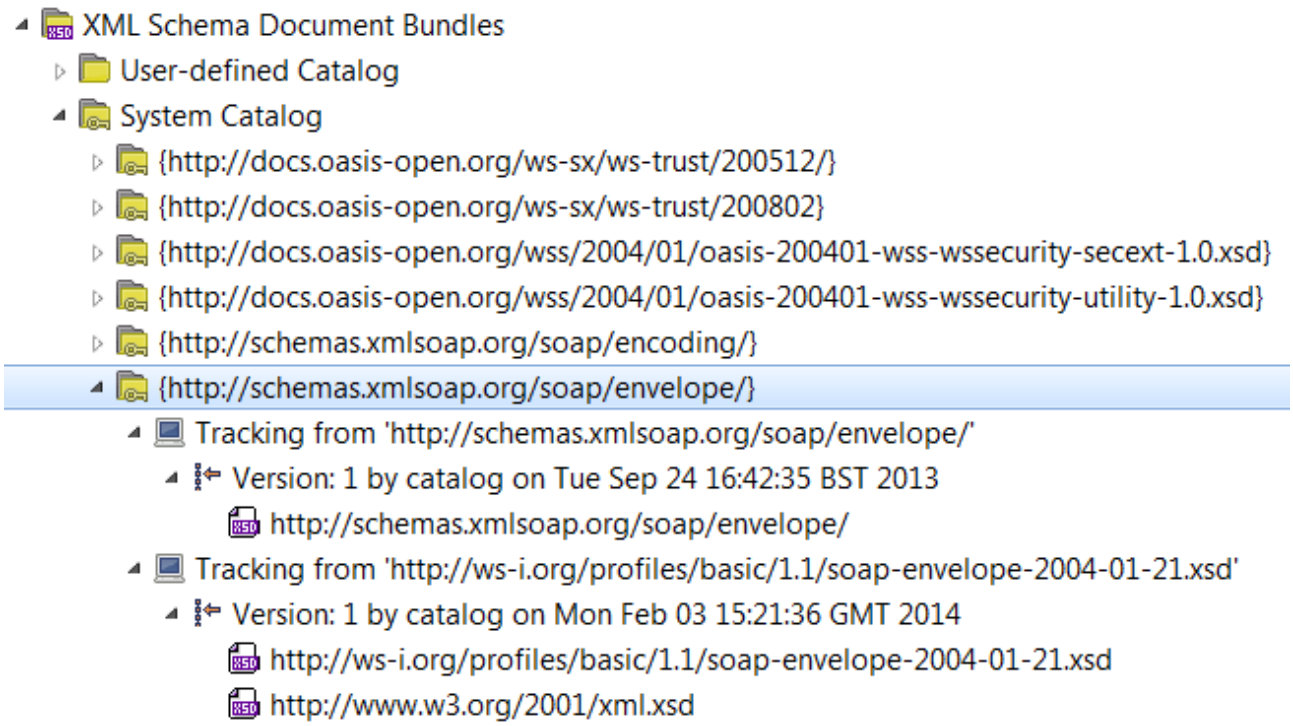
To view the global cache of WSDL documents, expand the **Resources > WSDL Document Bundles** tree node. The list of documents present in the cache is shown in the tree. To view the contents of a document, click the document node. A read-only view of the document is displayed in a tab on the right. WSDL documents can be added directly to this node, and they can be resynchronized.

To view the global cache of XML schema documents, expand the **Resources > XML Schema Document Bundles** tree node. This node contains two subnodes: **User-defined Catalog** and **System Catalog**. To view the documents in each catalog, expand the catalog node. The list of documents present in the catalog is shown in the tree. To view the contents of a document, click the document node. A read-only view of the document is displayed in a tab on the right.

XML schemas in the **System Catalog** are preloaded during API Gateway installation. You cannot add schemas to the system catalog, and schemas in the system catalog cannot be resynchronized. This is indicated by a key icon on these nodes.

The **User-defined Catalog** contains schemas that you have imported. These schemas can be resynchronized, and you can add new schemas directly to this catalog.

Document bundles are categorized by namespace/name, with subcategories used to indicate where the node is being tracked from (the URL it was retrieved from), and further subcategories for version information. The following figure shows an example of this.



Click a document bundle in the tree to list the versions that have been imported. Click a version to list all the documents contained in that version. Click a WSDL or XML schema document to view the WSDL or XML source for the document. A read-only view of the source is shown in a tab on the right.

Add XML schemas to the cache

To add an XML schema to the user-defined catalog in the global cache, perform the following steps:

1. In the Policy Studio tree view, right-click the **XML Schema Document Bundles > User-defined Catalog** node, and select **Add Schema**. The **Load Schema** dialog enables you to load a schema directly from the file system.
2. In the **File location** field, enter or browse to the location of the schema file and click **Next**.
3. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the schema. Click **Next**.

Note

If the XML schema fails validation, an error is displayed. For more information, see the section called "XML schema and WSDL document validation" and the section called "XML schema and WSDL docu-

ment limitations".

4. Click the **Finish** button to import the schema into the cache.

Add WSDL documents to the cache

WSDL documents are cached automatically when you import a WSDL file to register a web service. For more information, see the *Configure policies from WSDL files* topic.

Alternatively, you can import a WSDL document directly into the cache. To add a WSDL to the global cache, perform the following steps:

1. In the Policy Studio tree view, right-click the **WSDL Document Bundles** node, and select **Add a WSDL**. The **Load WSDL** dialog enables you to load a WSDL file directly from the file system, from a URL, or from a UDDI registry.
2. Select the appropriate option and enter or browse to the location of the WSDL file in the fields provided. To retrieve the WSDL file from a UDDI registry, click the **WSDL from UDDI** option, and click the **Browse UDDI** button. This option enables you to connect to a UDDI registry and search it for a particular WSDL file. For more information on how to retrieve a WSDL file from a UDDI registry, see the *Retrieving WSDL files from a UDDI registry* topic. Click **Next** to continue.
3. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL.
4. Click the **Finish** button to import the WSDL document into the cache.

Note

If the WSDL fails validation, an error is displayed. For more information, see the section called "XML schema and WSDL document validation" and the section called "XML schema and WSDL document limitations".

Important

If you import a WSDL document directly into the cache via the **WSDL Document Bundles** node, Policy Studio does *not* automatically generate policies and service handlers. To auto-generate policies and service handlers for a web service, you must use the **Register Web Service** option under the **Business Services > Web Service Repository** node.

Update cached WSDL or XML schema documents

You can update a WSDL document or XML schema in the cache by using the **Resynchronize WSDL** or **Resynchronize Schema** options. This enables you to import a more recent version of a WSDL or schema directly to the global cache.

To update a WSDL, perform the following steps:

1. Expand the **WSDL Document Bundles** node.
2. Expand the document bundle for the WSDL to be updated.
3. Right-click the **Tracking from** node and select **Resynchronize WSDL**.
4. Enter the location of the latest version of the WSDL and click **Next**. This defaults to the location from which the WSDL was originally loaded.
5. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL.
6. Click **Finish** to import the new version of the WSDL document into the cache.

API Gateway compares the contents of the WSDL and any of its imported schemas to those stored in its cache. If the

contents are different, it creates a new version of the WSDL.

To update an XML schema, perform the following steps:

1. Expand the **XML Schema Document Bundles > User-defined Catalog** node.
2. Expand the document bundle for the schema to be updated.
3. Right-click the **Tracking from** node and select **Resynchronize Schema**.
4. Enter the location of the latest version of the schema and click **Next**. This defaults to the location from which the schema was originally loaded.
5. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the schema.
6. Click **Finish** to import the new version of the schema document into the cache.

API Gateway compares the contents of the schema and any of its imported schemas to those stored in its cache. If the contents are different, it creates a new version of the schema.

See the section called “Update a web service” for more information on the types of changes to a WSDL or schema that trigger creation of a new version.

XML schema and WSDL document validation

XML schemas and WSDL documents are validated during import. If validation fails, an error is displayed. For example, validation fails if a schema type is referenced in a WSDL or schema, but that WSDL or schema does not contain an explicit import statement for the schema that contains the type definition for the referenced type.

Note

API Gateway requires all XML schema and WSDL documents to be present and valid at runtime, and applies very strict validation checks during import. Therefore, WSDL documents that validate successfully in other tools, will not necessarily validate in API Gateway, and you might need to preprocess any XML schemas or WSDL documents to make them valid, before attempting to import them in Policy Studio.

A good starting place is to ensure that all WSDLs and their schemas are Web Service Interoperability (WS-I) compliant before attempting to import them into Policy Studio. WS-I compliance ensures maximum interoperability between different vendors' implementation of web services standards, such as SOAP and WSDL. For more information on how to test for WS-I compliance, see the section called “Test a WSDL for WS-I compliance”.

An out-of-the-box installation of API Gateway contains a number of common XML schemas (for example, from OASIS and W3C) preloaded in the global cache. If you import a WSDL that imports any of these standard schemas, API Gateway uses the cached version of those schemas, instead of retrieving them from the web.

For example, the SOAP encoding schema is often imported into WSDLs that serialize message parts as SOAP encoding arrays. When such a WSDL is imported into Policy Studio, it recognizes that this schema already exists in the cache and does not download and import a duplicate version. This optimization avoids unnecessary duplication of shared schema resources and reduces the overall memory footprint of the API Gateway's configuration store. For more information, see the section called “Version and duplicate management”.

XML schema and WSDL document limitations

There are some additional limitations when importing XML schema or WSDL documents:

- **Non-SOAP bindings**
  Any HTTP and MIME bindings in the WSDL document are ignored, and only SOAP 1.1 and SOAP 1.2 bindings are imported.
- **Multiple ports for the same service**
  If the WSDL contains multiple ports for the same service (for example, a service is available over SSL and in the

clear, where the URL differs, but the binding is to the same SOAP service), you can select only one of the ports for import.

If you absolutely require both endpoints to be virtualized on API Gateway, you can create a separate service for each port in the WSDL. A distinct service handler is created for each service, which is responsible for processing requests for that service and routing them on to the endpoint URL specified in the port.

- **Schemas using the XML Schema namespace to extend element types, but not importing the namespace explicitly**
  Although some tools can work with invalid schemas like this, API Gateway requires them to be valid so it can run schema validation checks against the messages. The schema must be modified to import the namespace explicitly before you can import the schema in Policy Studio.
- **SOAP bodies with no children**
  For a SOAP binding style of `"document"`, API Gateway does not support any operation whose request SOAP Body has no child element.
- **Input-only SOAP operations**
  API Gateway does not currently support input-only SOAP operations, as these operations have no concept of a response message, and API Gateway has problems generating the **Web Service Filter** for these operations.

  When a **Web Service Filter** is generated as a result of virtualizing a web service, it handles both the request and response messages for the operations that are exposed by that service. Because input-only or notification-style SOAP operations have no concept of a response message, the **Web Service Filter** is not ideal, and a custom policy is recommended instead.
- **Output-only SOAP operations**
  API Gateway does not currently support output-only SOAP operations, as API Gateway has problems generating the **Web Service Filter** for these operations.

  Similar to input-only operations, output-only or solicit-response operations cause difficulties for a generated **Web Service Filter** and a custom policy is recommended to process this type of request instead.
- **Multiple bindings with different WS-Policy requirements**
  This results in multiple sets of security requirements that need to be configured by the user, and this is not currently supported by the **Import WSDL** wizard in Policy Studio.
- **WSDL messages that contain multiple parts**
  API Gateway does not support importing a WSDL where the `<wsdl:message>` contains multiple parts. A workaround is to change the WSDL so that each `<wsdl:message>` contains a single `<wsdl:part>` that references a schema complex type that *wraps* the message parts currently defined in each `<wsdl:message>`.
- **Schemas without the schemaLocation attribute**
  API Gateway requires schemas to import other schemas using both the namespace and schemaLocation attributes, except in the following circumstances:
  - The schemas are embedded in the WSDL. The schemaLocation attribute can be omitted from the schema import element and the schema resolver looks for the imported schemas in the WSDL itself.
  - The schemas import a schema from the system catalog (which comes preloaded with a number of common schemas, for example, the SOAP encoding schema). A schema from the system catalog can be imported into any schema using just the namespace attribute.

Version and duplicate management

API Gateway manages the resources associated with any WSDL documents or XML schema documents stored in the global cache. This includes the WSDL or XML schema documents themselves, and any WSDL or XML schemas imported by those documents.

When a new resource is being added to the global cache (for example, when you import a WSDL or add a new XML schema), API Gateway compares each resource against the existing resources in the cache. API Gateway tracks two items for each resource:

- The location of the resource

- The contents of the resource at that location

By tracking these two items, API Gateway can identify when a resource is a new version of an existing resource at the same location, or when a resource is the same as an existing resource at a different location. In both cases a new version of the resource is created.

This means that resources that are shared by multiple web services are not duplicated in the global cache, and that web services can be updated easily if the WSDL defining the back-end web service changes. For more information on updating a web service, see the section called “Update a web service”.

A common example of this is where a vendor implements multiple web services that share common data structures, for example, an object that stores employee details. When the WSDL and schema files for these services are generated, they will typically all import a common schema file that defines the employee details data structure. On importing these WSDLs into Policy Studio, API Gateway recognizes that the services all share a common `employees` schema, and avoids importing multiple copies of the schema into the cache. Instead, each imported web service *points* at the common schema.

Validate messages against XML schemas

The **Schema Validation** filter is used to validate XML messages against schemas stored in the cache. It can be configured to validate messages against schemas stored in the cache, and also against schemas embedded within WSDL stored in the cache. This filter is found in the **Content Filtering** category of filters in Policy Studio. For more information on configuring this filter, see the *Schema validation* topic.

Test a WSDL for WS-I compliance

Before importing a 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.

Policy Studio uses the Java version of the WS-I 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 WS-I testing tools is installed on the machine on which 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 **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.

Run 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.

Global caches

Overview

In cases where a backend 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 requestis 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 service to reprocess the identical request and generate the same response. The use of caching in this way helps divert unnecessary traffic from the service and makes it more responsive to new requests.

For example, assume you have deployed a 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 service handles hundreds or thousands of requests every day, this is 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 the API Gateway to cache the response from the service that contains the list of cities. Then when a request for this 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 service, therefore saving the processing effort required, and perhaps more importantly, saving the time it takes for the round trip.
- The 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 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, and even whether caches can be replicated across multiple API Gateway instances. This topic describes how to configure both local and distributed caches in the API Gateway, and shows examples 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 section called “Local caches”. 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 **Server Settings** node in the Policy Studio tree, and clicking **General** > **Cache**. Alternatively, you can access these settings in the Policy Studio main menu by selecting **Tasks** > **Manage Gateway Settings**. 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 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 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, ensure 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.

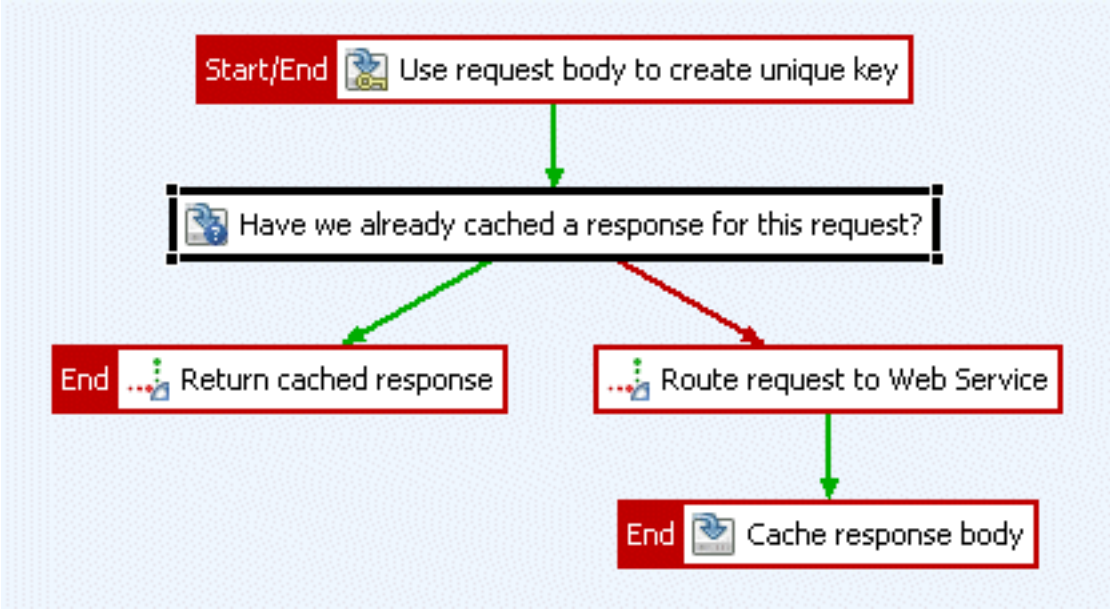
- **port (mandatory)**
  Specify the port on which the listener is listening, which by default is 4001. This setting is mandatory.
- **socketTimeoutMillis (optional)**
  Enter the number of seconds that client sockets wait when sending messages to this listener until they give up. The default is 2000 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 service. It uses the request body to

identify identical successive requests. 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 service. The following diagram illustrates the complete policy:



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 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 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, thr API Gateway routes the request to the service, and waits for a response. For more details on how to route messages, see *Getting Started with Routing Configuration*.

Cache Attribute:
When the response has been received from the 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*

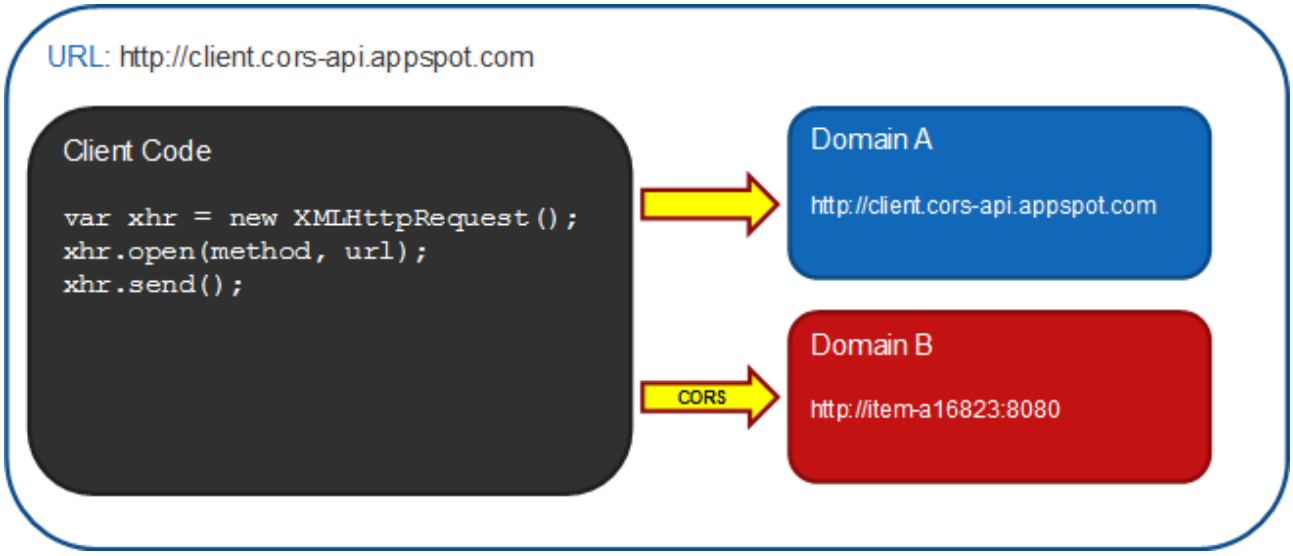
Cross-Origin Resource Sharing

Overview

Cross-Origin Resource Sharing (CORS) enables client-side code running in a browser in a particular domain to access resources hosted in another domain in a secure manner. Cross-origin requests are typically not permitted by browsers, and CORS provides a framework in which cross-domain requests are treated as same-domain requests.

For example, using CORS, JavaScript embedded in a web page can make an HTTP `XMLHttpRequest` to a different domain. This is used to send an HTTP or HTTPS request to a web server, and to load the server response data back into the script.

The following diagram shows an example CORS architecture:



This example is described as follows:

1. A user browses to the following URL:

   ```
   http://client.cors-api.appspot.com
   ```

2. The client page displayed on the left contains JavaScript that attempts to access resources in Domain A (`http://client.cors-api.appspot.com`), and in Domain B (`http://item-a16823:8080`).
3. Attempts by the browser to access resources in Domain A are granted because Domain A is the same domain in which the JavaScript code is running.
4. When the browser attempts to access resources in Domain B, it must use the CORS protocol because Domain B is in a different domain than that in which the JavaScript code is running. In this way, CORS enables client JavaScript code running in the browser in Domain A to access resources in Domain B.

The CORS standard provides CORS HTTP headers that enable servers to serve resources to permitted origin domains. Browsers support these CORS HTTP headers and enforce their restrictions. Browsers can also send *preflight* CORS requests to retrieve supported methods from the server using an HTTP `OPTIONS` method. Then on approval from the server, the browser client can send the request with the appropriate HTTP request method.

CORS request headers

The CORS HTTP request headers are as follows:

- `Origin`
- `Access-Control-Request-Method` (preflight only)
- `Access-Control-Request-Headers` (preflight only)

CORS response headers

The CORS HTTP response headers are as follows:

- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Credentials`
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age` (preflight only)
- `Access-Control-Allow-Methods` (preflight only)
- `Access-Control-Allow-Headers` (preflight only)

Adding a CORS Profile

To enable CORS support in API Gateway, you must first add a CORS profile in Policy Studio. Perform the following steps:

1. In the Policy Studio tree, select **Libraries** > **CORS Profiles**.
2. Right-click, and select **Add a CORS Profile**, and configure the settings on the following tabs:

General

Configure the following general settings:

- **Name**:
  Unique name of the CORS profile. Defaults to `Cross Origin Resource Sharing`.
- **Enable CORS**:
  Specifies whether CORS processing is enabled for the profile. Enabled by default.

Origins

The **Origins** table enables you to configure the list of origins that are allowed to access resources configured with this CORS profile, or exposed by a specific HTTP service. Click the **Add** button at the bottom right to add an origin to the table.

You can specify origins using the following values:

- * (permits all values supplied in the CORS `Origin` header)
- Domain (for example, `http://client.corsapi.appspot.com`)
- Wildcarded value (for example, `http://*.appspot.com` or `http://client.corsapi.appspot.com:*`)

Allowed Headers

The **Allowed Headers** table enables you to configure the list of HTTP headers that are permitted when requesting a resource exposed by this CORS profile or HTTP service. The list of headers is defined by the value of the `Access-Con-`

trol-Request-Headers CORS header. Click the **Add** button at the bottom right of the screen to add an allowed header to the table.

Note

The list of allowed headers is checked only during a CORS preflight OPTIONS request, which can be sent before access to the resource is granted.

Exposed Headers

The **Exposed Headers** table enables you configure the list of CORS HTTP response headers that are exposed to the client. Click the **Add** button at the bottom right of the screen to add an exposed header to the table.

You can specify the list of CORS headers that are exposed to the client, in response to a resource exposed by this profile or HTTP service. You do not need to include the following simple HTTP response headers:

- Cache-Control
- Content-Type
- Content-Language
- Expires
- Last-Modified
- Pragma

Credentials Support

The **Support credentials** setting specifies whether resources using this CORS profile or HTTP service support user credentials. When this option is selected, the Access-Control-Allow-Credentials CORS header is sent in the response, with a value of true. This setting is not selected by default.

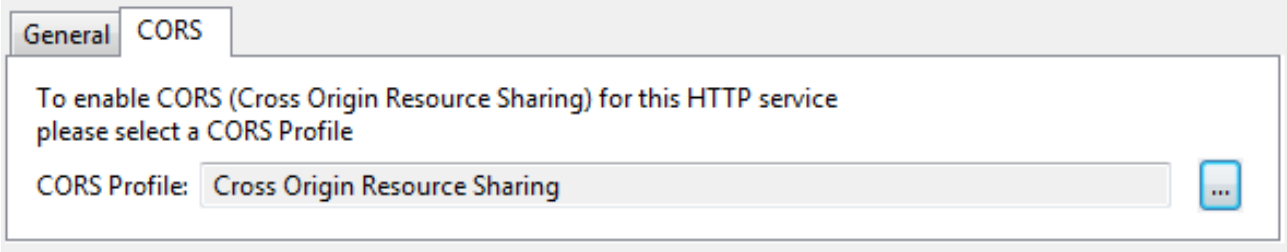
Preflight Results Cache

The **Max. age (seconds)** setting specifies how long the results of a CORS preflight OPTIONS request can be stored in the client preflight result cache. When this setting is configured, the Access-Control-Max-Age CORS header is sent in the response.

Configuring CORS for HTTP Services

You can also configure a CORS profile at the HTTP service level. This means that the settings configured for the profile are also applied to any child resolvers of this HTTP service. To configure CORS at the HTTP service level, perform the following steps:

1. In the Policy Studio tree, select an HTTP service (for example, **Listeners** > **API Gateway** > **Default Services**).
2. Right-click, and select **Edit** to display the **HTTP Services** dialog.
3. Select **CORS** tab, and click the browse button to select a preconfigured CORS profile. By default, no profile is selected, which means that CORS is disabled.
4. In the **Select CORS Profile** dialog, if no profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see the section called “Adding a CORS Profile”.



For more details on HTTP services, see *Configure HTTP services*.

Configuring CORS for Relative Paths

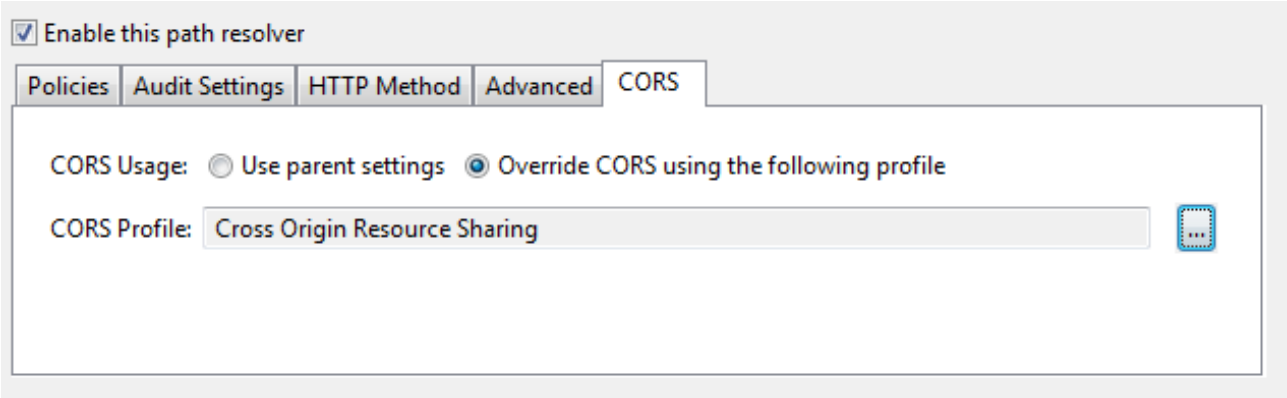
By default, the CORS profile set at the parent HTTP service level is used for all child resolvers of the HTTP service. However, you can override this at the relative path level as follows:

1. In the Policy Studio tree, select a list of relative paths (for example, **API Gateway** > **Listeners** > **Default Services** > **Paths**).
2. In the **Resolvers** screen on the right, right-click a resolver, and select **Edit** to display the **Resolve path to Policies** dialog.
3. Select **CORS** tab, and in the **CORS Usage** field, select **Override CORS using the following profile**. By default, no CORS profile is selected, and the parent settings are used.

Note

Relative paths can act as HTTP services, and can accommodate child resolvers. This means that when a relative path has children, and has a CORS profile configured, by default, the children use the parent profile (unless a child overrides it).

4. In the **CORS Profile** field, click the browse button to select a preconfigured CORS profile.
5. If no CORS profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see the section called “Adding a CORS Profile”.



Note

Similarly, you can also override CORS for the following relative path resolvers:

- Static Content Providers

- Static File Providers
- Servlet Application

For more details on relative paths and resolvers, see *Configure relative paths*.

Send to Amazon SQS

Overview

Amazon Simple Queue Service (SQS) is a hosted message queuing service for distributing messages amongst machines. API Gateway acts as a client to SQS and can send messages to SQS. You can use the **Send to Amazon SQS** filter to send messages to an SQS queue.

For more information on Amazon SQS, go to http://aws.amazon.com/sqs/.

General settings

Configure the following settings on the **Send to Amazon SQS** window:

Name:
Enter a suitable name for the filter.

AWS settings

AWS Credential:
Click the browse button to select your AWS security credentials (API key and secret) for Amazon SQS.

Region:
Select the region in which to access the SQS service. You can choose from the following options:

- US East (Northern Virginia)
- US West (Oregon)
- US West (Northern California)
- EU (Ireland)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- South America (Sao Paulo)
- US GovCloud

Client settings:
Click the browse button to select the AWS client configuration to be used by API Gateway when connecting to Amazon SQS. For more information on configuring client settings, see the section called “Configure AWS client settings”.

Send message settings

Configure the following settings on the **Send Message** tab:

Queue name:
Enter the name of the queue to send the message to. The default name is `publishQueue`. To create a new queue with the specified name click the **Create** option.

Send the message payload:
Select this option to send the message payload to the queue.

Or send the value of the attribute below to SQS:
Select this option to send the value of an attribute to the queue. Complete the following fields:

- **Attribute Name**:

Enter the name of the attribute.

- **Content Type**:
  Enter the content type to be used for sending the message to SQS (for example, `text/plain`).
- **Content Encoding**:
  Enter the content encoding.

Advanced settings

On the **Advanced** tab you can configure how to handle messages that are larger than 256KB in size. Configure these fields:

Split message into smaller ones:
Select this option to split the message into smaller messages before sending it to the queue.

Store in S3 and place pointer on SQS queue:
Select this option to store the payload in Amazon S3 and place a pointer to S3 in the queue. Configure the S3 settings as described in the section called “S3 settings”.

Further information

For more detailed information on Amazon Web Services integration, see the *AWS Integration Guide* available from Axway Support.

Upload to Amazon S3

Overview

Amazon Simple Storage Service (S3) is an online storage web service that you can use to store and retrieve any amount of data. API Gateway acts as a client to S3 and can upload data to S3. You can use the **Upload to Amazon S3** filter to upload data to Amazon S3.

For more information on Amazon S3, go to http://aws.amazon.com/s3/.

General settings

Configure the following settings on the **Upload to Amazon S3** window:

Name:
Enter a suitable name for the filter.

AWS settings

AWS Credential:
Click the browse button to select your AWS security credentials (API key and secret) for Amazon S3.

Region:
Select the region in which to store your data. You can choose from the following options:

- US East (Northern Virginia)
- US West (Oregon)
- US West (Northern California)
- EU (Ireland)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- South America (Sao Paulo)
- US GovCloud

Client settings:
Click the browse button to select the AWS client configuration to be used by API Gateway when connecting to Amazon S3. For more information on configuring client settings, see the section called “Configure AWS client settings”.

S3 settings

Bucket name:
Enter the name of the bucket in which to store the data. To create a new bucket with the specified name click the **Create** option.

Object key:
Enter the object key for the object to be stored. Alternatively, you can enter a selector that is expanded at runtime. For more details on selectors, see *Selecting configuration values at runtime*.

Encryption key:
Click the browse button to select an encryption key for the object.

How to store:
Select how to store the object. You can choose from the following options:

- Standard – This is the standard S3 storage option.
- Reduced Redundancy – This is a storage option within Amazon S3 for storing non-critical, reproducible data at lower levels of redundancy than standard storage.
- Glacier – This is a low-cost storage storage option for data archival.

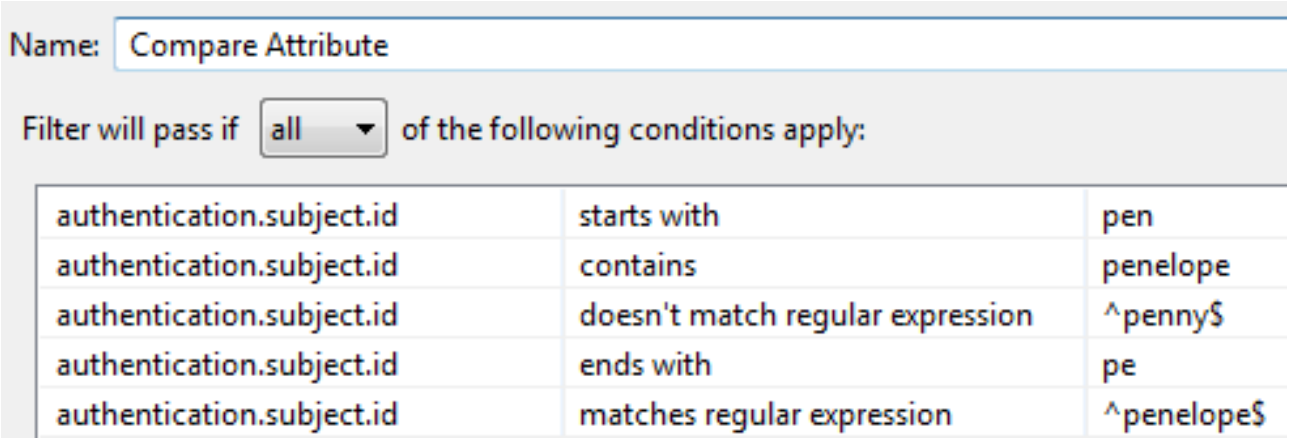
Further information

For more detailed information on Amazon Web Services integration, see the *AWS Integration Guide* available from Axway Support.

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`:



Configuration

Configure the following fields:

Name:
Enter an appropriate name for this filter.

Filter will pass if:
Select **all** or **one** of the specified conditions to apply. Defaults to **all**. Click the **Add** button at the bottom right to specify a rule condition. In the **Attribute filter rule** dialog, perform the following steps:

1. Enter a message attribute name in the **Value from** 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 request can be an HTTP GET or POST request. This filter is in the **Attributes** category in Policy Studio. For details on creating a REST request, see *Create REST Request*.

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 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.

This filter extracts all parameters from an incoming REST request, and stores 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 HTTP header values from an HTTP POST or GET request (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. 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 element

Overview

You can use the **Extract WSS Username Token** filter to extract a WS-Security `UsernameToken` from a message if it exists. The extracted `UsernameToken` token is stored in the `wss.usernameToken` message attribute.

To process the `UsernameToken` later in the policy, you can specify this message attribute in the configuration screen for the processing filter. For example, to sign the `UsernameToken`, 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 user name 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 `UsernameToken` 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.

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, 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*.

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.
`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.

Cookie Attribute Name	***Description***
`HTTPOnly`	Whether the browser should use cookies over HTTP only. This setting is not selected by default.

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 might 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 settings

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) to be used 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 of the assertion is sent. It is recommended that both the assertion **and** the SOAP Body must be signed if this option is se-	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	Meaning	Value
	lected. 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 settings

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**.

LDAP attribute authorization

Overview

The **LDAP RBAC** filter combines Local Directory Access Protocol (LDAP) with Role-Based Access Control (RBAC). This filter enables you to authorize a backend service based on user roles stored using LDAP. You can use the **LDAP RBAC** filter to read an attribute from LDAP, and compare it against some known values (for example, if `role` contains `engineering`, authorize the user). This filter combines functionality available in the **Retrieve from Directory Server** and **Compare Attribute** filters.

The **LDAP RBAC** filter enables you to define LDAP connection and search settings, and to configure how specified message attributes are processed. This filter also enables you to configure optional settings such as results caching and actions to take if a returned attribute is multi-valued.

General configuration

You must configure the following general setting:

Name:
Enter an appropriate name for this filter.

You must configure the following fields on the **Settings** tab:

Connection:
Click the button on the right to select your pre-configured LDAP directory server (for example, `openldap.qa.vordel.com`). For details on how to configure LDAP servers, see *Configuring LDAP Directories*.

Search Base:
Enter the Distinguished Name (DN) to use as the base from which the search starts (for example, `o=Vordel Ltd.,l=Dublin 4,st=Dublin,C=IE`).

Filter:
Enter the search filter to use. For example:

```
(&(objectclass=inetOrgPerson)(cn=${authentication.subject.id}))
```

Scope:
Select one of the following search scopes from the list.

- **Object**: Searches on the base DN only (compare)
- **One Level**: Searches the direct children of the base DN
- **Subtree**: Searches the base DN and all its descendants

Defaults to **Subtree**.

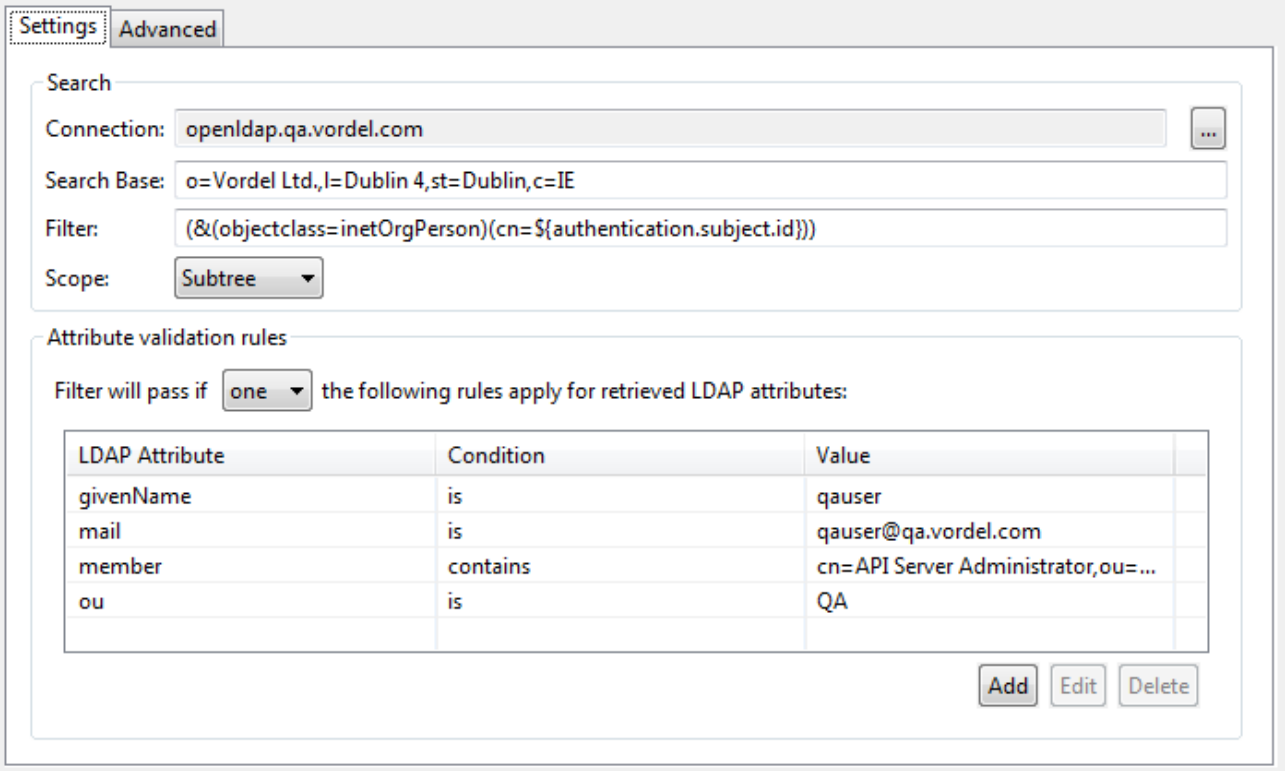
Attribute validation rules:
When the search completes, the attributes returned in the results are processed by the rules in the **Attribute validation rules** table. This processing is the same as the **Compare Attribute** filter. You can logically `AND` and `OR` rules together in the **Filter will pass if** list by selecting **all** or **one**.

For example, if **Filter will pass if** is set to **all**, and Rule A, Rule B, and Rule C all evaluate to true, the filter passes. However, if Rule A evaluates to false, the filter fails. If the **Filter will pass** is set to **one**, and Rule A and Rule B evaluate to false, but Rule C evaluates to true, the filter passes. However, if Rule C evaluates to false, the filter fails.

Select **all** or **one** of the specified conditions to apply. Click the **Add** button at the bottom right to specify a rule condition. In the **Attribute filter rule** dialog, perform the following steps:

1. Enter a message attribute name in the **LDAP attribute named** field on the left (for example, `member` or `mail`).
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**.

The following screen shows some example search settings and attribute validation rules:



Tip

When using this filter to determine if a user is a member of a `groupOfNames`, all the member attributes are concatenated together. The string containing the member attributes can be compared using a regular expression value provided in **Attribute validation rules**.

Because each attribute is not checked individually, you must create the regular expression string appropriately. For example, an expression such as `(?i:^.*${cert.subject.id}.*$)` allows for extra characters before and after the string searched for.

Advanced configuration

You can configure the following optional settings on the **Advanced** tab:

Cache settings:
Select whether to cache the LDAP search results. This setting is selected by default.

Store results in the cache:
Click the button on the right to select the pre-configured cache in which to store results. For more details, see *Global caches*. Select one of the following settings:

- **Use the LDAP search filter as cache key**: Uses the LDAP search filter configured on the **Settings** tab as the cache key.
- **Or use the following value as the cache key**: Enter a specific value for the cache key.

If returned attribute contains multiple values:
Select one of the following settings:

- **Concatenate values with the following**: Enter the character used to concatenate multiple attribute values. This setting is selected as a comma by default.
- **Use value at index**: Enter the index number of the attribute value to use. Defaults to `0`.

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 database, or by invoking a stored procedure call. The query results are represented as a list of properties. Each element in the list represents a query result row returned from the database for the specified SQL query or stored procedure call. These properties represent pairs of attribute names and values for each column in the row.

General settings

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database settings

Configure the following fields on the **Database** tab:

Database Location:
The API Gateway searches the selected database for the user's attributes. Click the browse button 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 *Database Connection*.

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 *Database Query*.

Advanced settings

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 database query results in message attributes. When selected, the message attribute names are generated based on the message attribute prefix and the attribute name. For example, if the specified prefix is `user` and the attributes are `PHONE` and `EMAIL`, the `user.PHONE` and `user.EMAIL` attributes are generated. This setting is selected by default.

Associate attributes with user ID returned by selector:
When the **Place query results into message attribute list** setting is selected, you can specify 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. This field is not selected 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 query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}`, or `${user.email[1]}`, and so on). Select this setting to use the legacy format for attribute naming instead.

Example of output attribute format with legacy attribute naming
The following table shows the output attribute format when legacy attribute naming is selected:

Place query results into user attribute list	Prefix for message attribute name (optional)	Output attribute format (when attribute name is PHONE)
Selected (default)	`user` (default)	• `attribute.lookup.list`: Map of retrieved attributes • `user.PHONE`: Attribute value • `${user.PHONE}`: Example selector
Selected (default)	None	• `attribute.lookup.list`: Map of retrieved attributes • `PHONE`: Attribute value • `${PHONE}`: Example selector
Not selected	`user` (default)	• `user.PHONE`: Attribute value • `${user.PHONE}`: Example selector
Not selected	None	• `PHONE`: Attribute value • `${PHONE}`: Example selector

Example of output attribute format without legacy attribute naming
The following table shows the output attribute format when legacy attribute naming is not selected:

Place query results into user attribute list	Prefix for message attribute name (mandatory)	Output attribute format (when attribute name is PHONE)
Selected (default)	`user` (default)	• `user`: List of properties, where each corresponds to a retrieved row (attribute name and value pair) • `${user[0].PHONE}`: Example selector
Not selected	`user` (default)	• `user.PHONE`: List of properties, where each corresponds to a re-

Place query results into user attribute list	Prefix for message attribute name (mandatory)	Output attribute format (when attribute name is PHONE)
		trieved row (attribute name and value pair) • `${user.PHONE[0]}`: Example selector

Prefix for message attribute:
Specifies an optional prefix for message attribute names used to store query results. The default prefix is `user`. For more details, see **Place query results into user attribute list** and **Enable legacy attribute naming for retrieved attributes**.

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.

Configure the following fields on the **Result Set Options** section:

Fail on empty result set:
Specify whether this filter fails if the result set is empty. This setting is not selected by default.

Attribute name for result set size:
Specify the attribute name used to store the size of the result set. Defaults to `db.result.count`.

Retrieve attribute 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 and retrieve that user's attributes represented as a list of search results. Each element of the list represents a list of multi-valued attributes returned from the directory server.

General settings

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database settings

Configure the following fields on the **Database** tab:

LDAP Directory:
The API Gateway queries the selected Lightweight Directory Access Protocol (LDAP) directory for user attributes. An LDAP connection is retrieved from a pool of connections at runtime. Click the browse button to select the LDAP directory to query. To use an existing LDAP directory, (for example, `Sample Active Directory Connection`), 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 *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 Selector Expression:
Select this option if the user ID is stored in a message attribute, and specify the selector expression used to obtain its value at runtime (for example, `${authentication.subject.id}`). 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 user name of the authenticated user. The name extracted from the specified message attribute is used to query the directory server. For more details on selector expressions, see *Selecting configuration values at runtime*.

From LDAP Search:
In cases where you have not already obtained the user's identity and the `authentication.subject.id` attribute has not been prepopulated 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:
You can specify where the API Gateway should begin searching the LDAP directory using a selector. The selector represents the value of a message attribute that is expanded at runtime. The two most likely message attributes to be used here are the authenticated user's ID and Distinguished Name. Select one of the predefined selectors from the list:

- `${authentication.subject.id}`
- `${authentication.subject.dname}`

Alternatively, you can enter a selector 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**

Unique Result:
Select this 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.

Attribute Name:
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 settings

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. This field is not selected 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 query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}`, or `${user.email[1]}`, and so on). Select this setting to use the legacy format for attribute naming instead.

Example of output attribute format with legacy attribute naming
The following table shows the output attribute format when legacy attribute naming is selected:

Prefix for message attribute name (optional)	Output attribute format (when attribute name is memberOf)
`user` (default)	• `attribute.lookup.list`: Map of retrieved attributes • `user.memberOf`: When retrieves only a single value for the given attribute • `user.memberOf.*` (for example, `user.memberOf.1`, `user.memberOf.2`, and so on): When retrieves multiple values for the given attribute • `${user.memberOf}`: Example selector
None	• `attribute.lookup.list`: Map of retrieved attributes • `memberOf`: When retrieves only a single value for the given attribute • `memberOf.*` (for example, `memberOf.1`, `memberOf.2`, and so on): When retrieves multiple values for the given attribute • `${user.memberOf}`: Example selector

Example of output attribute format without legacy attribute naming
The following table shows the output attribute format when legacy attribute naming is not selected:

Prefix for message attribute name (mandatory)	Output attribute format (when attribute name is user.memberOf)
`user` (default)	• `user`: List of search results, where each element of the list corresponds to search results (pairs of attribute names and values) • Example selector: `${user[0].memberOf[0]}`

Prefix for message attribute:
You can specify an optional prefix for message attribute names. The default prefix is `user`. For more details, see **Enable legacy attribute naming for retrieved attributes**.

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.

Retrieve attributes from JSON message

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.

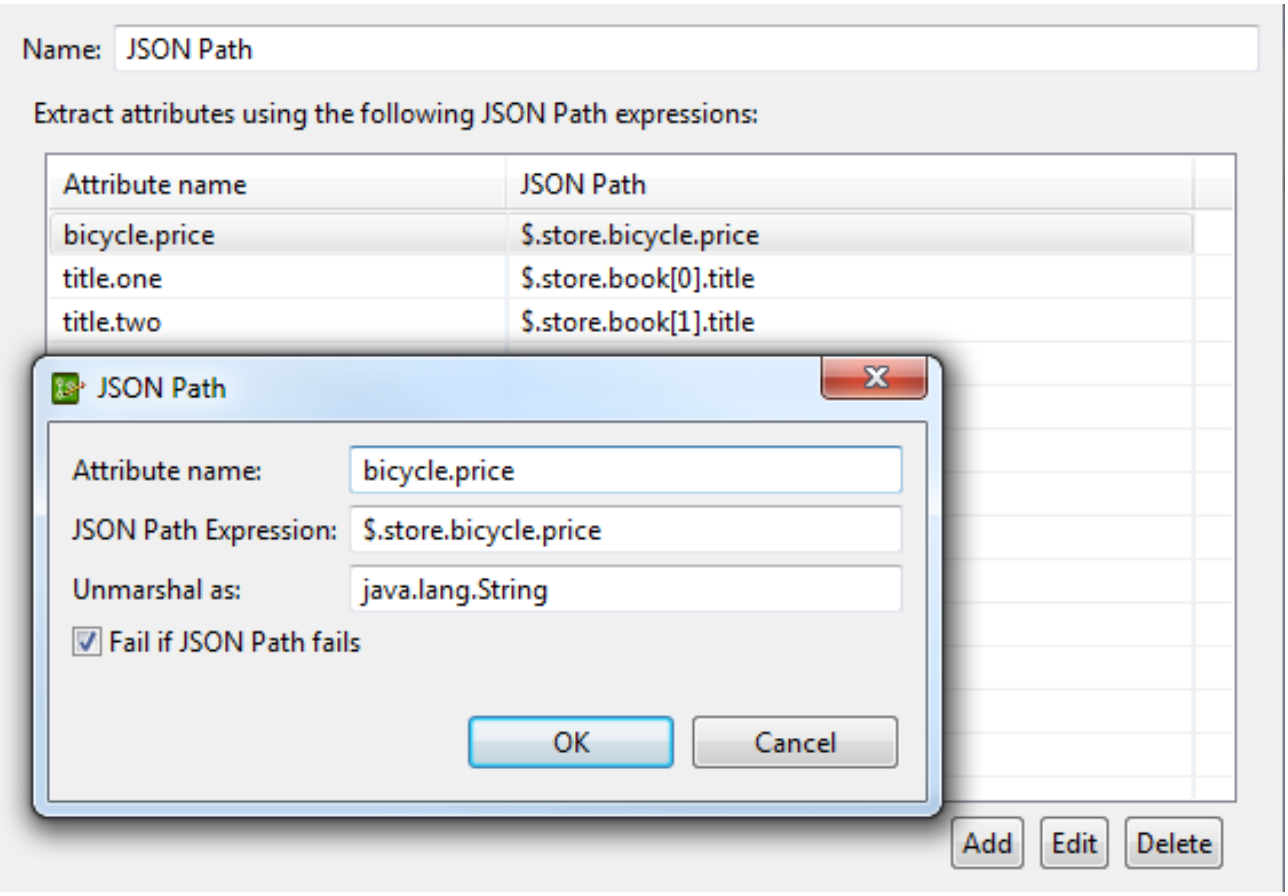
Note

If no attributes are specified, the API Gateway retrieves all the attributes in the message and sets them to the `attribute.lookup.list` attribute.

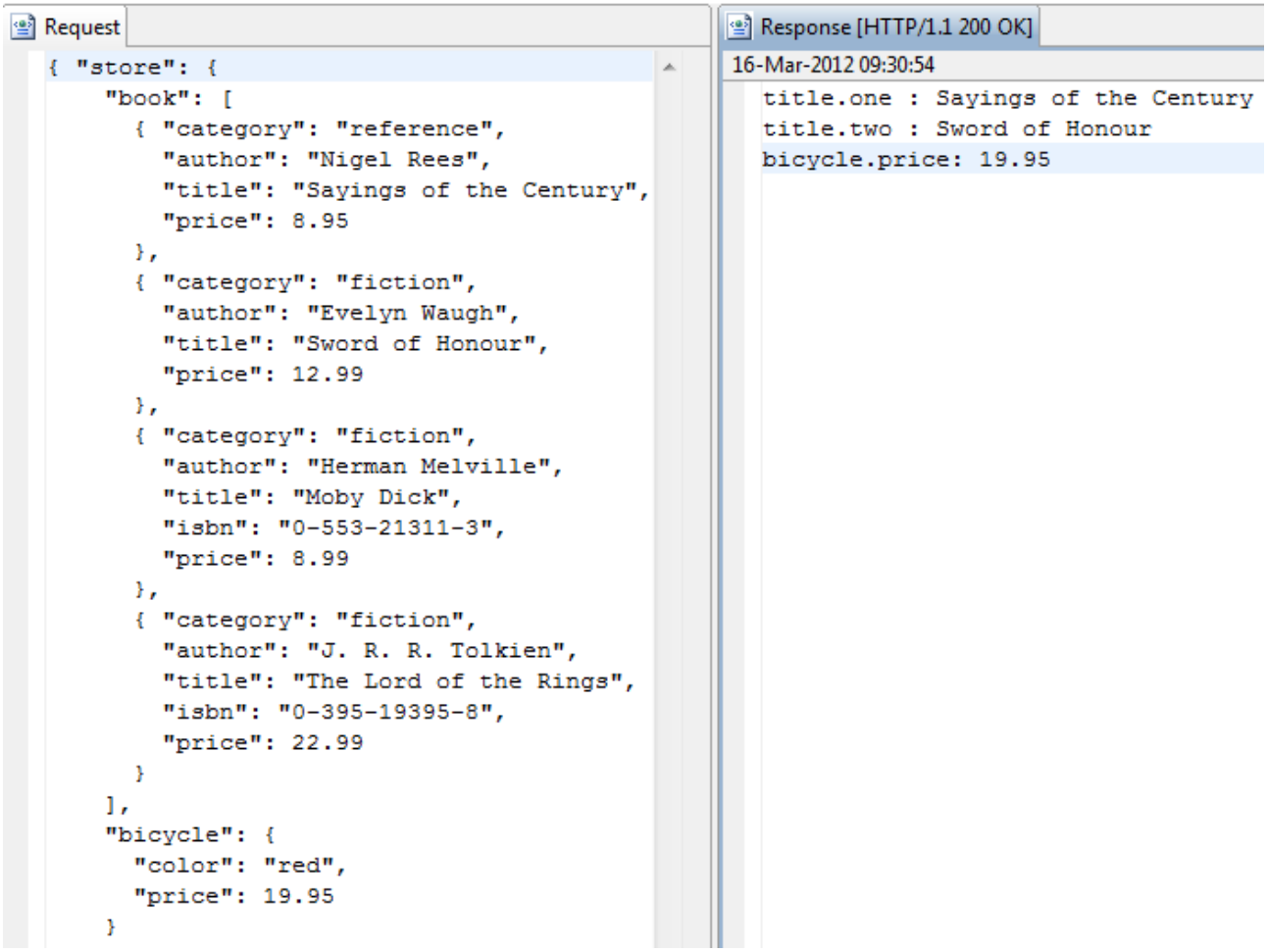
JSON Path examples

The following are some examples of using the **Retrieve Attributes with JSON Path** filter to retrieve data from a JSON message.

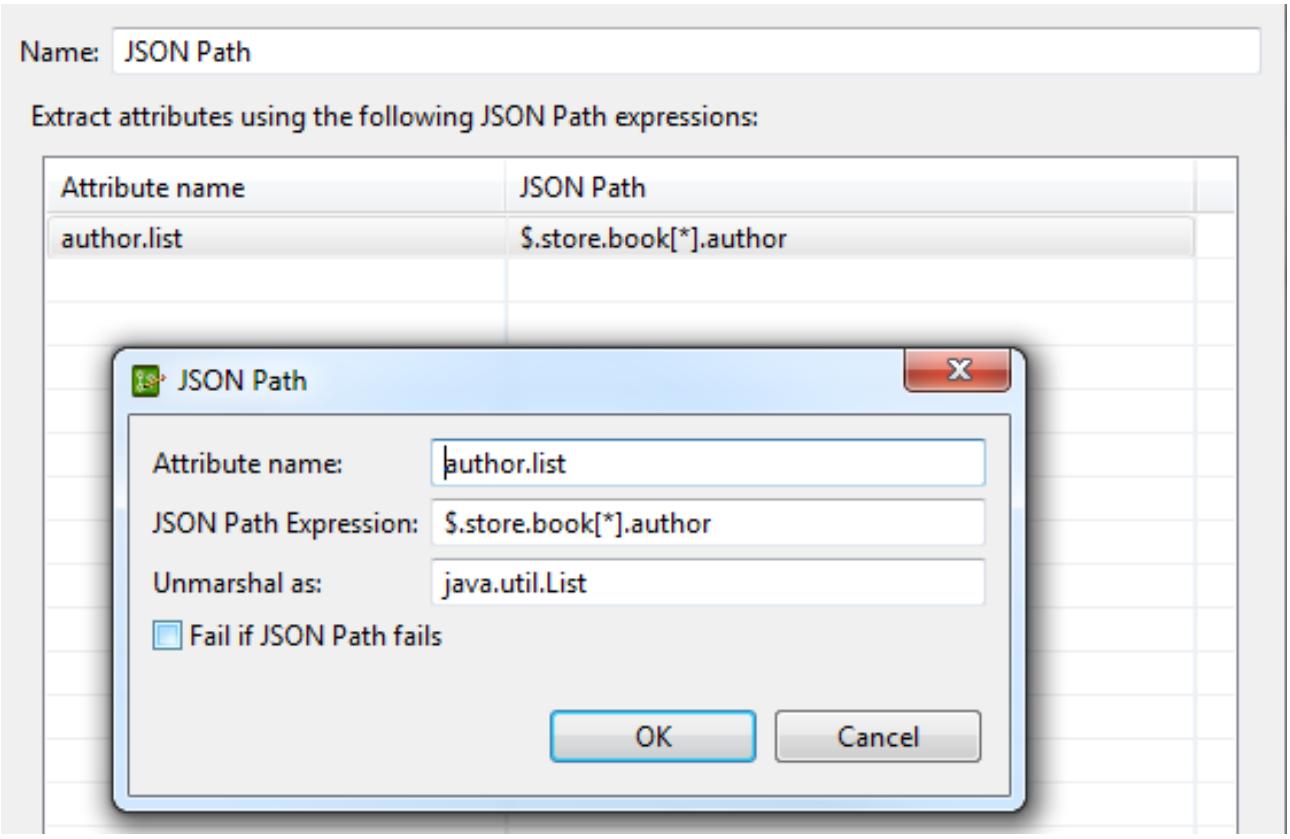
Retrieving attributes
The following example retrieves three different data items from the JSON message and stores them in the specified message attributes as strings:



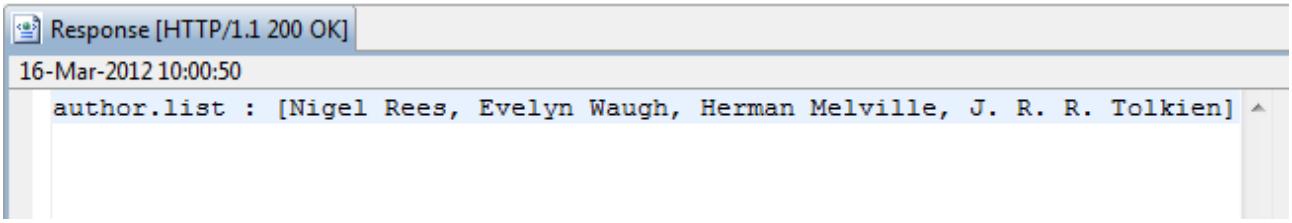
When the extracted attributes are added to the `content.body` message attribute, the following example shows the corresponding request and response message in Oracle API Gateway Explorer:



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`:



The following example shows the corresponding request and response in Oracle API Gateway Explorer:



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.

Use the following XPath:
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.

Store the extracted content:
Select an option to specify how the extracted content is stored. The options are:

- **of the node as text (java.lang.String)**
  This option saves the content of the node retrieved from the XPath expression to the specified message attribute as a `String`.
- **for all nodes found as text (java.lang.String)**
  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.
- **for all nodes found as a list (java.util.List)**
  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.

Extracted content will be stored in attribute named:
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.

Optionally the message payload can be replaced by the extracted content:
Select this option to take the value being set into the attribute and add it to the content body of the response. This option is not selected by default.

Use the following content type for new payload:
This field is only available if the preceding option is selected. This allows you to specify the content type for the response, based on what will be added to the content body.

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** filter can retrieve these attributes and store them in the `attribute.lookup.list` message attribute.

The following SAML attribute assertion contains three attributes, "role", "email", and "dept". The **Retrieve from SAML Attribute Assertion** filter stores all three 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** check box 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.

Retrieve attribute from SAML PDP

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 **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 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`
Bearer	A `<SubjectConfirmation>` is inserted into the SAMLP request.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	Meaning	Value
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 API Gateway user store contains user profiles, including attributes relating to each 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.

General settings

Configure the following field:

Name:
Enter an appropriate name for this filter.

Database settings

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`, select the message attribute containing the value `admin`. The API Gateway then looks up the user in 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 specified by the **User ID** field. To add attributes, click the **Add** button. Similarly, to edit or remove existing attributes, click the **Edit** or **Remove** buttons.

Advanced settings

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. This field is not selected 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 query for multi-valued retrieved attributes using an array syntax (for example, `${user.email[0]}`, or `${user.email[1]}`, and so on). Select this setting to use the legacy format for attribute naming instead.

Prefix for message attribute:
You can specify an optional prefix for message attribute names. The default prefix is `user`.

Fail on empty result set:
Specify whether this filter fails if the result set is empty. This setting is not selected by default.

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.

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 `authentication.subject.password`, which typically stores 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 Client Application Registry, see the *API Gateway OAuth User Guide*.

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 Client Application Registry. 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). For more details, see the Authentication Methods section in the *CA SOA Security Manager Policy Configuration Guide*.

Prerequisites

Integration with CA SOA Security Manager requires CA TransactionMinder SDK version 6.0 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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. For more details, see General settings in the *API Gateway Administrator Guide*.

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 *Manage API Gateway users*.

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 General settings in the *API Gateway Administrator Guide*.

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 *Manage 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 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 *Authentication Repository*.

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 other information relating to the context of the assertion is sent. It is recommended that both the assertion **and** the	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

Method	Meaning	Value
	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 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. Drag and drop this filter on to the policy canvas to configure this filter. This topic describes 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. To add a Kerberos Service, right-click the **Kerberos Services** node, and select **Add Kerberos Service**. Alternatively, you can add Kerberos Services under **External Connections** in the Policy Studio tree. For more details, see the *Kerberos Services*.

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 if you expect the Kerberos Service ticket to be contained in the message. You must enter an XPath expression to point to the expected location of the Kerberos token. You can select some default expressions that point to common locations from the list. Otherwise you can add a new XPath expression by clicking the **Add** button. Similarly, existing XPath expressions can be configured by clicking **Edit** and **Delete**. For more details, see *Configuring XPath Expressions*.
- **Message Attribute**:
  When using the **WS-Trust for SPNEGO** standard above, the **Consume WS-Trust** filter places 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 generates a new symmetric key and wraps 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 (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.

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 settings 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`, `ServicePrincipalName`, 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 STS response 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 attribute type 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*.

You can configure SSL settings, credential profiles for authentication, and other settings for the direct connection using the tabs in the **Connection Details** group. For more details, see the *Connect to URL* topic.

Delegate to Routing Policy:
Select this option 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 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:

Canonicalization Algorithm	When selected, additional elements are added to the RST, which specify a client-requested canonicalization 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.

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

Integration with RSA Access Manager requires the RSA ClearTrust SDK, version 6.0. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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 (**Authorization Server** or **Dispatcher Server**). 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 pre-configured Server Name in Access Manager.
- **Resource**:
  Enter the name of the requested resource. This resource must be 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*.

Attribute Authorization

Overview

The purpose of the filters in the **Attributes** filter group is to extract user attributes from various sources. You can 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 dialog.

For attribute checks involving attributes extracted from a SAML attribute assertion, you must specify the namespace of the attribute as 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.

Axway PassPort Authorization

Overview

Axway PassPort provides a central repository, identity broker, and security audit point for Axway Business-to-Business Integration (B2Bi) or Managed File Transfer (MFT) solutions. Axway PassPort centralizes and simplifies provisioning and management for your entire online ecosystem, enabling secure collaboration between applications, divisions, customers, suppliers, and regulatory bodies.

Axway Component Security Descriptor (CSD)
An *Axway Component Security Descriptor* (CSD) file is an XML file that defines resources, privileges, and roles for each component. For more details, see "Component Security Descriptor files" in the *Axway PassPort 4.6 Administrators Guide*. The **Axway PassPort Authorization** filter checks if the specified user has the privileges to perform the action on the specified resource. The CSD file defines the available actions that a resource supports. It may also define privileges and roles, which can also be created in the PassPort administration user interface.

This topic explains how to configure the settings in the **Axway PassPort Authorization** filter, which are used to configure integration between Axway PassPort and the API Gateway.

Configuration

Specify the following settings to configure the **Axway PassPort Authorization** filter:

Name:
Enter an appropriate name for this filter.

User ID:
Enter the ID of the user to authorize. You can enter a static name or a selector that specifies a message attribute. The selector is expanded at runtime to the value of the message attribute. Defaults to `${authentication.subject.id}`.

Resource:
Enter the name of the resource for which the user is seeking authorization. This resource must have been defined in the `<ResourceDefinition>` in the PassPort repository CSD. You can enter a static name or a selector that specifies a message attribute. The selector is expanded at runtime to the value of the message attribute. Defaults to `${http.request.uri}`.

Action:
Enter the action being performed on the resource for which authorization is sought. This action must have been defined in the `<AvailableActions>` section of the PassPort repository CSD. You can enter a static name or a selector that specifies a message attribute. The selector is expanded at runtime to the value of the message attribute. Defaults to `${http.request.verb}`.

PassPort Repository:
Select an existing connection to an Axway PassPort repository to use for authorization. To configure a connection in the Policy Studio tree, select **External Connections** > **Authentication Repository Profiles**, right-click **Axway PassPort Repository**, and select **Add a new Repository**. For more details, see *Axway PassPort Authentication Repository*.

Further Information
For more details on specifying settings as selectors, see *Selecting configuration values at runtime*.

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

Integration with CA SOA Security Manager requires CA TransactionMinder SDK version 6.0 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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.

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.

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`

Method	Meaning	Value
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 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.	`urn:oasis:names:tc:SAML:1.0:cm:bearer`

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**.

Management Services 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 on RBAC, see the *API Gateway Administrator Guide*.

The **Management Services 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

Configure the following settings:

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 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 following settings on the **SAML Subject** tab describe the *subject* of the SAML assertion:

- **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 is 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 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.

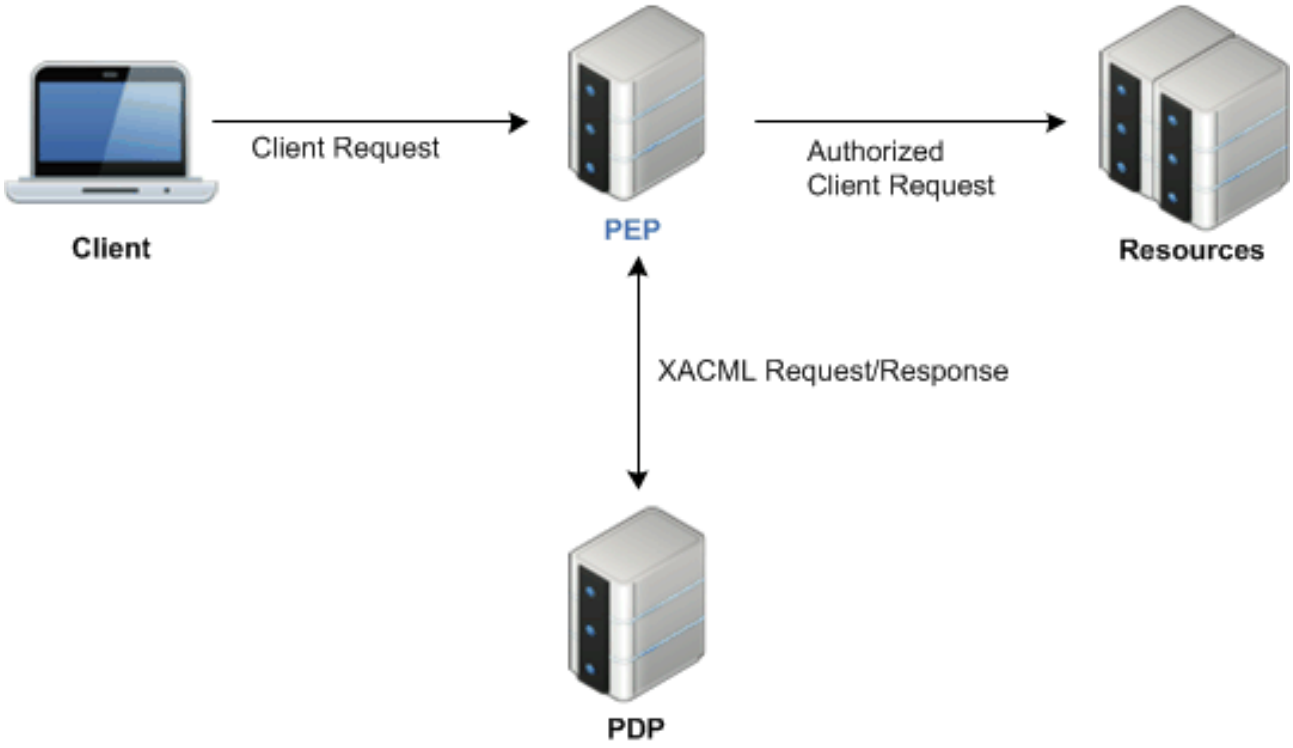
Tivoli Configuration Files:
A Tivoli configuration file that contains all the required connection details is associated with a particular Oracle API Gateway instance. Click the **Settings** button to display the **Tivoli Configuration** dialog.

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.

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 (wants access to the resource). The `Subject` element can contain multiple `Attribute` elements 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 contains 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 is 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, select a policy in the **XACML request security policy** dialog, and click **OK**.
XACML Response Security	Click the browse button, 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:
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}`. For more details on selectors, see *Selecting configuration values at runtime*.

You can configure SSL settings, credential profiles for authentication, and other settings for the direct connection using the tabs in the **Connection Details** group. For more details, see the *Connect to URL* topic.

Delegate routing to the following policy:
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 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

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.0-sp1-cr005 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.0-sp1-cr005 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.0-sp1-cr005 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.0-sp1-cr005 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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.

Static CRL certificate validation

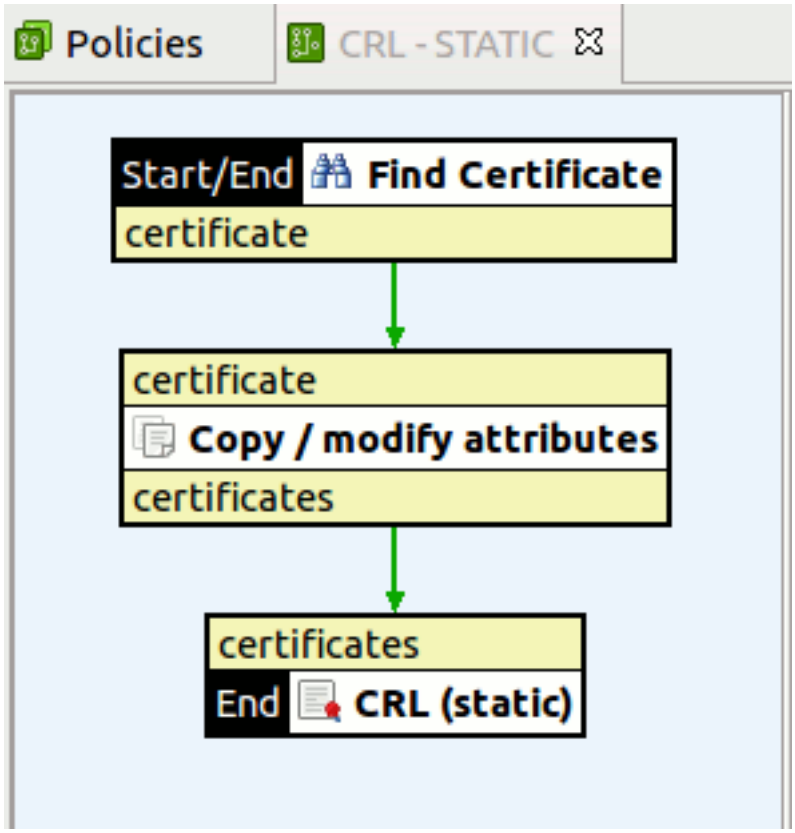
Overview

A Certificate Authority (CA) may wish to publish a Certificate Revocation List (CRL) to a file. In such cases, API Gateway can load the revoked certificates from the file-based CRL and validate user certificates against it.

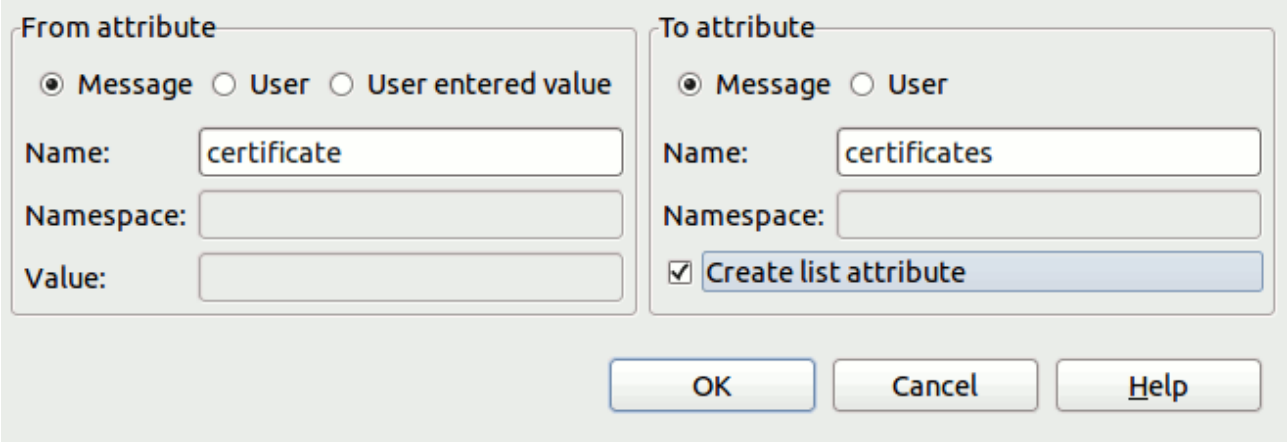
Because the CRL is typically signed by the CA that owns it, the certificate of the CA that issued the CRL *must* be imported into the certificate store before this filter can work correctly. In addition, the **CRL (Static)** filter requires the `certificates` message attribute to be set by a preceding 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:



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 window.

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)** window:

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. 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 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 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** window. API Gateway checks the CRL of all selected LDAP directories to validate the client certificate. The filter fails as soon as 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 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** window:

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** filter 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** window:

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, 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 (for example, `user_certificate`). The certificate can be extracted from this attribute by a successor filter in the policy.

Name:
Enter an appropriate name for the filter.

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 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 API Gateway 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 API Gateway 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 check box. 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, 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 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.

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 *Manage certificates and keys* topic.

Configuration

You can configure the following settings on the **Certificate Chain Check** window:

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 check box 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 client

Overview

You can use the Online Certificate Status Protocol (OCSP) to retrieve the revocation status of a certificate, as an alternative to retrieving Certificate Revocation Lists (CRLs).

You can use the **OCSP Client** filter to retrieve certificate revocation status from an OSCP responder, such as Axway Validation Authority. The input to this filter is the certificate to be checked. The message attribute containing the certificate is user defined. The output of this filter is:

- `True` if the certificate status is good
- `False` if the certificate status is revoked or unknown (or if an exception occurs)

General settings

Configure the following general settings on the **OCSP Client** dialog:

Name:
Enter a suitable name for this OCSP client filter.

OCSP Responder URL:
Enter the URL of the OCSP responder.

Message settings

Configure the following OCSP message settings on the **Settings** tab:

The message attribute storing the certificate to validate:
Enter the name of the attribute that contains the certificate to be checked. The default is `${certificate}`.

The key to sign the request:
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 the OCSP responder.

You can select a specific certificate from the certificate store in the dialog, or click **Create/Import** to create or import a certificate. Alternatively, you can specify a certificate to bind to at runtime using an environment variable selector (for example, `${env.serverCertificate}`). For more details on selectors, see *Selecting configuration values at runtime*.

Validate response:
Select the **Do not validate response** option to disable response validation. The response from the OCSP responder is not validated when this option is selected.

Select the **Validate response** option to enable response validation. Click one or more of the following options to specify how the response from the OCSP responder is validated:

- **Against the certificate contained in the response**:
  The response is validated against the certificate contained in the response. This option is selected by default.
- **Against the CA certificate of the certificate being validated**:
  The response is validated against the CA certificate of the certificate being validated. This option is selected by default.
- **Against the specified certificate**:
  Click **Signing Key** to choose a certificate from the certificate store or to specify a certificate to bind to at runtime.

You can select any combination of these options. If multiple options are selected, the filter continues as soon as the re-

sponse is successfully validated against one of the selected options.

In the **Allowable time difference in seconds between this system and time stamp on received responses** field, enter a value in seconds. You can use this field to allow for drift on server and client machines. It validates against the value `producedAt` in the OCSP response. The default value is 300 (5 minutes). This value is only validated if the **Validate response** option is selected.

Use nonce to prevent reply attack:
Select this option to include a nonce in the request. This is a randomly generated number that is added to the message to help prevent reply attacks.

Store results of certificate status in:
Click the browse button to select the cache in which to store the certificate status result. 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 *Global caches* topic.

Storing the certificate status in the cache enables the certificate status to be retrieved without having to return to the OCSP responder.

Routing settings

You can configure the settings for routing the OCSP request to the OCSP responder on the **Routing** tab.

You can configure SSL settings, credential profiles for authentication, and other settings for the connection using the **SSL**, **Authentication**, and **Settings** tabs. For more details, see the *Connect to URL* topic.

Advanced settings

On the **Advanced** tab, you can enable a specific policy to run after the message is created, or after the response is received.

Configure the following advanced settings:

Run this policy after the message has been created:
Click the browse button to select a policy to be run after the message has been created.

Run this policy after a response has been received:
Click the browse button to select a policy to be run after a response has been received.

Record outbound transactions:
Select this option to enable recording of outbound transactions under traffic monitor. This field is not selected by default. For more information, see the *API Gateway Administrator Guide*.

Integration with Axway Validation Authority

When using the **OCSP client** with Axway Validation Authority (VA) as an OCSP responder, you can use the following trust models:

- **Direct trust**
  In this model, OCSP responses are signed with the OCSP signing certificate of the VA server. The signing certificate is not included in the OCSP response.
- **VA delegated trust**
  In this model, the signing certificate is included in the OCSP response. API Gateway might not have this certificate. If not, it must have the issuer (CA) certificate of the signing certificate.

You can import certificates into API Gateway's trusted certificate store under the **Certificates and Keys** node in the

Policy Studio tree. For more information, see the *Manage certificates and keys* topic.

Note
A complete documentation set for Axway Validation Authority is available on the Axway Support website: https://support.axway.com.

Validate certificate store

Overview

The **Validate Server's Certificate Store** filter checks 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 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 API Gateway.

Configuration

Configure the following fields:

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 API Gateway's certificate store. This is selected by default.

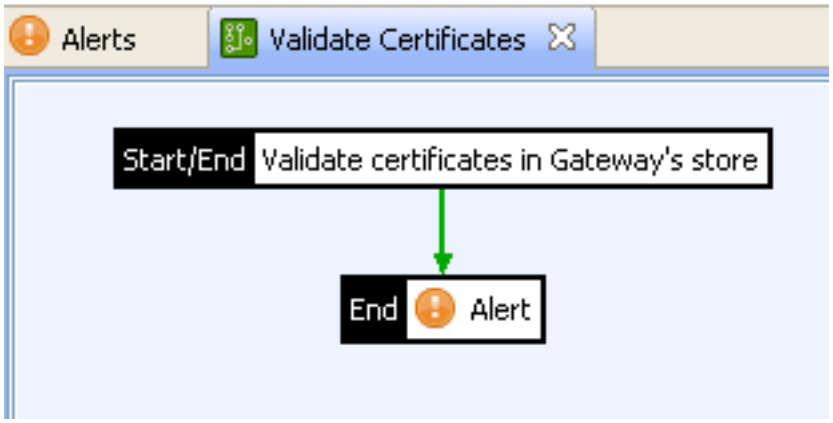
Check Server's Java Keystore:
Select whether to check the certificates in API Gateway's Java Keystore. This is not selected by default. When selected, you must enter the password for this keystore. The default password is `changeit`.

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:

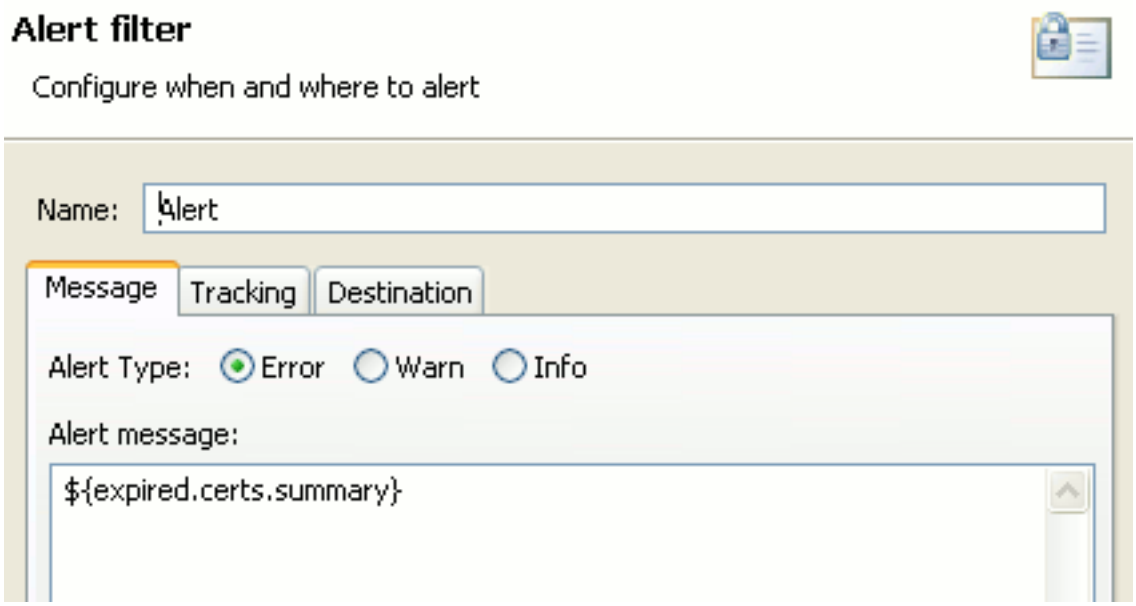
Keystore Location	Specify the path to this keystore (for example, `/home/oracle/osr-client.jks`).
Password	Enter the password for this keystore.

Deployment example

The following example shows a **Validate Certificates** policy that includes a **Validate Server's Certificate 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:



You must also select a preconfigured email alert destination on the **Destination** tab (for example, **Email API Gateway Administrators**). For more details on configuring email alert destinations, see the *Configure system alerts* 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 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:

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. API Gateway can query an XKMS responder to determine whether a given certificate can be trusted.

Configuration

Configure the following fields:

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.

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 API Gateway 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.

Scan with ClamAV anti-virus

Overview

You can use the **ClamAV Anti-Virus** filter to 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 **Content Type** filter that blocks multipart messages with parts of type `image/jpeg`.

Allow or deny content 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 those types here. Multipart messages containing the selected MIME or DIME type parts 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.

Configure MIME/DIME types

The **MIME/DIME Settings** dialog enables you to configure new and existing MIME types. When a type is 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.

Configure 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.

Configure 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.

Configure 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 the section called “Configure 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**.

Send to ICAP

Overview

You can use an **ICAP** filter to send a message to a preconfigured 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:

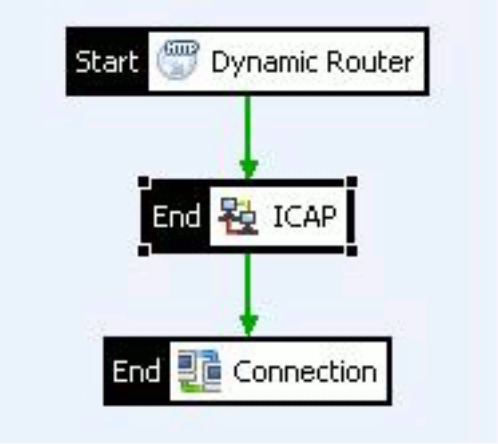
Name:
Enter an appropriate name for the filter.

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

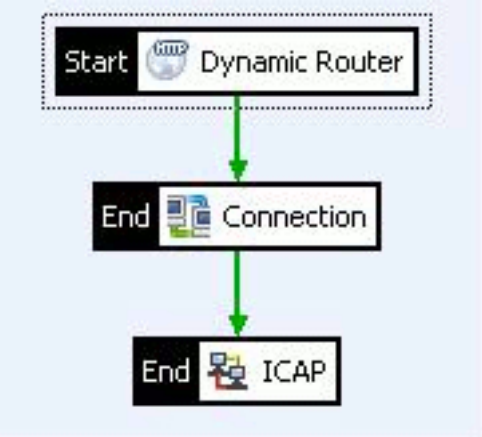
This section shows some example use cases of the **ICAP** filter configured in policies.

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*.

Scan with 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 5600 Scan Engine. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

Add McAfee binaries to API Gateway

To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Add McAfee binaries to Policy Studio

To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart Policy Studio.

Configuration

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.

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.

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.

Load 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
```

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 filtering

Overview

It is sometimes useful to filter incoming messages based, not only on the internal content of the message, but on external characteristics of the message such as size. You can use the **Message Size** filter to configure the API Gateway to reject messages that are greater or less than a specified size.

Note

You should not use the **Message Size** filter on HTTP `GET` requests

Configuration

To configure the API Gateway to block messages of a certain size, complete the following fields:

At least:
Enter the size (in bytes) of the smallest message that should be processed in the field. Messages smaller than this size will be rejected.

At most:
Enter the size (in bytes) of the largest message that should be processed. Messages larger than the size entered here will be rejected.

Use in Size Calculation:
Select one of the following options to specify the portion of the message that is to be used when calculating the size of the message:

- **Root body only**: The API Gateway calculates the size of the message body excluding all other MIME parts (attachments).
- **Attachments only**: The API Gateway only calculates the size of all attachments to the message. This excludes the size of the root body payload from its calculation. The **Message Size** filter still works even when there are no attachments.
- **Root body and attachments**: The API Gateway includes 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.

Configure 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.

Configure 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.

Configure 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 the section called “Configure query string 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 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

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 Policy Studio. Drag and drop the filter on to a policy to perform schema validation.

General settings

Configure the following general settings:

Name:
Enter an appropriate name for the filter.

Selecting the schema

To configure the XML schema to validate messages against, click the **Schema to use** tab. You can select to use either a schema from the WSDL for the current Web service, or a specific schema from the global cache of WSDL and XML schema documents.

Select one of the following options:

Use Schema from WSDL of Web service:
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 message with:
Select this option to use a schema from the global cache. This is the default option. Click the browse button, and select a schema from the tree view. The XML schemas under both the **XML Schema Document Bundles** and the **WSDL Document Bundles** nodes can be selected. Click the check box next to the schema document to select the schema. You can also select a particular version of a schema by clicking the check box next to the version information.

To add a new schema, right-click the **XML Schema Document Bundles** tree node, and select **Add Schema**. Alternatively, you can add schemas under the **Resources** node in the Policy Studio tree. For more details on configuring schemas, see the *Manage WSDL and XML schema documents*

Tip

If you have a WSDL file that contains an XML schema, you can use this schema to validate messages by importing the WSDL file into the Web service repository. The **Import WSDL** wizard automatically adds any XML schemas contained in the WSDL to the global cache under the **Resources > WSDL Document Bundles** node. For more details on importing WSDL files, see the *Configure policies from WSDL files* topic.

Selecting which part of the message to match

To configure which part of the message to validate, click the **Part of message to match** tab.

A portion of the XML message can be extracted using an XPath expression. API Gateway can then validate this portion against the specified XML schema. For example, you might need to validate only the SOAP Body element of a SOAP message. In this case, enter or select an XPath expression that identifies the SOAP Body element of the message. This portion should then be validated against an XML schema that defines the structure of the SOAP Body element for that particular message.

Click the **Add** or **Edit** buttons to add or edit an XPath expression using the **Enter XPath Expression** dialog. To remove an expression select the expression in the **XPath Expression** field and click the **Delete** button.

You can configure XPath expressions manually or using a wizard. For more details, see the *Configuring XPath Expressions* topic.

Advanced settings

The following settings are available on the **Advanced** tab:

Allow RPC Schema Validation:
When the **Allow RPC Schema Validation** check box 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 might 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 check box 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 element 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 API Gateway (for example, using the **Extract MTOM Content** 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 an `<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 schema validation fails.

To resolve this issue, select the **Inline MTOM Attachments into Message** check box 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 API Gateway 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 (for more information, see the *Set Message* topic) 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:

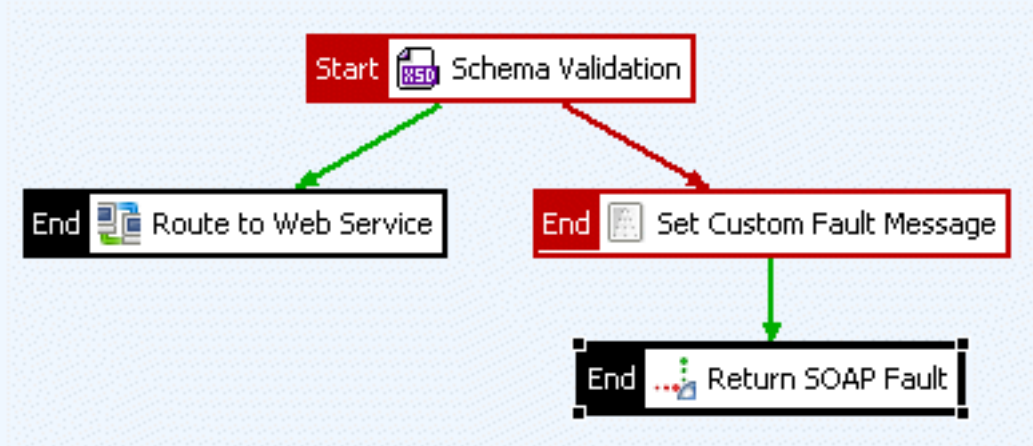
```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
        <env:Subcode>
          <env:Value xmlns:fault="http://www.Oracle.com/soapfaults">
            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** filter (for more information, see the *Reflect Message Filter* topic). 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 figure 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.

Generate 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/.

Scan with 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

Integration with Sophos requires Sophos SAV Interface version 4.8. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

API Gateway
To add 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 *`install-dir`*`/apigateway/ext/lib` directory.
   - Add `.dll` files to the *`install-dir`*`\apigateway\Win32\lib` directory.
   - Add `.so` files to the *`install-dir`*`/apigateway/platform/lib` directory.
2. Restart the API Gateway.

Policy Studio
To add third-party binaries to Policy Studio, you must perform the following steps:

1. Select **Windows** > **Preferences** > **Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart 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 settings

To configure the **Scanning Details** tab, 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 type settings

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 enables to limit the number of requests that pass in a specified time period. For example, this enables you to enforce a specified message quota or *rate limit* on a client, and to protect a backend service from message flooding. You can configure this filter to allow only a specified number of messages from a specified client over a configured time frame through to a virtualized API. If the number of messages exceeds the specified limit, the filter fails for the excess messages.

Important

The **Throttling** filter 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 on message 21, but passes for the first 20 incoming messages.

When the configured constraints are breached, the API Gateway behavior is determined by the filter next in the policy failure path from the **Throttling** filter. Typically, an **Alert**, **Trace**, or **Log** filter is configured as the successor filter in the failure path.

An example use case for this filter would be to enforce a message rate limit specified in a contract agreed with a customer (for example, the customer has purchased a maximum of 100 messages per client per hour only). Another example use case would be to protect a service that can handle a maximum of only 20 messages per client per second. If the filter detects a higher number of incoming requests, it blocks the messages.

Rate limit settings

Configure the following rate limit settings:

Name:
Enter an appropriate name for the filter.

Allow:
Specifies the number of messages allowed in the time interval specified in the **Messages every** field. If the API Gateway receives more than the specified number of messages during the specified time interval, the filter fails. Otherwise, the filter passes.

Messages every:
Specifies the allowed time interval. If the API Gateway receives more than the number of messages in the **Allow** field in the time interval specified, the filter fails. Otherwise, the filter passes. The time interval depends on the value selected in the drop-down box on the right (seconds, minutes, hours, days, or weeks). For example, if you enter `10` in the **Messages every** field, and select `Minutes` from the drop-down list, the time interval lasts 10 minutes.

Note

The specified time period starts when a message is received, and lasts for that time period (for example, 10 minutes). When the time period is over, the message count is reset, and the counter starts again when another message is received. If you select `Day` or `Week` from the drop-down list, you must configure the **When do days/weeks start** fields on the **Advanced** tab.

Rate limit based on:
Select this setting if you wish to configure the API Gateway to keep track of request messages based on a specified key value (for example, to track rates based on client IP address). All rate limits are stored in a cache, and a key is required to look up the cache. The key can be any value that identifies the message sender. For example, this includes the au-

thenticated subject, the client IP address, or even a combination of API method name and IP address. If you do not specify a key, the cache will contain a single entry for the filter, and the associated rate limit is used each time the filter is invoked.

To track rate limits based on the client IP address, use the following setting:

```
${http.request.clientaddr.getAddress()}
```

The value entered can also be a combination of a fixed string value and/or an API Gateway message attribute selector. For example, you could use the following value to keep track of the number of times an API named `StockQuote` is requested by an authenticated subject:

```
StockQuote-${authentication.subject.id}
```

Store rate limits in cache:
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 API Gateway instances, you can configure a distributed cache to cache request messages.

For example, you wish to prevent a burst of more than 50 messages per second from reaching a backend service. Assume that a load balancer is deployed in front of two API Gateway instances, 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 therefore 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*.

Advanced settings

Configure the following advanced settings:

A day starts on the following hour:
You must configure this field if you select either `Day` or `Week` from the drop-down list when configuring the **Messages every** field. For example, if you select `Day`, 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.

A week starts on the following day:
You must configure this field only if you select `Week` from the drop-down list when configuring the **Messages every** field. For example, if you select `Week` and `00:00`, and enter `Sunday` in this field, this means that the time period starts next Sunday at midnight, and lasts for one week exactly. The time period is reset on midnight of the next Sunday.

Include remaining limit in HTTP response headers:
Specifies whether to include the following `X-Rate-Limit` headers in the HTTP response message:

`X-Rate-Limit-Limit`	Rate limit ceiling for the given request (for example, `100` messages).
`X-Rate-Limit-Remaining`	Number of requests left for the time window (for example, `45` messages).
`X-Rate-Limit-Reset`	Remaining time window before the rate limit resets in UTC epoch seconds (for example, `1353517297`).

Begin rate limit HTTP Headers with:

Specifies the string used to begin the name of rate limit HTTP headers. Defaults to `X-Rate-Limit-`. For example, if you specify a value of `My-Corp-Quota-`, the following HTTP headers are inserted into the response message:

- `My-Corp-Quota-Limit`
- `My-Corp-Quota-Remaining`
- `My-Corp-Quota-Reset`

Use multiple throttling filters

To use two or more **Throttling** filters to maintain separate message counts, you must use a different rate limit value for each filter, or use different caches for each filter:

- **Use a different rate limit per filter**:
  With this approach, you can use a unique **Rate limit based on** value in each **Throttling** filter. The easiest way to do this is to prepend the `${http.request.clientaddr.getAddress()}` selector value with the filter name, for example:

  ```
  My Corp Quota Filter ${http.request.clientaddr.getAddress()}
  ```

  This ensures that each filter maintains its own separate message count in the selected cache.
- **Use 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 selector expression

Overview

The **Validate Selector Expression** filter can use regular expressions to check values specified in selectors (for example, message attributes, Key Property Store (KPS), or environment variables). This enables you to make decisions on what to do with the message at runtime. Filters configured in a policy before the **Validate Selector Expression** filter can generate message attributes and store them in the message. For example, you could use the the **Validate Selector Expression** filter to specify that if the attribute value is `X`, route the message to service `X`. For more details on selectors, see *Selecting configuration values at runtime*.

You can configure the following sections on the **Validate Selector Expression** screen:

- **Enter Regular Expression**:
  You can configure selectors that are 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 selector is acceptable. For example, if you know that a message attribute-based selector named `${my.test.attribute}` must have a value of `ABCD`, a regular expression of `^ABCD$` 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 configured selector. 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.

Configure selector-based regular expressions

The **Enter Regular Expression** table displays the list of configured selectors, together with the **White list** of regular expressions that restrict their values. For this filter to run successfully, *all* configured selector checks must have values matching the configured regular expressions.

The **Selector** column shows the name configured for the selector. The **Regular Expression** column shows the name of the regular expression that the API Gateway uses to restrict the value of the named selector. A number of common regular expressions are available from the global **White list** library.

Configure a Regular Expression

You can configure regular expressions by clicking **Add**, **Edit**, or **Delete**. 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 selector in the **Selector Expression** field (for example, `${my.test.attribute}`).
2. Select whether this attribute is **Optional** or **Required**. 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).
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**.

The **Advanced** section enables you to extract a portion of the attribute value that is run against the selector. The extrac-

ted substring can also be Base64 decoded if necessary.

Threatening content regular expressions

The regular expressions entered in this section guard against message attributes 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 section called “Configure selector-based regular expressions”, 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.

Click **Add** to add threatening content regular expressions. You can edit or remove existing expressions by selecting them in the list, and clicking **Edit** or **Delete**. You can enter 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 ASCII control characters in attribute values.

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 **Add** or **Edit**. Enter a **Name** for the expression followed by the **Regular Expression**.

For more details on regular expressions, see the following topics:

- *Query string validation*
- *HTTP header validation*

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 path parameters against a set of restrictive conditions called a *request parameter restriction*
- Each of the query string parameters against 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 Policy Studio. For details on how to create a REST request, see the *Create REST Request* filter.

General settings

Complete the following general settings:

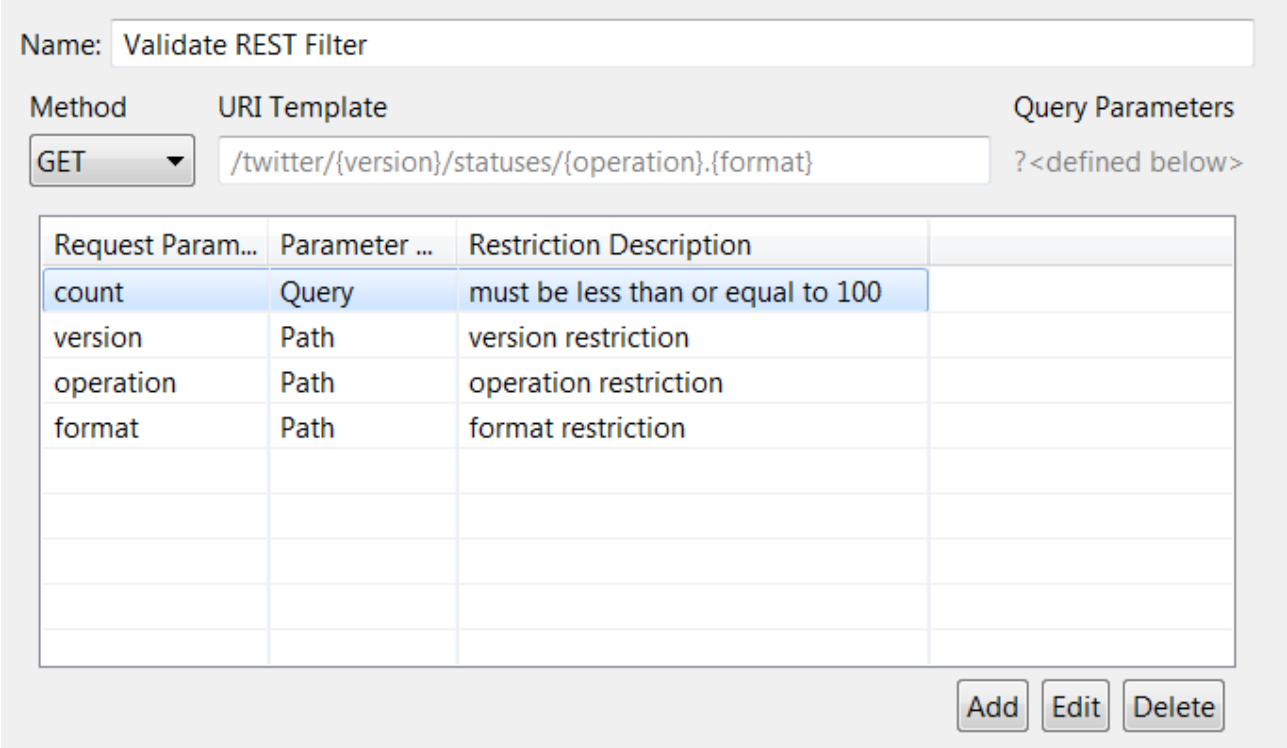
Name:
Enter an appropriate name for the filter.

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.

URI Template:
Enter the URI template of the inbound path. You can include parameters in the path (for example, `/twitter/{version}/statuses/{operation}.{format}`). For more information, see the section called "URI path templates". Any path parameters are automatically added to the table of parameter restrictions below. The path of the incoming request must match the path specified here, and the path parameters must meet the restrictions defined in the parameter restrictions table.

Query Parameters:
You can define query string parameters and restrictions in the parameter restrictions table below. The query string parameters must meet the restrictions defined in the parameter restrictions table. For more information, see the section called "Adding REST request parameter restrictions".

The following figure shows the parameter restrictions table for the path `/twitter/{version}/statuses/{operation}.{format}` and the query string parameter `count`:



Fail if unspecified query string parameters found:
Select this option to have this filter fail if a request parameter is found on the incoming query string that has not been specified in the parameter restrictions table. This option is selected by default. 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);`).

Extract valid parameters into individual message attributes:
Select this option to store valid parameters in message attributes. This option is not selected by default.

On failure save invalid parameter name as a message attribute:
Select this option to save the invalid parameter name as a message attribute if the filter fails. This option is not selected by default.

Adding REST request parameter restrictions

Click the **Add** button to configure restrictions on the values of query string parameters. Click the **Edit** button to configure restrictions on the values of path parameters that have been automatically added to the table. You can configure the following settings in the **REST Request Parameter Restriction** dialog:

Request Parameter Name:
Enter the name of the parameter to validate (for example, `customer_name`). This field is prefilled for path parameters. This is displayed in the table of parameter restrictions.

Description:
Enter a description of the restriction (for example, `Name parameter must be a string no longer than 10 characters`). This field is prefilled for path parameters. This is displayed in the table of parameter restrictions.

Data Type:
Enter or select the data type of the parameter (for example, `string` or `integer`). The default type is `string`.

Fail if request parameter not found:
Select this option if the specified request parameter must be present in the request. The filter fails if the parameter is not found. This option is not selected by default for query string parameters, but is selected by default for path parameters.

Complete the following fields on the **Request Parameter Restrictions** tab:

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 of Allowed Values	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.

Complete the following fields on the **Advanced Restrictions** tab:

Greater than	Specifies that the value entered in the **Minimum Value** field represents an exclusive lower bound (the value must be 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 be greater than or equal to this).
Minimum Value	Specifies the lower bounds for numeric values (for example, the value must be greater than 20).
Less than	Specifies that the value entered in the **Maximum Value** field represents an exclusive lower bound (the value must be 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 be less than or equal to this).
Maximum Value	Specifies the upper bounds for numeric values (for example, the value must be 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 ignored.
Whitespace	Specifies how white space is handled (for example, line feeds, tabs, spaces, and carriage returns). You can select 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).

URI path templates

Paths are specified using a formatted Jersey `@Path` annotation string. The `@Path` annotation's value is a relative URI path (for example, `/twitter`). The Jersey `@Path` annotation enables you to parameterize the path specified in the incoming request, by embedding variables in the URIs.

URI path templates are URIs with variables embedded within the URI syntax. These variables are substituted at runtime in order for a resource to respond to a request based on the substituted URI. Variables are denoted by curly braces (for example, `/twitter/{version}`).

The following is an example URI template:

```
/twitter/{version}/statuses/{operation}.{format}
```

It contains the variables `version`, `operation`, and `format`. At runtime, this URI template might resolve to:

```
/twitter/1.1/statuses/retweets_of_me.json
```

For more information on Jersey `@Path` annotations, go to https://jersey.java.net/.

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.

Selector Expression to Retrieve Timestamp:
Enter the name of the selector expression that contains the value of the timestamp. Defaults to `${timestamp}`. The specified selector is expanded at runtime to the corresponding message attribute value. For more details, see *Selecting configuration values at runtime*.

Note

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).

Verify the WS-Policy security header 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 the **WS-SecurityPolicy Layout** 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 filtering* 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*.

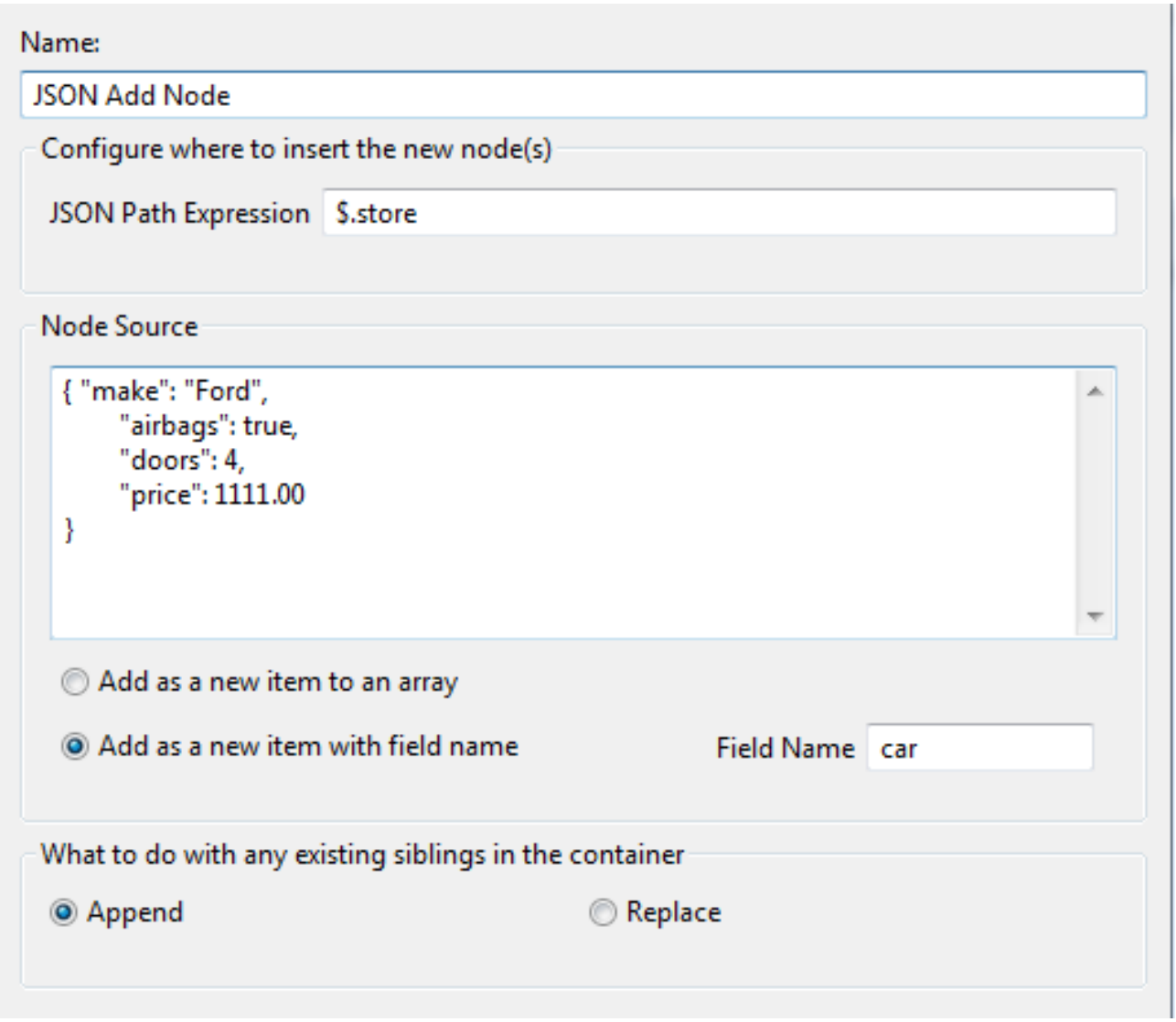
What to do with any existing siblings in the container:
Select one of the following options to determine where the new node is placed relative to the node(s) returned by the JSON Path expression:

- **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:



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
    }
  }
}
```

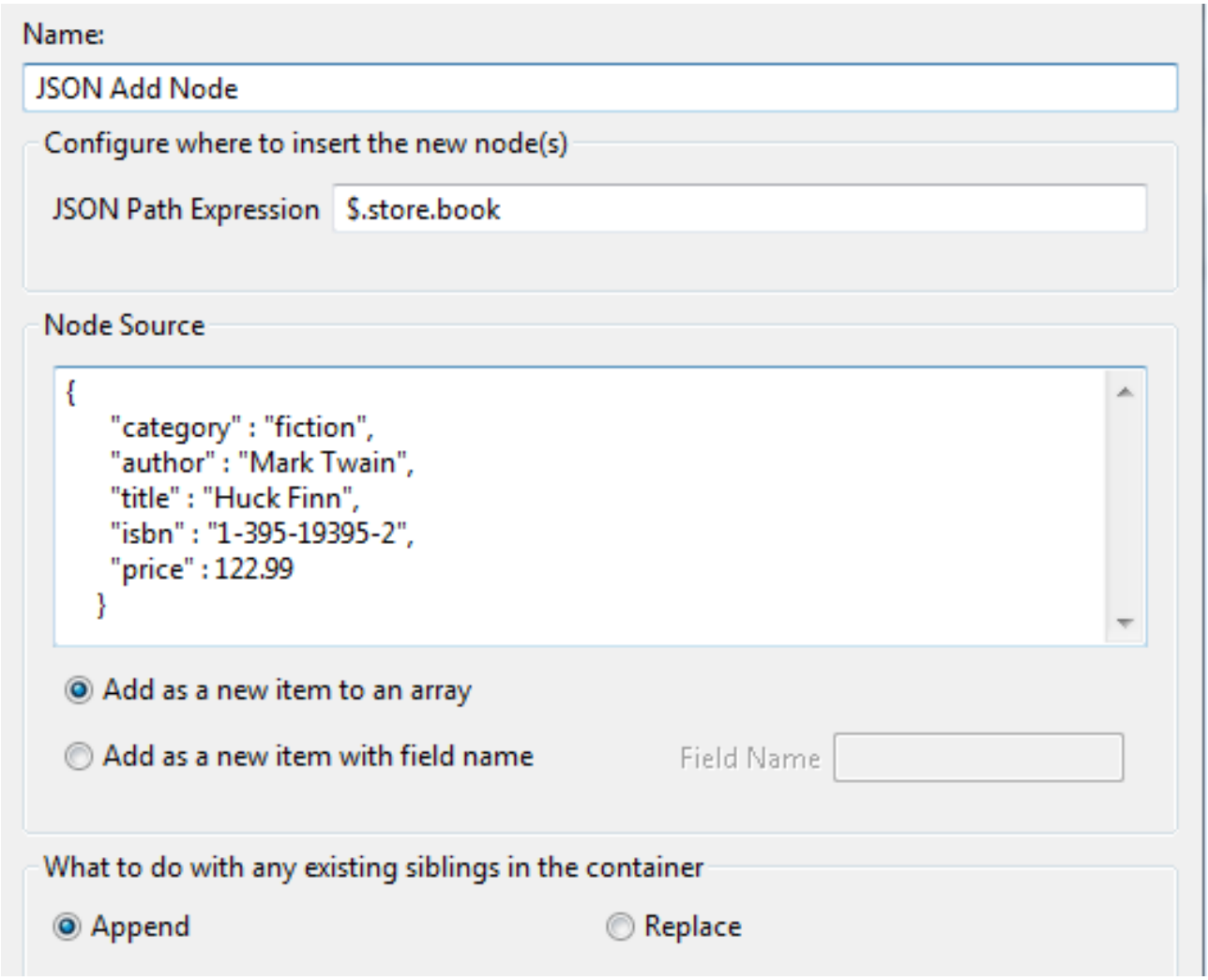
Response [HTTP/1.1 200 OK]

16-Mar-2012 09:46:27

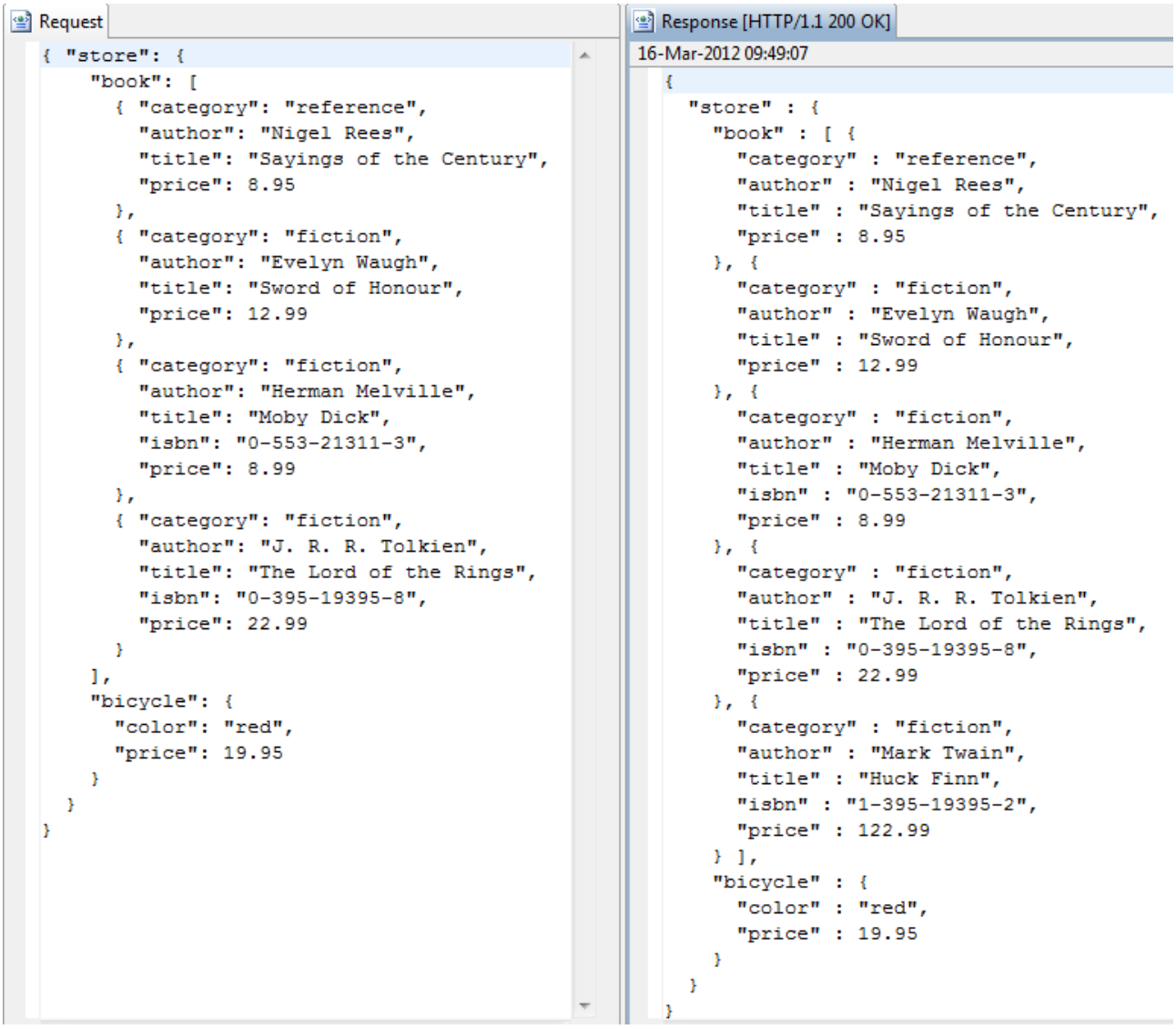
```
{
  "store" : {
    "book" : [ {
      "category" : "reference",
      "author" : "Nigel Rees",
      "title" : "Sayings of the Century",
      "price" : 8.95
    }, {
      "category" : "fiction",
      "author" : "Evelyn Waugh",
      "title" : "Sword of Honour",
      "price" : 12.99
    }, {
      "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

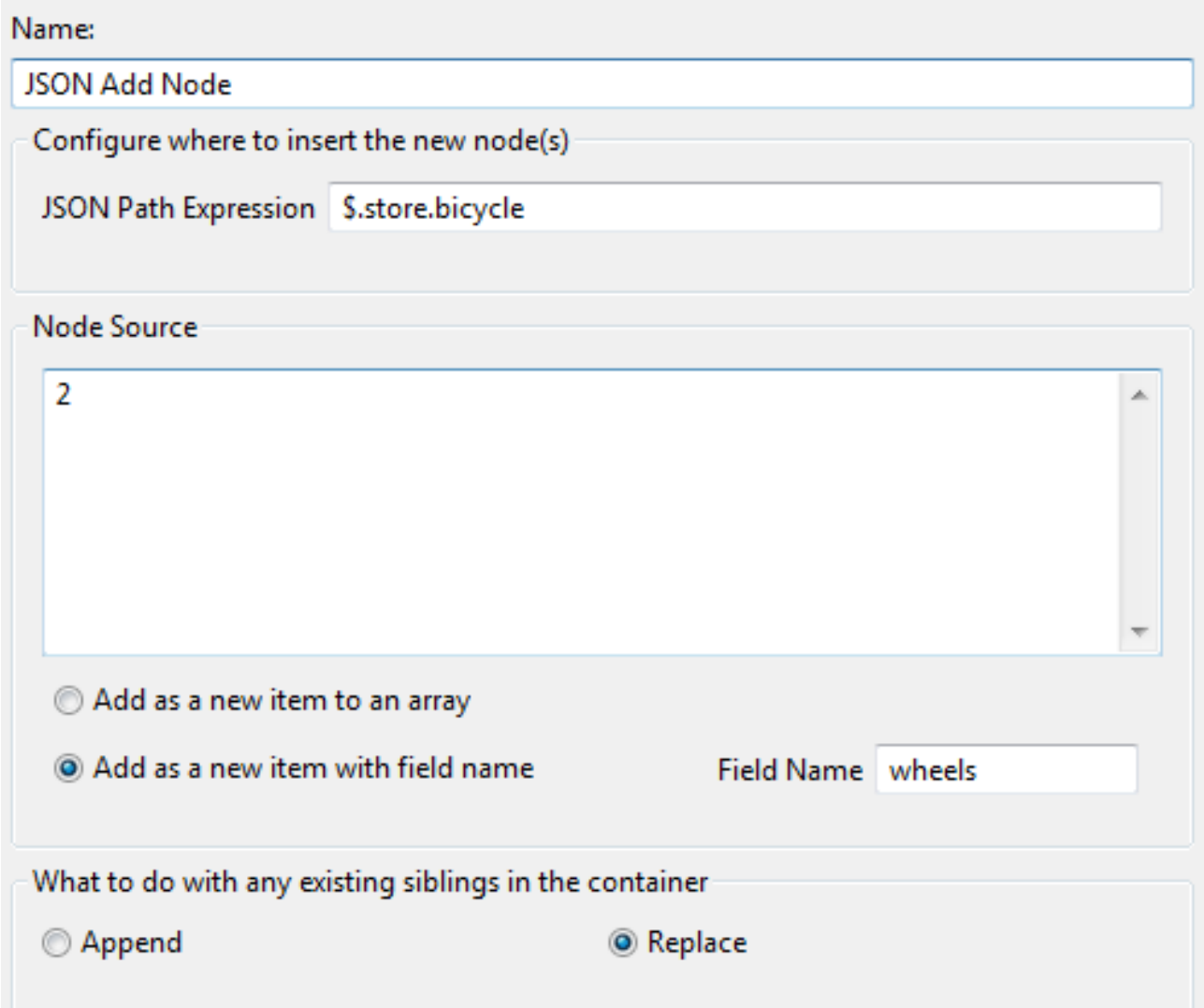
The following example shows the settings required to add a book to an array:



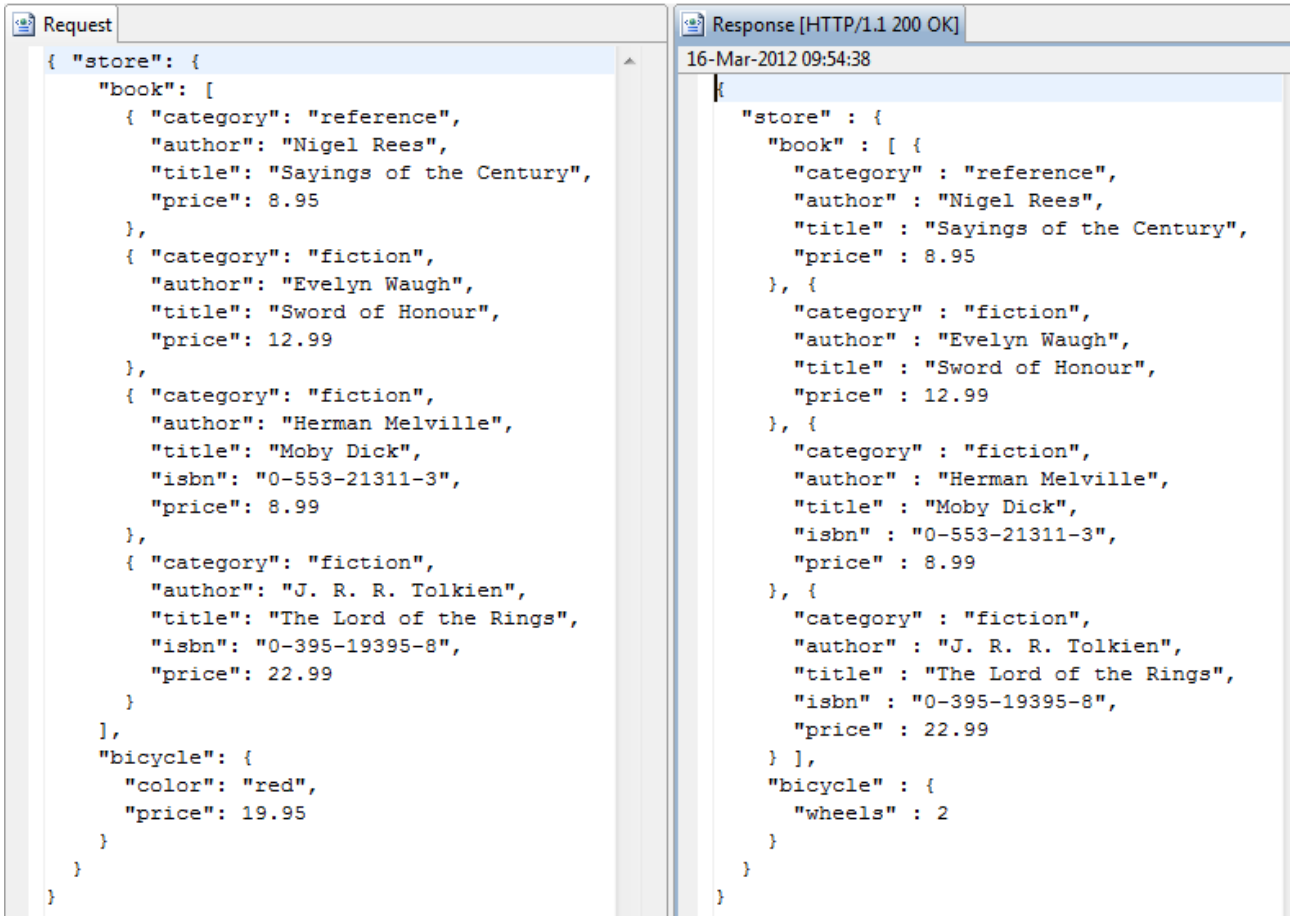
The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



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:



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 09:54:38
{
"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, you might use the **Load File** filter in cases where an external application drops XML or JSON files on to the file system to be validated, modified, and potentially routed on over HTTP, JMS, or 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 range of message processing filters available in the API Gateway. This ensures that only properly validated messages are routed on to the target system.

Input Settings

Configure the following field:

File:
Enter the name of the file to load, or browse to the file in the file system. This setting is required.

Processing Settings

The fields in this section determine what processing is performed on the input files, and where files are placed before and after processing.

Processing Directory:
Enter or browse to the directory to which the input file is copied prior to processing. This field is optional. If this is not specified, the input file remains in the current input directory.

Response Directory:
Enter or browse to the directory to which the response file is copied. This field is optional. If this is not specified, the response file is not written to disk.

Processing Policy
Select the policy executed on the input file. For example, the policy could perform message validation, routing, virus checking, or XSLT transformation. This field is optional.

File Type:
Specifies how the input file is interpreted. Select one of the following options:

- **Raw**:
  Assumes a content-type of `application/octet-stream`. This is the default.
- **Treat as HTTP Message (including headers)**:
  Assumes the inbound file contains an HTTP request (optionally with HTTP headers).
- **Infer content-type from extension**:
  Performs a lookup on configured MIME/DIME types to determine the content-type of the file based on its extension.
- **Use Content-type**:
  Enables you to specify a content-type in the textbox.

On Completion Settings

You can specify what to do when the file processing has completed. Select one of the following options:

- **Do Nothing**:
  The input file remains in the input directory or in the **Processing Directory**. This is the default.
- **Delete Input File**:
  The input file is deleted from the input directory or the **Processing Directory**.
- **Move Input File**:
  The input file is moved (archived) to the directory specified in the **To Directory** field. You can also specify an optional **File Prefix** or **File Suffix** for the archived file.

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.

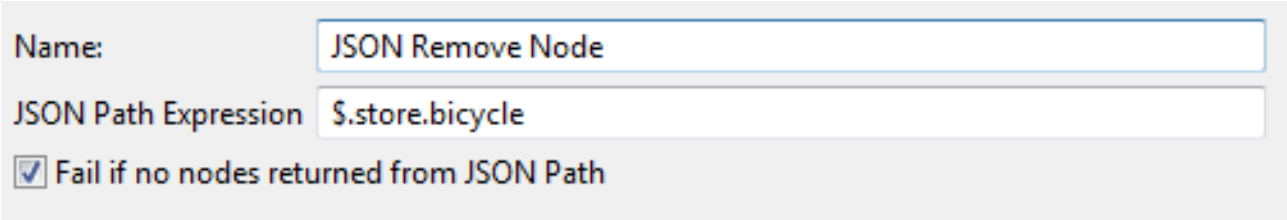
Fail if no nodes returned from JSON Path:
When this option is selected, and the JSON Path expression returns no nodes, the filter returns false. If this option is *not* selected, and the JSON Path returns no nodes, the filter returns true, and no nodes are removed. This option is not selected by default.

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*.

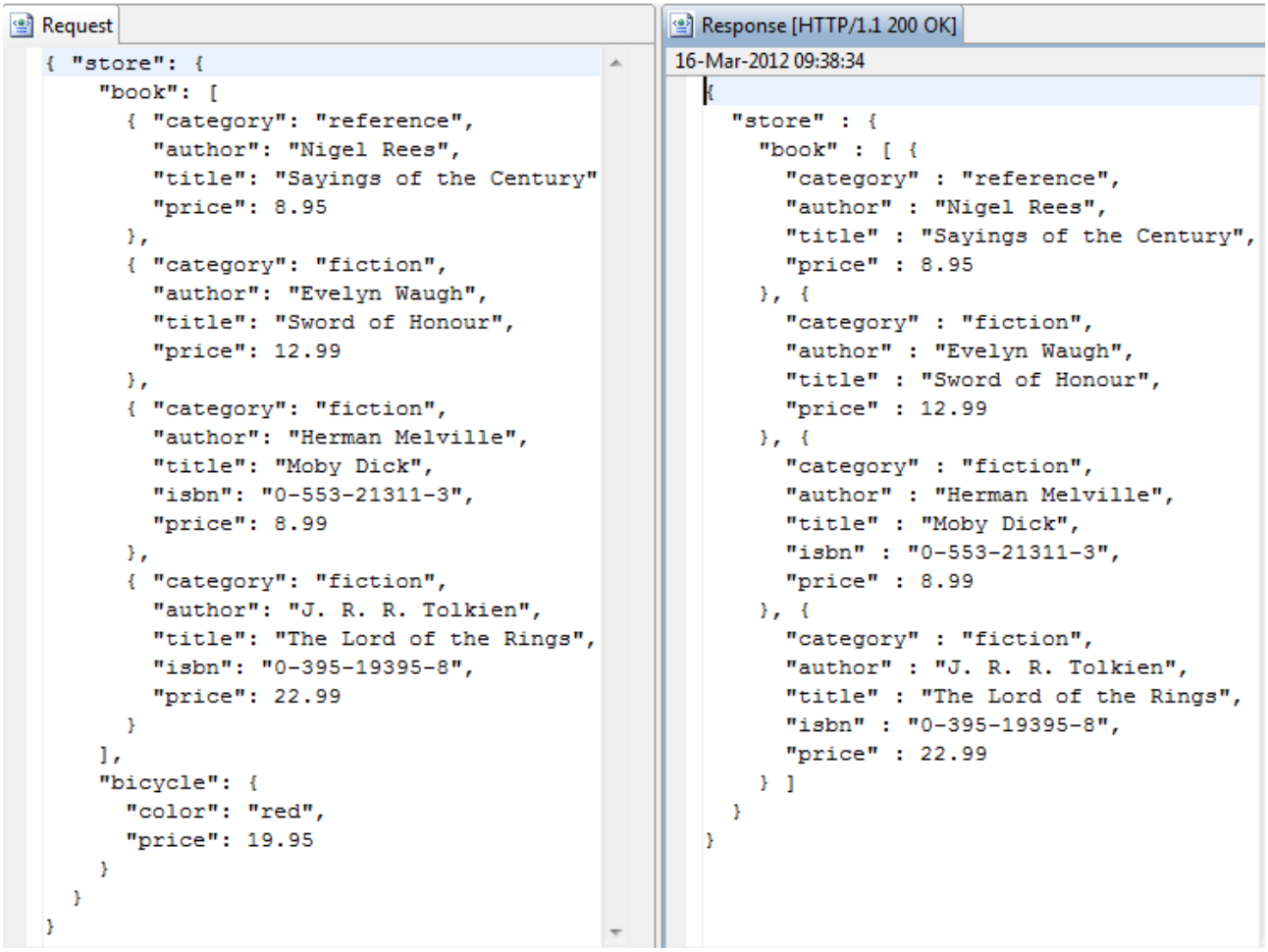
Examples

The following are some examples of using the **JSON Remove Node** filter.

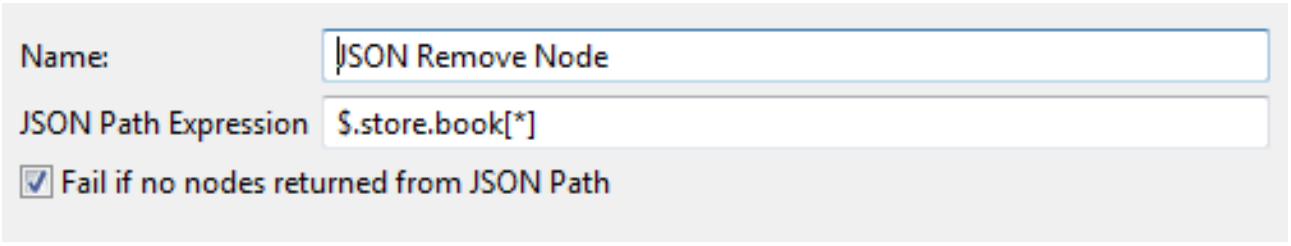
Removing a Node
The following example shows removing a bicycle from the store:



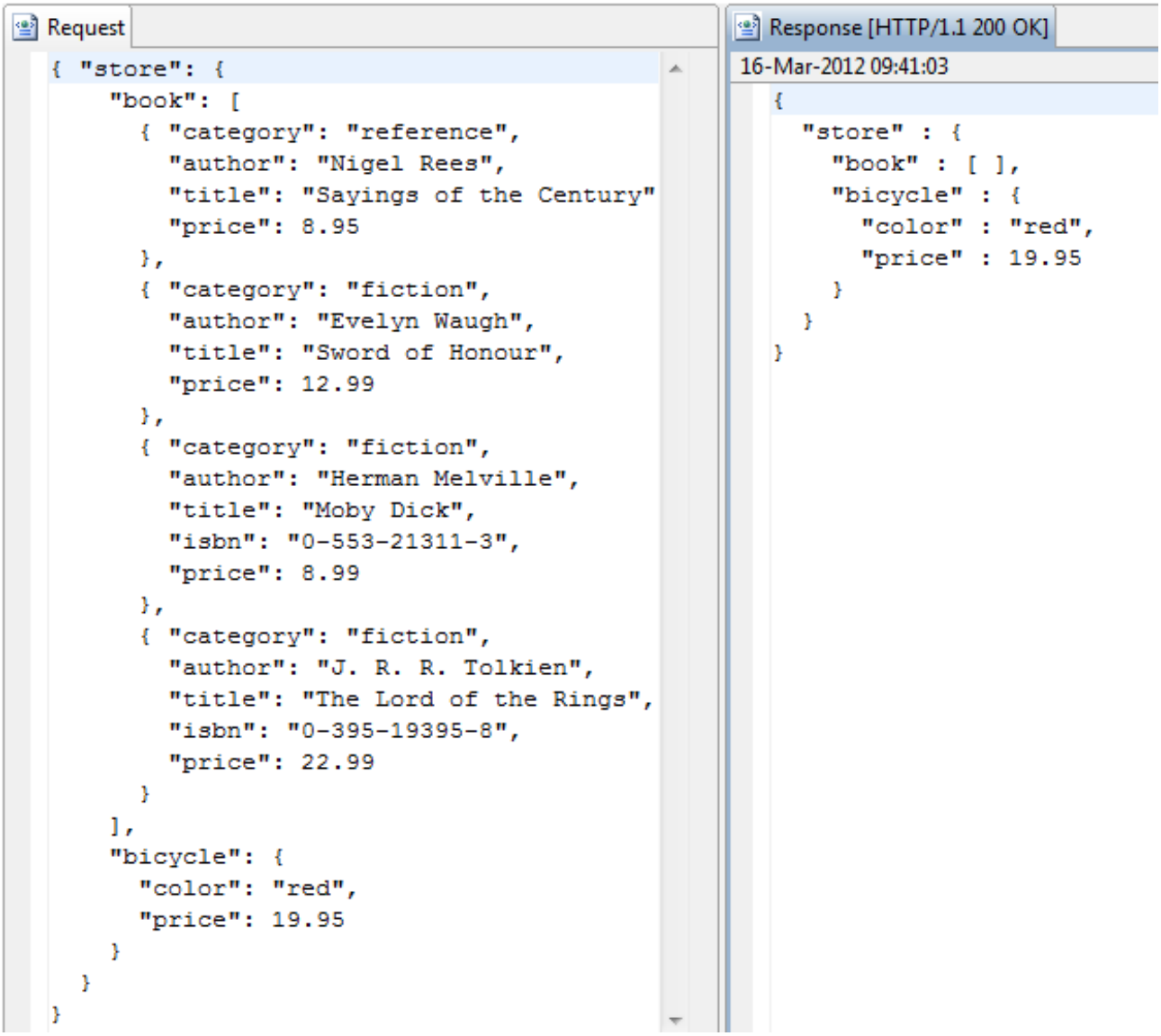
The following example shows the corresponding request and response message in Oracle API Gateway Explorer:



Removing all Items in an Array
The following example shows removing all books in an array:



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 09:41:03
{
"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 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 to ensure that current message attribute values are inserted into the message body at the appropriate places. For more information, see the section called “Example of using selectors in the message body”.

   Alternatively, click **Populate** on the right of the window, and select **From file on disk** to load the message contents from a file, or select **From web service operation** to load the message contents from a web service (WSDL file) that you have already imported into the web service repository.

   You can also insert REST API parameters into the message body. Right-click within the message body at the point where the parameter should be inserted and select **Insert > REST API Parameter**.

Example of using selectors in the message body

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*.

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, go to 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.

Convert number/boolean/null elements to primitives:
Select this option to convert number, boolean, or null elements in the incoming XML document to JSON primitive types. This option is selected by default.

When this option is selected, the filter converts an XML number element to a JSON primitive. Otherwise, an XML number element is converted to a JSON text node. For example:

Incoming XML:

```
<number>123.4</number>
```

JSON output with this option selected:

```
"number" : 12.4
```

JSON output with this option not selected:

```
"number" : "12.4"
```

Similarly, XML boolean or null elements are converted to JSON primitives if this option is selected.

Convert namespace declarations:
Select this option to convert namespace declarations in the incoming XML and add them to the resulting JSON. This option is not selected by default, and any namespace declarations are removed from the resulting JSON.

Use the following XPath to convert:
Select an XPath expression to specify which elements of the incoming XML to convert. The options are:

- **All elements inside SOAP body (SOAP 1.1 or SOAP 1.2)**
- **All elements inside SOAP body (SOAP 1.1)**
- **The entire message**

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 *Security Token Service Client*.

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`. You can also select `Symmetric Key`, which is based on Hash-based Message Authentication Code - Secure Hash Algorithm (HMAC-SHA1).

Key Size:
Enter the key size in bits. Defaults to `2048` bits.

PGP decrypt and verify

Overview

You can use the **PGP Decrypt and Verify** filter to decrypt a message encrypted with Pretty Good Privacy (PGP). 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 API Gateway, and then encrypted again using the **PGP Encrypt and Sign** filter.

An example use case for this filter would be when files are sent to API Gateway over Secure Shell File Transfer Protocol (SFTP) in PGP-encrypted format. API Gateway can use the **PGP Decrypt and Verify** 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 and Sign** filter before being sent over SFTP to their target destination. For more details, see the *PGP encrypt and sign* filter.

You can also use the **PGP Decrypt and Verify** filter to to verify signed messages passing through the API Gateway pipeline. Signed messages received by API Gateway can be verified by validating the signature using the public PGP key of the message signer.

Note

PGP decryption and verification require two different keys: your own private key for decryption, and the sender's public key for verification.

Configuration

Complete the following fields to configure this filter:

Name:
Enter an appropriate name for this filter.

Decrypt:
Select whether to use this filter to PGP decrypt an incoming message with a private key.

PGP Private Key to be retrieved from one of the following locations:
If you selected the **Decrypt** option, select the location of the private key from one of the following options:

- **PGP Key Pair list:**
  Click the browse button on the right, and select a PGP key pair configured in the certificate store. If no PGP key pairs have already been configured, right-click **PGP Key Pairs**, and select **Add PGP Key**. For details on configuring PGP key pairs, see the section called “Configure PGP key pairs”.
- **Alias:**
  Enter the alias name used to look up the PGP key in the certificate store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression with the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **Message attribute:**
  Enter a selector expression with the name of the message attribute that contains the key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.private.key}`).

For more details on selectors, see *Selecting configuration values at runtime*.

Verify:
Select whether to use this filter to verify an incoming signed message with the public key used to sign the message.

Verification Key Location (Public Key):
If you selected the **Verify** option, select the location of the public key from one of the following options:

- **PGP Key Pair list:**
  Click the browse button on the right, and select a PGP key pair configured in the certificate store. If no PGP key pairs have already been configured, right-click **PGP Key Pairs**, and select **Add PGP Key**. For details on configuring PGP key pairs, see the section called "Configure PGP key pairs".
- **Alias:**
  Enter the alias name used to look up the PGP key in the certificate store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression with the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **Message attribute:**
  Enter a selector expression with the name of the message attribute that contains the key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.public.key}`).

For more details on selectors, see *Selecting configuration values at runtime*.

Signing Method:
If you selected to verify but not decrypt the incoming message, select a signing method from one of the following options:

- **Compressed:**
  Verifies a compressed signature. Because the message is contained in the signature, this signature is used in place of the message. This is the default.
- **Clear signed:**
  In a clear signed message, the message is intact with a signature attached beneath 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.

Decrypt and Verify Method:
If you selected to decrypt and verify the incoming message, select the decrypt and verify method from one of the following options:

- **Decrypt and Verify in One Pass:**
  Decrypts and verifies the message in a single pass. This is the default. API Gateway decrypts the message while reading the data packet, and continues on sequentially when it reaches the signature packet.
- **Decrypt and Verify in Two Passes:**
  Decrypts the message in the first pass, and then verifies the signature in the second pass. Use this option when the message has been encrypted and signed in two passes.

Content type:
Enter the `Content-Type` of the unencrypted message data. Defaults to `application/octet-stream`.

PGP encrypt and sign

Overview

You can use the **PGP Encrypt and Sign** 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 Encrypt and Sign** 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 and verify* filter.

You can also use the **PGP Encrypt and Sign** filter to digitally sign messages passing through the API Gateway pipeline. Messages signed by API Gateway can be verified by the recipient by validating the signature using the public PGP key of the signer. Signed messages received by API Gateway can be verified in the same way.

Note

PGP encryption and signing require two different keys: a public key for encryption and your own private key for signing.

General settings

Complete the following field on the **PGP Encrypt and Sign** window:

Name:
Enter an appropriate name for the filter.

Encrypt and sign settings

Complete the following fields on the **Encrypt and Sign** tab:

Encrypt:
Select whether to use this filter to PGP encrypt an outgoing message with a public key.

Encryption Key location (Public Key):
If you selected the **Encrypt** option, select the location of the public key from one of the following options:

- **PGP Key Pair list:**
  Click the browse button on the right, and select a PGP key pair configured in the certificate store. If no PGP key pairs have already been configured, right-click **PGP Key Pairs**, and select **Add PGP Key**. For details on configuring PGP key pairs, see the section called “Configure PGP key pairs”.
- **Alias:**
  Enter the alias name used to look up the PGP key in the certificate store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression with the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **Message attribute:**
  Enter a selector expression with the name of the message attribute that contains the key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.public.key}`).

For more details on selectors, see *Selecting configuration values at runtime*.

Sign:
Select whether to use this filter to sign an outgoing message with a private key.

Signing Key location (Private Key):
If you selected the **Sign** option, select the location of the private key from one of the following options:

- **PGP Key Pair list:**
  Click the browse button on the right, and select a PGP key pair configured in the certificate store. If no PGP key pairs have already been configured, right-click **PGP Key Pairs**, and select **Add PGP Key**. For details on configuring PGP key pairs, see the section called “Configure PGP key pairs”.
- **Alias:**
  Enter the alias name used to look up the PGP key in the certificate store (for example, `My PGP Test Key`). Alternatively, you can enter a selector expression with the name of a message attribute that contains the alias. The value of the selector is expanded at runtime (for example, `${my.pgp.test.key.alias}`).
- **Message attribute:**
  Enter a selector expression with the name of the message attribute that contains the key. The value of the selector is expanded at runtime (for example, `${my.pgp.test.private.key}`).

For more details on selectors, see *Selecting configuration values at runtime*.

Signing Method:
If you selected to sign but not encrypt the outgoing message, select the signing method from one of the following options:

- **Compressed:**
  Compresses the message and creates a hash of the contents before signing. Because the message is contained within the signature this signature can be used in place of the message. The typical use of this method produces a signature in printable ASCII form (ASCII Armor). This option can be turned off to produce a binary signature.
- **Clear signed:**
  Clear signing a message leaves the message intact and adds the signature beneath the clear message text. This provides for optional verification of the message signature and contents. The output has the content type `application/pgp-signature`. 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.

Encrypt and Sign Method:
If you selected to encrypt and sign the outgoing message, select the encrypt and sign method from one of the following options:

- **Encrypt and Sign in One Pass:**
  Encrypts and signs the message in a single pass. This is the default setting. This enables the receiver to decrypt and verify in a single pass.
- **Encrypt and Sign in Two Passes:**
  Signs the message in a first pass, and then encrypts it in a second pass. Because the message is signed before it is encrypted, this means that the message must be decrypted first before the signature can be verified.

Advanced settings

Complete the following fields on the **Advanced** tab:

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 selected by default.

Symmetric Key Algorithm:
Select a symmetric-key algorithm to use to encrypt the data. The default is `CAST5`.

Hash Algorithm:
Select a hash algorithm to use to protect against modification. The default is `SHA1`.

Compression Algorithm:
Select a compression algorithm to use to compress the data. The default is `ZIP`.

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 settings

Complete the following field:

Name:
Enter an appropriate name for the filter.

Recipient settings

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 settings

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.

Nodes to decrypt

An XML message may contain several `EncryptedData` blocks. The **Node(s) to Decrypt** section enables you to specify 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

The **Decryption Key** section enables you to specify 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 the **Options** 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 settings

The settings on the **Encryption Key** 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 settings

The **Key Info** 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.

Recipient settings

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 re-

cipient (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 settings

The **What to Encrypt** 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 settings

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 **Key in Store**, and click **Signing Key**. 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 details, see *Manage certificates and keys*. The *Distinguished Name* of the selected certificate appears 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 **Private Key from Selector Expression**, and enter the selector expression (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.

Using the WS-Trust for SPENGO standard, the Kerberos session keys are not used directly to sign 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 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>11.1.2.3.0</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="11.1.2.3.0">
    <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="11.1.2.3.0">
    <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=11.1.2.3.0>
    <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-fffffffff28f6103-1">
                <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-fffffffff28f6103-2">
                    <dsig:X509Data>
                        <dsig:X509Certificate>........</dsig:X509Certificate>
                    </dsig:X509Data>
                </dsig:KeyInfo>
            </dsig:Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
       <product version=11.1.2.3.0>
```

```
            <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 the message is just plain XML, the signature is inserted as an immediate child of the root element of the XML message. The following example shows a non-SOAP XML message 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 contents of the SOAP message, you can also sign attachments to the SOAP message. To sign all attachments, select **Include Attachments**. 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 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 example 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 if the API Gateway is the initiator as outlined in the scenario above. The API Gateway keeps a record of the signed data and compares 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.

Note

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 list. For details on all the configuration options available in this section, see *Signature Location*.

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, see *What To Sign*.

Note

If all attachments are required to be signed, select **All attachments** 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>`, use the default value entered 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 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.

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

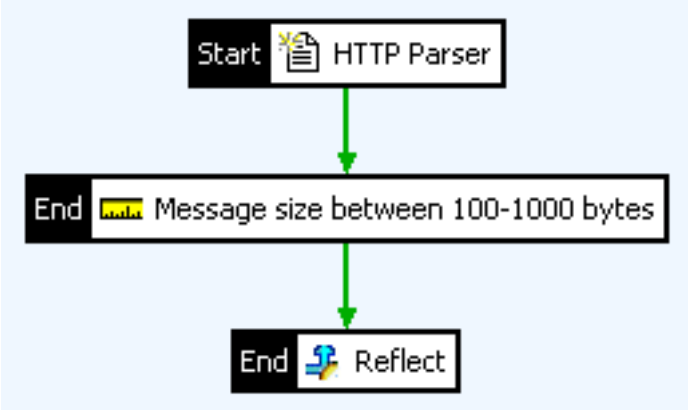
You can use the following approaches to create customized SOAP faults:

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.

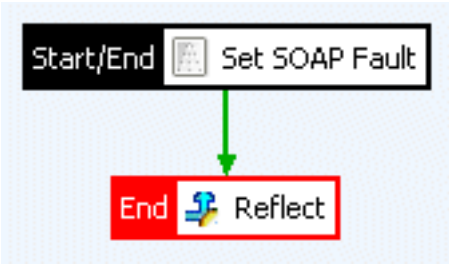
Using the Set Message Filter
You can create customized SOAP Faults using the **Set Message** filter with the **SOAP Fault** filter. The **Set Message** filter can change the contents of the message body to any arbitrary content. When an exception occurs in a policy, you can use this filter to customize the body of the SOAP fault. The following example demonstrates how to generate customized

SOAP faults and return them to the client.

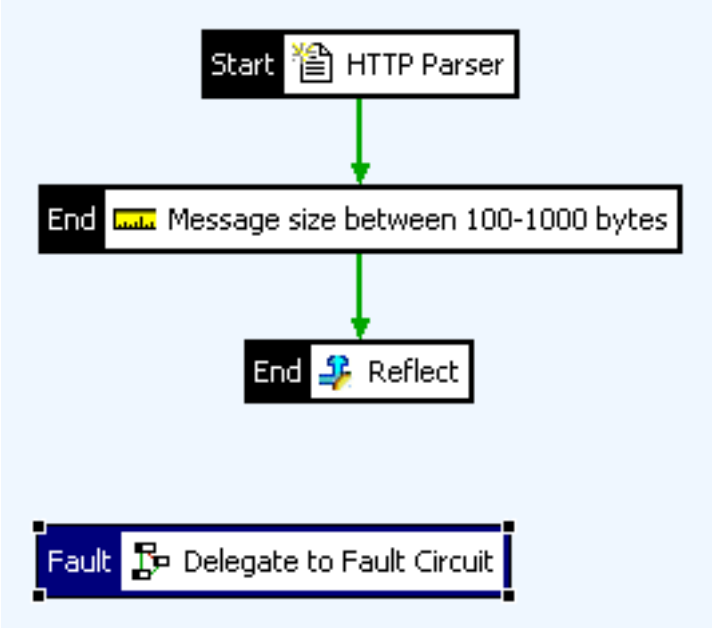
Step 1: Create the Top-level Policy:
This example first creates a very simple policy called **Main Policy**. This policy ensures the size of incoming messages is between 100 and 1000 bytes. Messages in this range are echoed back to the client.



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:



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.

Configure system alerts

Overview

This topic shows the API Gateway system alert capabilities. System alerts and events are usually sent when a filter fails, but they can also be used for notification purposes. API Gateway can send system alerts to several alert destinations, including a Windows Event Log, UNIX/Linux syslog, SNMP Network Management System, Check Point Firewall-1, email recipient, Amazon Simple Notification Service (SNS), or Twitter.

There are two main steps involved in configuring API Gateway to send system alerts:

1. Configure an alert destination
2. Configure an alert filter

Configure an alert destination

The first step in configuring API Gateway to send alerts is to configure an *alert destination*. This section describes the destinations to which the API Gateway can send alerts to. 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, 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. Click the **Libraries** > **Alerts** node, and then click **Add** > **Syslog Remote** at the bottom of the window on the right.
2. The **Syslog Server** dialog enables you to specify details about the machine on which the syslog daemon is running. 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 API Gateway sends alerts to.
3. Click **OK**.

Local syslog (UNIX only)
To configure a local syslog alert destination, perform the following steps:

1. Click the **Libraries** > **Alerts** node, and then click **Add** > **Syslog Local (UNIX only)** at the bottom of the window 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 API Gateway sends alerts to.

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. Click the **Libraries** > **Alerts** node in the Policy Studio tree, and then click **Add** > **Windows Event Log** at the bottom of the window on the right.
2. The **Windows Event Log Alerting** dialog enables you to specify the machine of the event log 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)

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, 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. Click the **Libraries** > **Alerts** node in the Policy Studio tree, and then click **Add** > **OPSEC** at the bottom of the window 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 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 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 window.

For more information on OPSEC settings, see the documentation for your OPSEC application.

SNMP Network Management System

This alert destination enables 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. Click the **Libraries** > **Alerts** node in the Policy Studio tree, and then click **Add** > **SNMP** at the bottom of the window on the right.
2. The **SNMP Alerting** dialog enables you to specify details about the NMS that API Gateway sends alerts to. Complete the following fields:
   - **Host:**
     The host name or IP address of the machine on which the NMS resides.
   - **Port:**
     The port on which the NMS is listening.
   - **Timeout:**
     The timeout in seconds for connections from API Gateway to the NMS.
   - **Retries:**
     The number of retries that should be attempted whenever a connection failure occurs.
   - **SNMP Version:**
     Select the version of SNMP to use for this alert.
3. Click **OK**.

Email recipient

This alert destination enables alert messages to be sent by email. To add an email alert destination, perform the following steps:

1. Click the **Libraries** > **Alerts** node in the Policy Studio tree, and then click **Add** > **Email** at the bottom of the window on the right.
2. The **Email Alerting** dialog enables you to configure how the email alert is 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 specified in this field.

3. In the **SMTP Server Settings**, specify the following fields:
   - **Outgoing Mail Server (SMTP)**:
     Specify the SMTP server that 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 API Gateway when attempting to send email alerts. All trace files are written to the `/trace` directory of your API Gateway installation. This setting is not selected by default.
6. Click **OK**.

Amazon SNS

This alert destination enables alert messages to be sent to the Amazon Simple Notification Service (SNS). Amazon SNS is a managed push messaging service that can be used to push to mobile devices and Internet connected smart devices, as well as to other distributed services. For more information on Amazon SNS, go to http://aws.amazon.com/sns/.

To add an Amazon SNS alert destination, perform the following steps:

1. Click the **Libraries** > **Alerts** node in the Policy Studio tree.
2. Click **Add** > **Amazon SNS** at the bottom of the window on the right.
3. Complete the following fields on the **AWS SNS Alert** dialog:
   - **Name**:
     Enter a name for this alert destination.
   - **AWS Credential**:
     Click the browse button to select your AWS security credentials (API key and secret) to be used by API Gateway when connecting to Amazon SNS.
   - **Region**:
     Select the region appropriate for your deployment. You can choose from the following options:
     - US East (Northern Virginia)
     - US West (Oregon)
     - US West (Northern California)
     - EU (Ireland)
     - Asia Pacific (Singapore)
     - Asia Pacific (Tokyo)
     - Asia Pacific (Sydney)
     - South America (Sao Paulo)
     - US GovCloud

- **Client settings**:
  Click the browse button to select the AWS client configuration to be used by API Gateway when connecting to Amazon SNS. For more information on configuring client settings, see the section called “Configure AWS client settings”.
- **Topic ARN**:
  Enter the topic Amazon Resource Name (ARN) to send alerts to.

  When you create a topic, Amazon SNS assigns a unique ARN to the topic, which includes the service name (for example, SNS), the region, the AWS ID of the user, and the topic name. Whenever a publisher or subscriber needs to perform any action on the topic, they should reference the unique topic ARN. The ARN is returned as part of the API call to create the topic. For example, `arn:aws:sns:us-east-1:1234567890123456:mytopic` is the ARN for a topic named `mytopic` created by a user with the AWS account ID `123456789012` and hosted in the US East region.
- **Subject**:
  Enter the subject of the alert to be sent to Amazon SNS.

4. Click **OK**.

Twitter

This alert destination enables API Gateway to send tweet alerts to Twitter. Twitter uses the OAuth open authentication standard. To enable 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.

Register a client application
To use the Twitter API, you must create a Twitter account, and register a client application for 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 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** might already be registered to another user, so you may need to specify a different unique name.

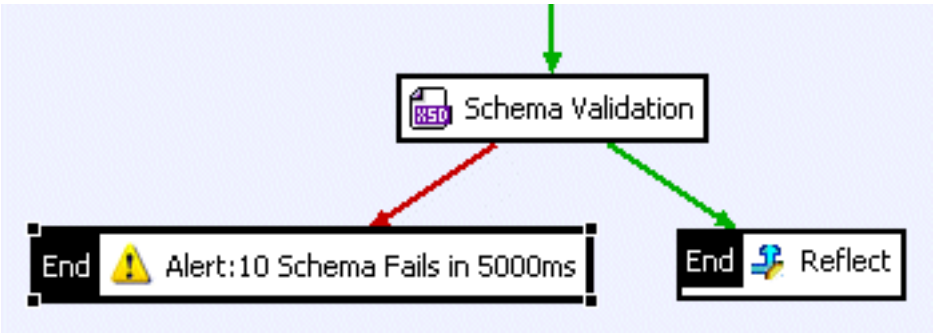
5. Click **Register Application**. Each client application you register is provisioned a consumer key and consumer secret. These are used, in conjunction with the OAuth library, to sign every request you make to the API. Using this signing process, Twitter trusts that the traffic identifying itself as 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.

Configure a Twitter alert destination
To configure a Twitter alert destination, perform the following steps:

1. Click the **Libraries** > **Alerts** node in the Policy Studio tree.
2. Click **Add** > **Twitter** at the bottom of the window on the right.
3. The **Twitter Alerting** dialog enables you to specify credentials for the Twitter user that 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.

Configure an alert filter

Typically, an **Alert** filter is placed on the failure path of another filter in the policy. For example, to 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 example.



When editing policies, you can drag and drop the **Alert** filter from the **Monitoring** filter group.

General settings

Configure the following settings on the **Alert** filter window:

Name:

Enter a descriptive name for the filter.

Alert Type:
Select the severity level of the alert. The options are **Error**, **Warn**, and **Info**. This option is only relevant for alert destinations that support severity levels, such as the Windows Event Log.

Notifications settings

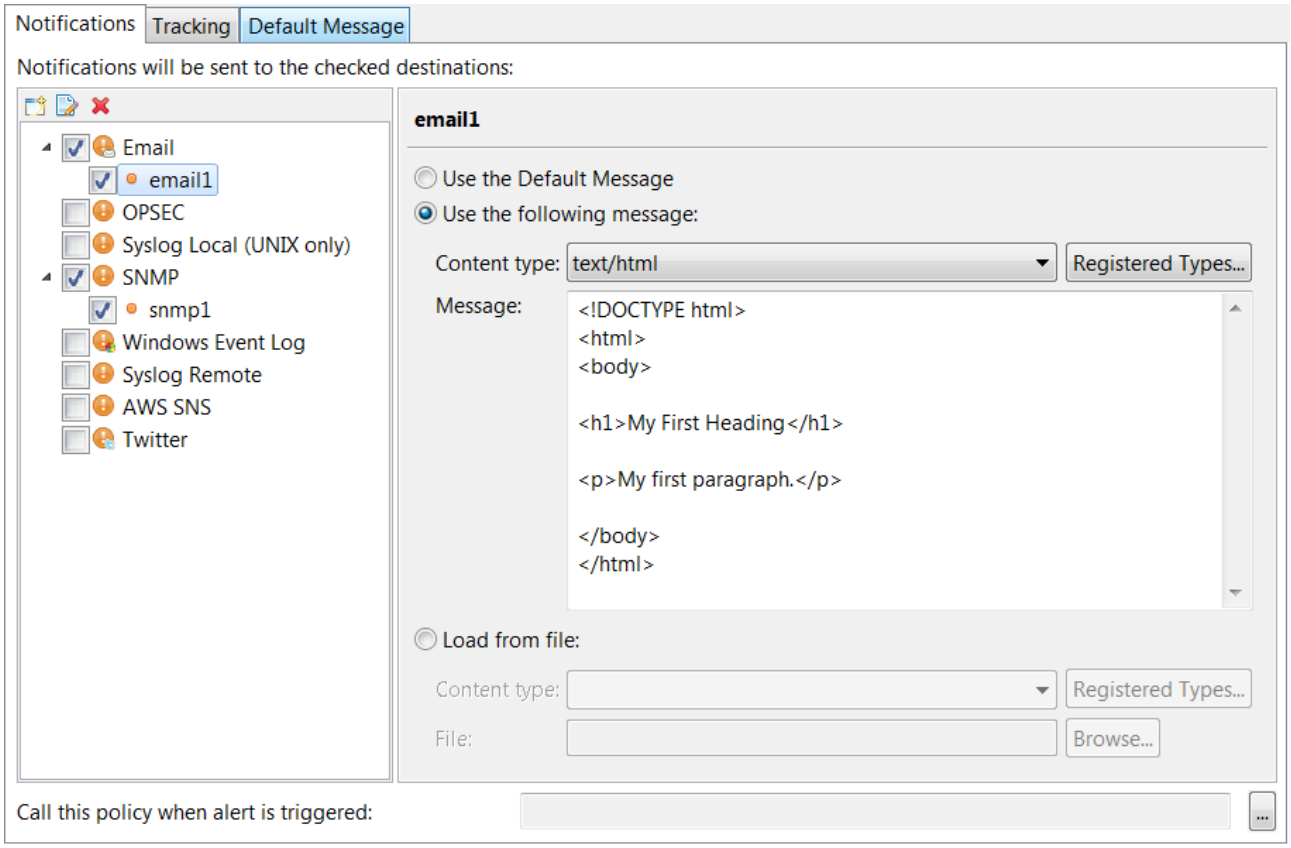
The **Notifications** tab enables you to configure the alert destinations. Any alert destinations that are already configured are shown under the respective alert destination type on the left of the window. You can also create new alert destinations, and edit or delete existing alert destinations directly from this window.

Select an existing alert destination
To select an existing alert destination, follow these steps:

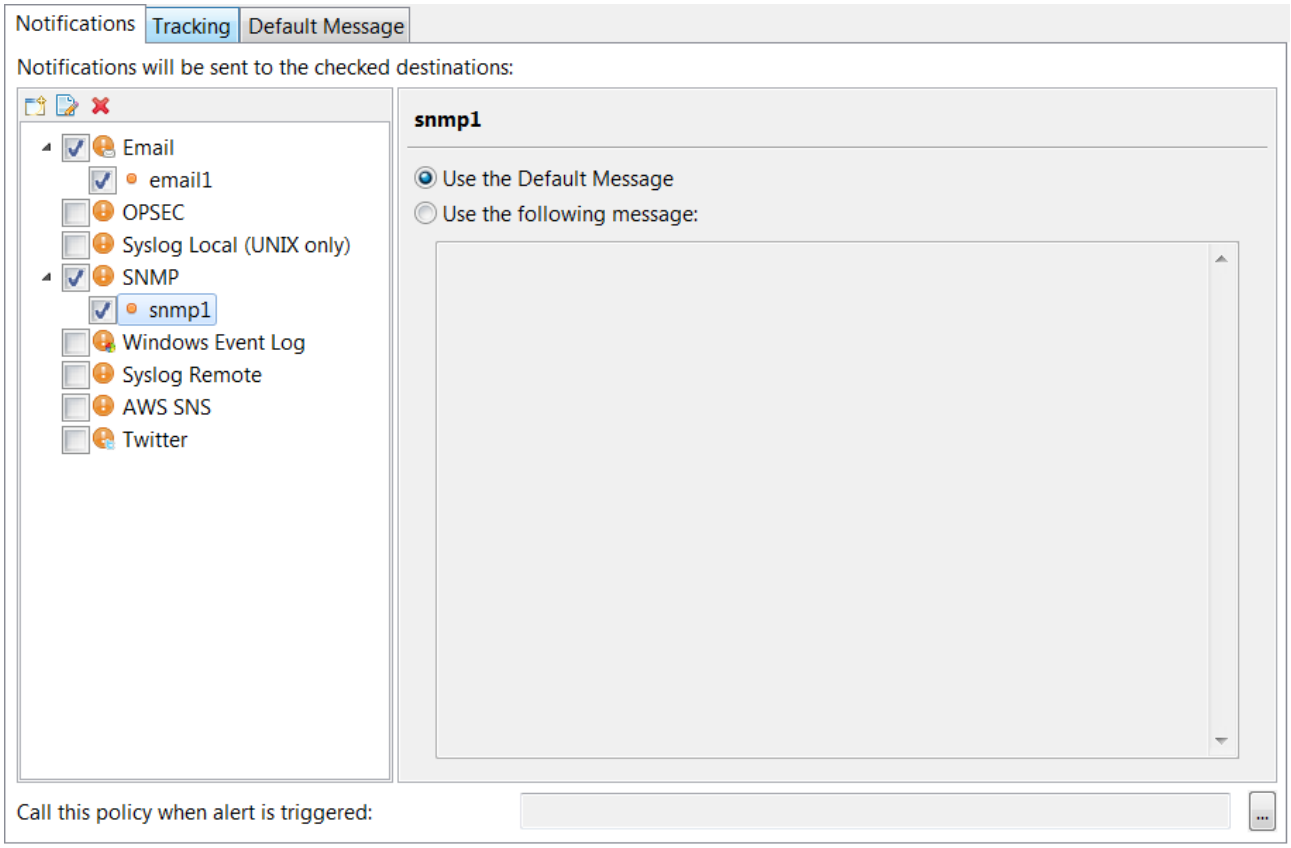
1. Click the check box next to the alert destination on the left of the window. Alternatively, click the check box next to the alert destination type to select all alert destinations of that type (for example, all **Email** destinations).
2. Choose the message to use for the alert destination on the right side of the window.
   - To set the message for an email destination, select **Use the Default Message** to use the message from the **Default Message** tab, select **Use the following message** to enter a custom message, or select **Load from file** to load a message from a file.

     You can select the **Content type** of a custom message or a message loaded from a file (for example, as `text/html`). You can also register new MIME/DIME content types by clicking the **Registered Types** button.
   - To set the message for all other destinations, select **Use the Default Message** to use the message from the **Default Message** tab, or select **Use the following message** to enter a custom message.

The following figure shows an example of an email alert destination with a `text/html` custom message.



The following figure shows an example of an SNMP alert destination that uses the default message.



Create a new alert destination
To create a new alert destination, click the add button on the top left of the window and choose from the following options:

- Email
- OPSEC
- Syslog Local (UNIX only)
- SNMP
- Windows Event Log
- Syslog Remote
- Twitter

For more information on configuring alert destinations see the section called “Configure an alert destination”.

Edit or delete an existing alert destination
To edit or delete an existing alert destination, select the destination on the left of the window and click the edit or delete button at the top left of the window.

Call this policy when alert is triggered:
Click the browse button to select a policy to be used by API Gateway when an alert is triggered.

Tracking settings

The **Tracking** tab enables you to configure how often alerts are sent. 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. The default value is `1`.

In time period (secs):
Enter the time period in which the accumulated number of messages can occur before an alert is triggered. The default is `60` seconds.

Track per client:
Select this option to record the accumulated number of messages in the specified time period for each client. This option is selected by default.

Default message settings

The **Default Message** tab enables you to configure a default message for alerts. Enter the message text to appear in the alert in the **Message to send** field.

You can enter message attributes using selectors, which API Gateway looks up and expands at runtime. For example, instead of sending a generic alert stating `Authentication Failed`, you can use a message attribute to include the ID of the user whose authentication failed.

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}.
```

For more information on selectors, see *Selecting configuration values at runtime*.

Note

An alert message is not required for alerts sent to an OPSEC firewall.

Set transaction log level and log 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 *API Gateway Administrator Guide*.

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 **Transaction Log Level and Message** window by clicking the **Next** button on the main window of all filters. This window 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*.

Transaction 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 *API Gateway Administrator Guide*.

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 *API Gateway Administrator Guide*.

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.

Service level agreement

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.

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 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 (3000ms), the *data age* is set to 6 seconds (6000ms), and you have a web service with an average processing time of 100ms. A single client sending a stream of requests through API Gateway will be able to generate about 10 requests per second, given the web service'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.

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 might 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 request to the web

Response Time Measurement	Description
	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.

API Gateway will measure each of the 8 time values. They will be available for processing after the policy has completed for a single request. These 8 options are available for the following reasons:

- API Gateway might start to send the first byte to the web service before the last byte is received from the client (send-request-start <receive-request-end). This will occur if the invoked policy does not require the full message to be read into memory.
- API Gateway might start to send the response to the client before the complete response has been received from the web service, (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 might start to send the response before it has received the complete request. However, 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 might cause an SLA to trigger unnecessarily. By using percentages, such requests will not distort the statistics collected by 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, that is, 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. 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. API Gateway will *stop* measuring the response time from this time.

HTTP status requirements

HTTP status codes might be received from a web service. 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 can 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,

HTTP Status Code Class	Description
	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.

API Gateway might monitor a class (that is, range) of status codes, or it might 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 can yield HTTP status codes between 400 and 499
- At most 0% of requests can 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** field. 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.

To monitor just 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 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

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 can be expressed as follows:

- No more than 4% of requests can 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 (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. 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 window 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 *Configure system alerts*.

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 window to configure whether 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 API Gateway Analytics reporting tools.

General settings

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 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 API Gateway Analytics to produce reports showing how and who is calling this service. For details on the fields, see the section called “Monitoring settings” in the *Web service filter* topic.

Send event to Sentinel

Sentinel event filter overview

You can use the **Sentinel Event** filter to send tracked events to Axway Sentinel. Sentinel uses *tracked objects* to identify events. Every tracked object contains a unique name, version number, and a list of attributes.

Every tracked event must specify a tracked object, and this tracked object must already be defined in Sentinel. A tracked event can also contain attributes, and the attributes must already be defined as tracked object attributes in Sentinel.

General settings

Configure the following settings on the **Sentinel Event** window:

Name:
Enter a suitable name for the filter.

Settings tab

Sentinel Server:
Click the browse button to select a Sentinel server connection.

The **Tracked object** section enables you to specify the tracked object to use in the Sentinel event.

Note

Tracked objects must exist in your Sentinel database before you can start using Sentinel to monitor your applications and track their activities. For more information on defining tracked objects in Sentinel, see the *Sentinel Configuration Guide* available on the Axway Support website: https://support.axway.com.

Use the following tracked object:
Select this option and click the browse button to select a Sentinel tracked object to use. If no tracked objects are already defined, right-click **Sentinel Tracked Objects** in the dialog and select **Add a tracked object**. Enter a **Name** and a **Version** for the tracked object. The values entered must correspond to the **Public name** and **Version** of the tracked object in Sentinel.

Or use the following from the message:
Select this option to retrieve the tracked object name and tracked object version from a message received from an upstream product (for example, B2Bi). If the upstream product has inserted tracking information inside some HTTP headers, you can use selectors to retrieve these from the message. For example, `${http.headers["X-TRACKEDOBJECT-NAME"]}` retrieves the tracked object name from the HTTP headers, and `${http.headers["X-TRACKEDOBJECT-IDENTITY"]}` retrieves the tracked object version from the HTTP headers.

The **Event will contain the following** section enables you to specify the tracked object attributes to use in the Sentinel event.

Note

The named event attributes specified in this section must be contained within the tracked object definition in Sentinel.

Include Cycle ID:
Select this option to include the cycle ID in the event, and enter a value. For example, enter `${id}` to use the API Gateway transaction ID as the cycle ID. This value is used to populate the `CycleId` attribute of the tracked object in Sentinel.

Include policy name in event named:
Select this option to include the name of the policy in the event, in an attribute with the specified name. You can use any name for the attribute, as long as the attribute name exists in the tracked object definition in Sentinel.

Include filter name in event named:
Select this option to include the name of the filter in the event, in an attribute with the specified name. You can use any name for the attribute, as long as the attribute name exists in the tracked object definition in Sentinel.

You can also add other attributes to be included in the event by populating entries in the table. Click the **Add** button to add an attribute. Enter a **Name** and a **Value** for the attribute. For example, to populate an attribute named `UserName` with the authenticated user, you would enter `UserName` for the **Name** and `${authentication.subject.id}` for the **Value**.

Send as update:
Select this option to send the event as an update.

Tracking tab

Add Cycle ID in header named:
Select this option and enter a value, to include the cycle ID in the HTTP header.

Add Tracked Object name in header named:
Select this option and enter a value, to include the tracked object name in the HTTP header.

Add Tracked Object version in header named:
Select this option and enter a value, to include the tracked object version in the HTTP header.

Further information

For more detailed information on Sentinel integration, see the *Sentinel Integration Guide* available from Axway Support.

Send cycle link event to Sentinel

Sentinel cycle link filter overview

You can use the **Sentinel Cycle Link** filter to send cycle link events to Axway Sentinel. Sentinel uses *cycle links* to link processing cycles sequentially. A *processing cycle* is a group of related tracked events (identified by the same cycle ID).

Every cycle link must specify a parent processing cycle and a child processing cycle. A processing cycle is identified by a cycle ID, and the tracked object name and version are used to identify the tracked events within the processing cycle.

This filter can be used to link related events from different products. For example, if B2Bi and API Gateway are both sending events to the same Sentinel server, each product sends the events with different cycle IDs. You can link the events from B2Bi with the events from API Gateway by sending a cycle link event to the Sentinel server. This links the two cycle IDs in Sentinel.

General settings

Configure the following settings on the **Sentinel Cycle Link** window:

Name:
Enter a suitable name for the filter.

Sentinel Server:
Click the browse button to select a Sentinel server connection.

The **Parent Settings** section enables you to specify the cycle ID for the parent processing cycle. You also need to specify the tracked object name and version to identify the relevant tracked events.

Parent Cycle ID:
Enter the cycle ID of the parent processing cycle. This should be the cycle ID of the upstream product (for example, B2Bi). For example, `${http.headers["X-TRACKING-CYCLEID"]}` retrieves the parent cycle ID from the HTTP request headers.

Use the following tracked object:
Select this option and click the browse button to select a Sentinel tracked object to use. If no tracked objects are already defined, right-click **Sentinel Tracked Objects** in the dialog and select **Add a tracked object**. Enter a **Name** and a **Version** for the tracked object. The values entered must correspond to the **Public name** and **Version** of the tracked object in Sentinel.

Or use the following from the message:
Select this option to retrieve the tracked object name and tracked object version from a message received from an upstream product (for example, B2Bi). If the upstream product has inserted tracking information inside some HTTP headers, you can use selectors to retrieve these from the message. For example, `${http.headers["X-TRACKEDOBJECT-NAME"]}` retrieves the tracked object name from the HTTP headers, and `${http.headers["X-TRACKEDOBJECT-IDENTITY"]}` retrieves the tracked object version from the HTTP headers.

The **Child Settings** section enables you to specify the cycle ID for the child processing cycle. You also need to specify the tracked object name and version to identify the relevant tracked events.

Child Cycle ID:
Enter the cycle ID of the child processing cycle. This should be the cycle ID of API Gateway. For example, enter `${id}` to specify the API Gateway transaction ID.

Use the following tracked object:
Select this option and click the browse button to select a Sentinel tracked object to use. If no tracked objects are already defined, right-click **Sentinel Tracked Objects** in the dialog and select **Add a tracked object**. Enter a **Name** and a **Version** for the tracked object. The values entered must correspond to the **Public name** and **Version** of the tracked object

in Sentinel.

Or use the following from the message:
Select this option to retrieve the tracked object name and tracked object version from a message received from an upstream product (for example, B2Bi). If the upstream product has inserted tracking information inside some HTTP headers, you can use selectors to retrieve these from the message. For example, `${http.headers["X-TRACKEDOBJECT-NAME"]}` retrieves the tracked object name from the HTTP headers, and `${http.headers["X-TRACKEDOBJECT-IDENTITY"]}` retrieves the tracked object version from the HTTP headers.

Further information

For more detailed information on Sentinel integration, see the *Sentinel Integration Guide* available from Axway Support.

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.

General Configuration

Configure the following general fields:

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.

Request Configuration

Configure the following fields to authorize a user for a particular resource against Oracle Access Manager:

Resource Type:
Enter the resource type for which you are requesting access (for example, `http` for access to a Web-based URL).

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 send the message to the API Gateway (for example, POST).

Include Query String
Select whether the OAM server uses the HTTP query string parameters to determine the policy that protects this resource. This setting is optional if the configured policies do not rely on the query string parameters. This setting is not configured by default.

OAM Access SDK Configuration

Configure the following fields for the OAM Access SDK:

OAM ASDK Directory:
Enter the path to your OAM Access SDK directory. For more details on the OAM Access SDK, see your Oracle Access Manager documentation.

OAM ASDK Compatibility Mode:
Select the Oracle Access Manager server version to which this filter connects (`10g` or `11g`). Defaults to `11g`.

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 send 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 SDK Configuration

Configure the following fields for the OAM Access SDK:

OAM ASDK Directory:
Enter the path to your OAM Access SDK directory. For more details on the OAM Access SDK, see your Oracle Access Manager documentation.

OAM ASDK Compatibility Mode:
Select the Oracle Access Manager server version to which this filter connects (`10g` or `11g`). Defaults to `11g`.

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 ASDK Directory:
Enter the path to your OAM Access SDK directory. For more details on the OAM Access SDK, see your Oracle Access Manager documentation.

OAM ASDK Compatibility Mode:
Select the Oracle Access Manager server version to which this filter connects (`10g` or `11g`). Defaults to `11g`.

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 ASDK Directory:
Enter the path to your OAM Access SDK directory. For more details on the OAM Access SDK, see your Oracle Access Manager documentation.

OAM ASDK Compatibility Mode:
Select the Oracle Access Manager server version to which this filter connects (`10g` or `11g`). Defaults to `11g`.

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 example shows an HTTP 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

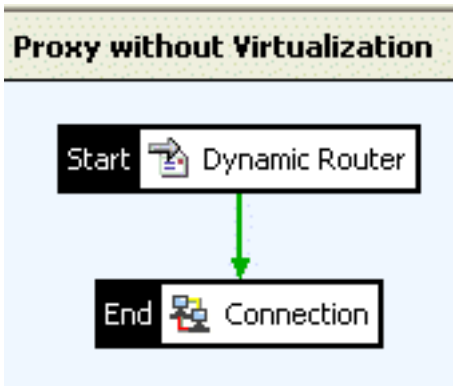
In this case, the API Gateway is configured as an HTTP proxy for the client, and maintains the original path used by the client in the HTTP request. For example, if the API Gateway is listening at `http://localhost:8080/`, and the Web service is running at `http://localhost:5050/services/getQuote`, the 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*

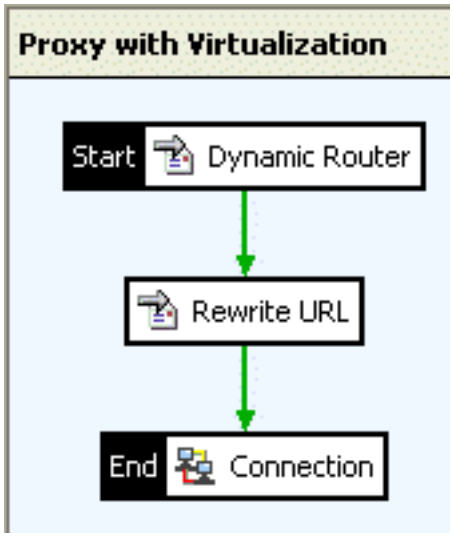
Case 2: Proxy with Service Virtualization

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*

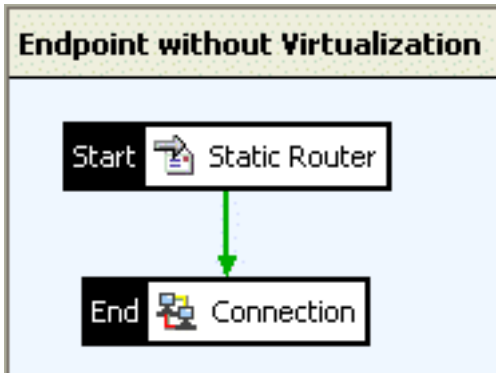
Case 3: Endpoint without Service Virtualization

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*

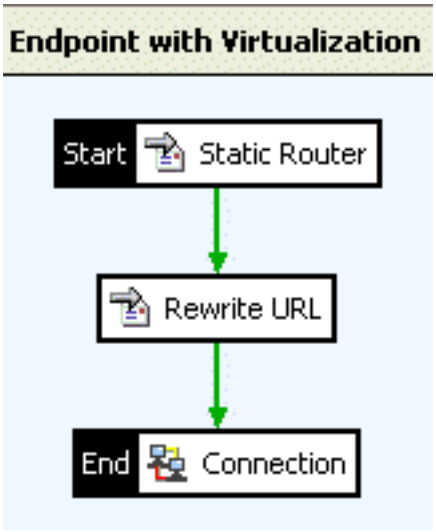
Case 4: Endpoint with Service Virtualization

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 (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 end-point 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 de-ployed.	*Rewrite URL*
Connection	Establishes the connection to the destination Web service, and sends the message over this connection. This connec-tion 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.

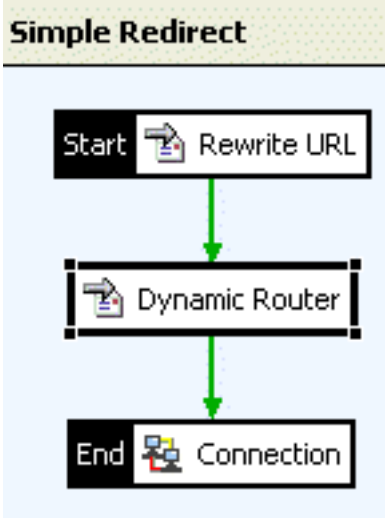
Case 5: Simple Redirect

In some cases, the API Gateway must route the incoming message to an entirely different URL. You can use the **Re-write 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 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 sitting 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 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, `ht-`	*Rewrite URL*

Filter	Role in Policy	Details
	`tp://HOST:8080/myServices`). Because you are routing through a proxy, the full URL is sent in the request line of the HTTP request.	
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

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.

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 *Configure 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 settings

Enter an appropriate name for the filter in the **Name** field. Click the tabs to configure the various aspects of the **Connection** filter.

SSL settings

You can configure SSL settings, such as trusted certificates, client certificates, and ciphers on the **SSL** tab. For details on the fields on this tab, see the section called “SSL settings” in the *Connect to URL* topic.

Authentication settings

You can select credential profiles to use for authentication on the **Authentication** tab. For details on the fields on this tab, see the section called “Authentication settings” in the *Connect to URL* topic.

Additional settings

The **Settings** tab allows you to configure the following additional settings:

- **Retry**
- **Failure**
- **Proxy**
- **Redirect**
- **Headers**

By default, these sections are collapsed. Click a section to expand it.

For details on the fields on this tab, see the section called “Additional settings” in the *Connect to URL* topic.

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 settings

Configure the following general settings:

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}`.

Tip

You can also enter any query string parameters associated with the incoming request message as a selector, for example, `${http.request.uri}?${http.raw.querystring}`.

Request settings

On the **Request** 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*. The values specified on this tab are used in the outbound request to the URL.

Method:
Enter the HTTP verb used in the incoming request (for example, `GET`). Defaults to `${http.request.verb}`.

Request 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 **Request 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>
```

Request Protocol Headers:
Enter the HTTP headers associated with the incoming request message. Defaults to `${http.headers}`.

SSL settings

Configure the SSL settings on the **SSL** tab. You can select the server certificates to trust on the **Trusted Certificates** tab, and the client certificates on the **Client Certificates** tab.

You can also specify the ciphers that API Gateway supports in the **Ciphers** field. The API Gateway sends this list of supported ciphers to the destination server, which 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.

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 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.

To select all certificates for a particular CA, select the box next to the CA parent node in the table.

Alternatively, you can select the **Trust all certificates in the Certificate Store** option to trust all certificates in the store. This is selected by default.

Client certificates

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 Certificates** tab.

To select a client certificate click the **Signing Key** button, and complete the following fields on the **Select Certificate** dialog:

Choose a specific certificate:
Select this option to choose a certificate from the Certificate Store. Select the check box next to the client certificate to use it to authenticate to the server specified in the **URL** field.

Bind the certificate at runtime:
Select this option to bind the certificate at runtime. You can specify this setting as an environment variable selector, which enables values to be expanded at runtime. For more details, see *Selecting configuration values at runtime*.

Authentication settings

The **Authentication** tab enables you to select a client credential profile for authentication. You can use client credential profiles to configure client credentials and provider settings for authentication using API keys, HTTP basic or digest authentication, Kerberos, or OAuth.

Click the browse button next to the **Choose a Credential Profile** field to select a credential profile. You can configure client credential profiles globally under the **External Connections** node in the Policy Studio tree. For more details on configuring client credentials, see the *Configuring client credentials* topic.

Additional settings

The **Settings** tab enables you to configure the following additional settings:

- **Retry**
- **Failure**
- **Proxy**
- **Redirect**
- **Headers**

By default, these sections are collapsed. Click a section to expand it.

Retry settings

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 **Retry** 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. Defaults to `5`.

Retry Interval (ms):
Enter the time to delay between retries in milliseconds. Defaults to `500` ms.

Add an HTTP status range
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.
End status	Enter the last HTTP status code in the range.

To add 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 **Edit** and **Delete**.

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 location of a running API Gateway instance.

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.

Proxy settings

To specify the proxy settings for this filter, complete the following fields:

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 browse button next to this field, and select an existing proxy server. 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 **External Connections** in the Policy Studio tree. For more details, see the *Proxy Servers* topic.

Transparent Proxy (present client's IP address to server):
Enables 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 the *Configuring a Transparent Proxy* topic.

Redirect settings

To specify the redirect settings for this filter, complete the following fields:

Follow Redirects:
Specifies whether the API Gateway follows HTTP redirects, and connects to the redirect URL specified in the HTTP response. This setting is enabled by default.

Header settings

To specify the header settings for this filter, complete the following fields:

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, 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, 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 field is not selected by default.

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.

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.

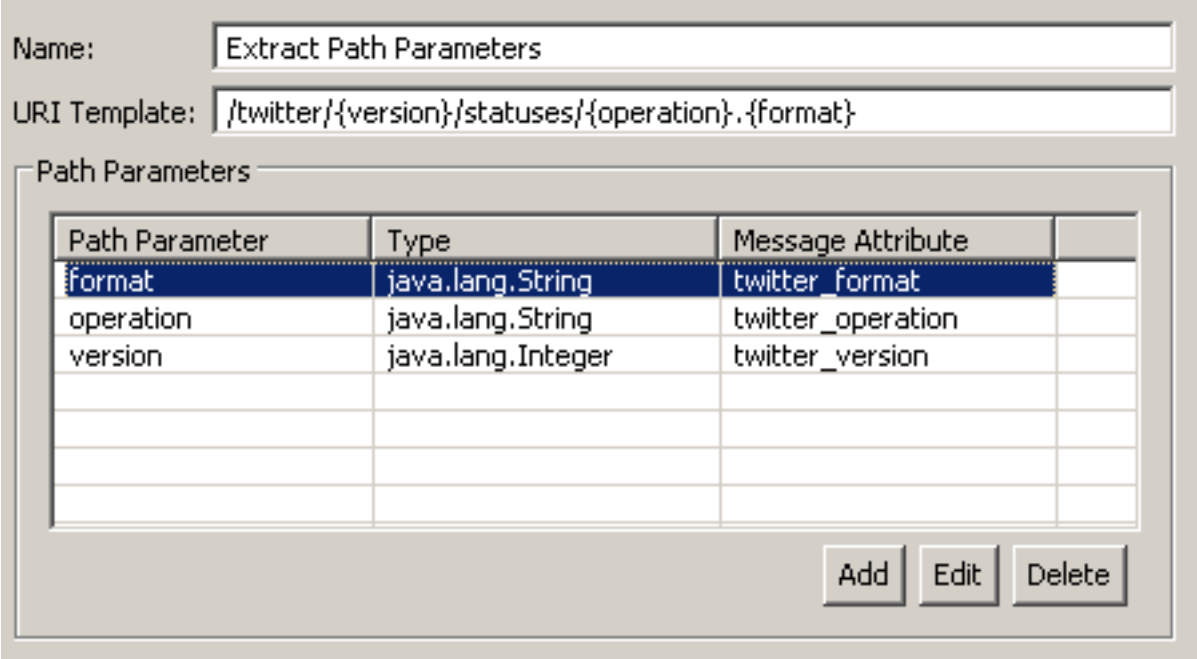
URI Template:
Enter the URI template for the path to be parameterized. This is a formatted Jersey `@Path` annotation string, which enables you to parameterize the path specified in the incoming `http.request.path` message attribute. The following is an example URI template entry:

```
/twitter/{version}/statuses/{operation}.{format}
```

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:



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 con-

nection 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 *Manage 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 *Manage 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.

Java Message Service (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, 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 iinstalled (`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 *JMS Services* topic.

Destination:
Enter the name of the JMS queue or topic that you want to drop messages on to.

Note

For Apache ActiveMQ, this is the JNDI name, and not the JMS queue or topic name.

Delivery Mode:
The API Gateway supports the following 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 priority 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 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 Message 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 used to read from the queue. For example, 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`). Then 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 location of a running API Gateway instance.
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.

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.

Filter Expression	Description
`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].

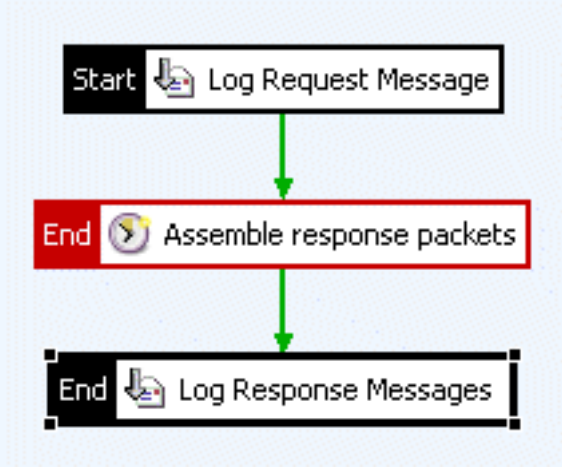
Promiscuous Mode:
When listening in promiscuous mode, the Packet Sniffer captures all packets on the same Ethernet network, regardless of whether the packets are addressed to the network interface that the Sniffer is monitoring.

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.

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`.

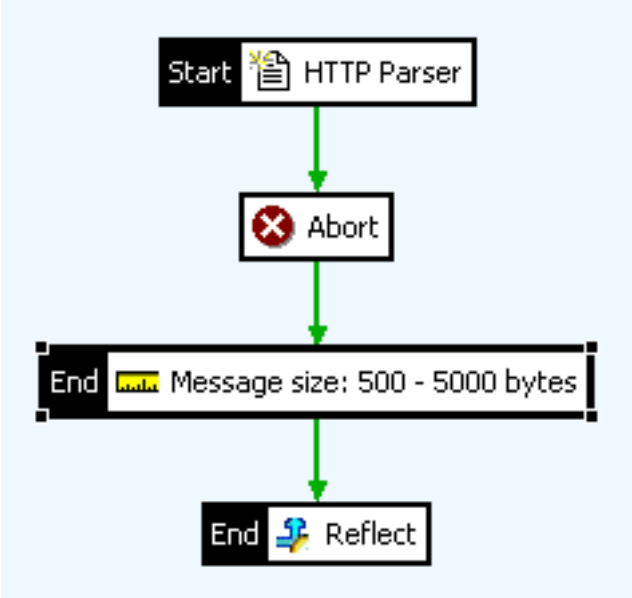
Generate Authenticator Using:
You can verify the authenticator using the **Generated** or **Consumed** message. In either case, you should select the appropriate type of WS-Trust message from the available options.

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 user group. The user and the group are both stored in the API Gateway user store. For more details, see *Manage API Gateway users*.

Configuration

Configure the following required fields:

Name:
Enter an appropriate name for this filter.

User:
Enter the user name configured in the API Gateway user store. You can specify this value as a string or as a selector that expands to the value of the specified message attribute at runtime. Defaults to `${authentication.subject.id}`.

Group:
Enter the group name configured in the API Gateway user store (for example, `engineering` or `sales`). You can specify this value as a string or as selector that expands to the value of the specified message attribute at runtime (for example, `${groupName}`).

Note

The message attribute specified in the selector must exist on the message white board prior to calling the filter. For more details on selectors, see *Selecting configuration values at runtime*.

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 Selector

Overview

The **Evaluate Selector** filter enables you to evaluate the contents of a specified selector expression, and return a boolean result. A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime.

This filter enables you to evaluate a specified selector expression and make a decision in a policy based on whether the expression value fails or passes. For example, you could use the following expression to check if the user belongs to a particular group that allows the user to access a particular resource:

```
${user[0].memberOf.contains("CN=Group Policy Creator Owners,CN=Users,DC=acmeqa,DC=com")}
```

This expression checks if the `memberOf` attribute retrieved for the first `user` contains the specified value (in this case, membership of a particular group). If the expression matches, the filter will pass.

Alternatively, you could use the following selector expression to check if the user email address is valid:

```
${user[0].mail.contains("admin@qa.acme.com")}
```

This expression checks if the `mail` attribute retrieved for the first `user` contains the specified value (in this case, a particular email address). If the expression matches, the filter will pass.

For more details on selectors, see *Selecting configuration values at runtime*.

Configuration

Name:
Enter a descriptive name for this filter.

Expression:
Enter the selector expression to be evaluated. Defaults to the following selector expression:

```
${1 + 1 == 2}
```

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 location of a running API Gateway instance. 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:

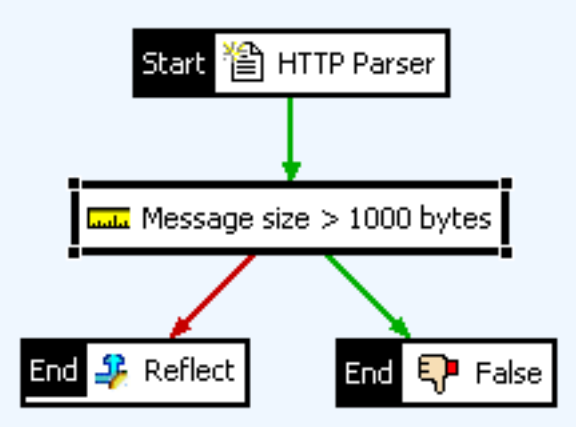
Environment variables to set	Click **Add** to add environment variables. In the dialog, specify an **Environment variable name** (for example, `JAVA_HOME`) and a **Value** (for example, `c:\jdk1.6.0_18`), and click **OK**. Repeat to add multiple variables.
Block till process finished	Select whether to 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 expres-

sion. 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 *Configure 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`

- `Equals`
- `Does not Equal`
- `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.

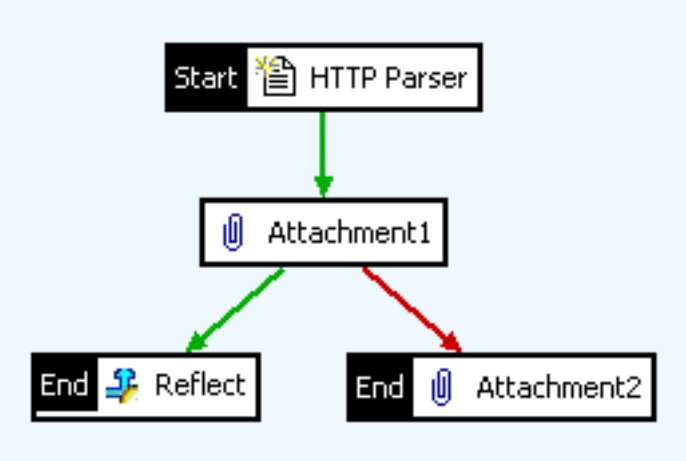
Include Body:
Select this option if you wish to trace the entire message body.

Indent XML:
If this option is selected, the XML message is pretty-printed (indented) before it is output to the trace destination.

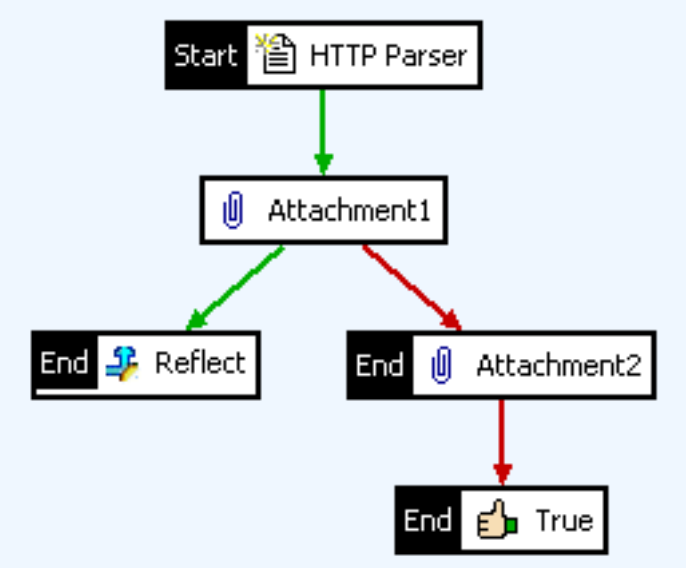
True Filter

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 service 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 where the imported WSDL file contains WS-Policy assertions, a number of policies are automatically created containing the filters required to generate and validate 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 service repository, see the *Manage the web service repository* topic.

When you select a web service from the **Web Service Context** field, the **Message Interception Points** tab is automatically populated with resolvers for the operations exposed by that web service. Similarly, the **Routing** tab is automatically populated with the routing information for the selected web service.

Routing settings

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 SSL settings, credential profiles for authentication, and other settings for the direct connection using the tabs in the **Connection Details** group. For more details, see the *Connect to URL* topic.

Delegate to Routing Policy:
To use a dedicated routing policy to send messages on to the web service, you can select this radio button. For example, you might 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 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 settings

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. 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, 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. 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.
- **SOAP Version**:
  API Gateway also validates requests by matching the SOAP protocol version in the message against the SOAP binding version (1.1 or 1.2) of the corresponding operation definition in 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 service repository, Policy Studio can extract the XML schema from the WSDL and configure API Gateway to validate incoming requests against it. Select **Use WSDL Schema** 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 to use to validate requests, and click the **OK** button.

Configuring 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 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 window are automatically configured when importing a service into the web service 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 `UsernameToken` elements, 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 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 check box next to a previously configured policy.

1B)Operation-specific Policy Shortcuts:
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 can 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 check box next to the policy 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:
You 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.

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 you to 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.

6. Response to Client:
This enables you to process the response before it is returned to the client.

6A) Before Operation-specific Policy:
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 settings

You can expose an imported WSDL file to clients of API Gateway. A client can retrieve a WSDL for a service by appending `WSDL` 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 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 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 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 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 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 API Gateway, generate SOAP requests, and send these requests to API Gateway. API Gateway then routes the requests on to the web service.

WSDL Access Policy:
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 to run on requests to retrieve the WSDL.

Monitoring settings

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 to store message metrics for this web service.
- **Monitor service usage per client**:
  Select this option to generate reports that monitor which authenticated clients are calling which web services.
- **Monitor client usage**:
  To generate reports on authenticated clients, but not 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 user name 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, 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 service 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 *Manage 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 preconfigured `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 *Manage the web service repository* topic.

General settings

Name:
Enter a name for the filter.

Service WSDL settings

The **Service WSDL** tab enables you to select the web service to obtain resources from in the web service repository.

Click the browse button to 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 **Business Services > Web Services** node, and select **Register Web Service**. For more details on adding services to the web service repository, see the *Manage the web service repository* topic.

Monitoring settings

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. For details on the fields on this tab, see the section called “Monitoring settings” in the *Web service filter* topic.

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 *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}
  ```

Important

Do not use hyphens (`-`) in selector expressions. Hyphens are not supported by the Java-based selector syntax. You can use underscores (_) instead.

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}
```

Tip

You can use the **Trace** filter to determine the appropriate selector expressions to use for specific message attributes. When configured after another filter, the **Trace** filter outputs the available message attributes and their Java type (for example, `Map` or `List`). For details on `com.vordel` classes, see:

```
<install-dir>/apigateway/docs/javadoc/index.html
```

For example, for the `OAuth2AccessToken` class, you can use selector expressions such as `${accesstoken.getAdditionalInformation()}`.

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.

Key Property Stores

Overview

A *Key Property Store* (KPS) is an external data store of API Gateway policy properties, which is typically read frequently, and seldom written to. Using a KPS enables metadata-driven policies, whereby policy configuration is stored in an external data store, and looked up dynamically when policies are executed. You can specify policy configuration settings in Policy Studio using selectors, which are evaluated and expanded dynamically at runtime.

Every KPS consists of a table of data. For example, the following example shows a simple KPS table structure:

Column	Type	Description
`email`	`String`	User email address. This is the primary key used to identify a row in the table.
`password`	`String`	User password. This confidential data is encrypted.
`firstName`	`String`	User first name.
`lastName`	`String`	User surname.
`age`	`Integer`	User age.

The following example shows some simple table data that follows this structure:

`email`	**`password`**	**`firstName`**	**`lastName`**	**`age`**
`jdoe@acme.com`	`*****`	`John`	`Doe`	`21`
`jbloggs@acme.com`	`*****`	`Joe`	`Bloggs`	`42`
`jdupont@acme.com`	`*****`	`Jean`	`Dupont`	`33`

Every KPS table is assigned alias in Policy Studio so that it can be referred to from a policy. For example, an alias used to identify the example table could be `customers`. In this example, the `email` column is the primary key for the table. You can use this primary key to look up and uniquely identify a row using a selector expression. For example, the following selector expression evaluates to `John`:

```
${kps.customers["jdoe@acme.com"].firstName}
```

The following selector expression evaluates to `42`:

```
${kps.customers["jbloggs@acme.com"].age}
```

For more details on selectors, see *Selecting configuration values at runtime*.

KPS Data Sources

A KPS provides a consistent interface to object data in different data sources. The API Gateway provides support for the following KPS data sources:

- Embedded Apache Cassandra database (default)—used across an API Gateway group to provide high availability in

a production environment
- JSON file system—for development use with a single API Gateway instance only
- Relational Database#—enables you to use your existing database (for example, Oracle, Microsoft SQL Server, DB2, MySQL, JPA, or database schema)

The relational database support includes the generic Java Persistence API (JPA) for serializing Java objects to a database, and an existing or custom database schema for writing beans that map keyed property attributes to the schema.

Note

File-based KPS stores are cached on startup. Database-based KPS stores are read each time.

You can configure a KPS using Policy Studio (for example, you can specify KPS collections, tables, and aliases). 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.

Adding a KPS Collection

A KPS collection is a group of KPS tables. To add a KPS collection, perform the following steps:

1. In the main Policy Studio tree, right-click **Key Property Stores**, and select **Add KPS Collection**.
2. Complete the following fields in the **Add KPS Collection** dialog:
   - **Name**:
     Enter the KPS name (for example, `Customer`).
   - **Alias prefix**:
     Enter an alias used to identify your KPS (for example, `customer`).
   - **Description**:
     Enter a text description of your KPS.
   - **Default data source**:
     Select one of the following from the list:
     - `Embedded (Cassandra)`
     - `File`
     - `SQL Database`

     Defaults to `Embedded (Cassandra)`.

The newly created KPS collection is displayed on the screen on the right.

Editing a KPS Collection

To edit a KPS collection, perform the following steps:

1. In the main Policy Studio tree, select the KPS collection that you wish to edit under **Key Property Stores**.
2. You can edit values for the KPS collection on the **Properties** tab (for example, **Name**, **Alias prefix**, or **Default data source**).
3. You can add or edit values for the KPS collection data sources on the **Data Sources** tab.
4. Click **OK**.

Adding a File Data Store

To add a file-based data store to a selected KPS collection, perform the following steps:

1. On the **Data Sources** tab, select **Add** > **Add File** at the bottom right of the screen.
2. Complete the following fields in the **Add File Data Source** dialog:

- **Name**:
  Enter the KPS name (for example, `Customer File Data Source`).
- **Description**:
  Enter a text description of your file data source.
- **Directory Path**:
  Enter the full directory path to the file (for example, `c:\kps\customer.json`).

3. Click **OK**.

Note

A file-based data store is for development use with a single API Gateway instance only, and should not be used in a production environment.

Adding a Database Data Store
To add an SQL database data store to a selected KPS collection, perform the following steps:

1. On the **Data Sources** tab, select **Add** > **Add Database** at the bottom right of the screen.
2. Complete the following fields in the **Add File Data Source** dialog:
   - **Name**:
     Enter the KPS name (for example, `Customer DB Data Source`).
   - **Description**:
     Enter a text description of your SQL database data source.
   - **Database Connection**:
     Click the button on the right, select a database connection in the dialog (for example, `Default Database Connection`), and click **OK**. You can add more database connections to the list by right-clicking the **Database Connections** node, and selecting **Add DB Connection**.
3. Click **OK**.

When finished editing the KPS collection, click the **Save** button in the top right corner of the screen.

Further Information
For more detailed information on using a KPS, please contact the Oracle Support Team with your queries.

Adding a KPS Table

To add a KPS table to a KPS collection, perform the following steps:

1. In the main Policy Studio tree, right-click a KPS collection (for example **Customer**), and select **Add Table**.
2. Complete the following fields in the **Add KPS Table** dialog:
   - **Name**:
     Enter the KPS name (for example, `Customer KPS`).
   - **Description**:
     Enter a text description of your KPS.
   - **Aliases**:
     Click **Add**, and enter an alias used to identify your KPS (for example, `customer`). Click **OK**. Every KPS must have at least one alias.
3. Click **OK**.

The newly created KPS table is displayed on the screen on the right.

Defining the KPS Table Structure

To define the KPS table structure, perform the following steps:

1. In the main Policy Studio tree, select a KPS table (for example **Customer**), and click the **Structure** tab in the screen on the right.
2. Click **Add** and complete the following fields in the **Add Property** dialog:
   - **Name**:
     Enter the name of the table column (for example, `email`).
   - **Type**:
     Select the data type from the list (for example, `java.lang.String`).
   - **Key**:
     For `java.util.Map` types, select the key type from the list (for example, `java.lang.Integer`).
   - **Value**:
     For `java.util.Map` and `java.util.Map` types, select the value type from the list (for example, `java.lang.Boolean`).
3. Click **OK**. The newly created KPS table structure is displayed on the screen on the right.
4. Select the **Primary Key** setting for `email` to make this field the primary key for the table.
5. Select the **Encrypted** setting for `password` to encrypt this field in the KPS data source.

Important

Encrypted fields are always read and written in the clear. For security, use HTTPS when accessing a KPS over its REST API.

Further Information

For more detailed information on Key Property Stores and on Apache Cassandra configuration, see the *Key Property Store User Guide* available from Axway Support.

Scripting Language Filter

Overview

The **Scripting Language** filter uses the Java Specification Request (JSR) 223 [http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/] to embed a scripting environment in the API Gateway 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.

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:

- You must implement the `invoke()` method. This 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.
- Do not perform attribute substitution as follows:

  ```
  msg.get("my.attribute.a") == msg.get("my.attribute.b")
  ```

  This is not thread safe and can cause performance issues.

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.

Adding your Script JARs to the Classpath

You must add your custom JavaScript or Groovy JAR files to your API Gateway classpath and to the list of runtime dependencies in Policy Studio.

Adding your script JARs to the API Gateway classpath
Because the scripting environment is embedded in the API Gateway engine, it has access to all Java classes on the API Gateway 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.

Adding your script JARs to Policy Studio
Your custom JavaScript or Groovy script JARs must also be compiled and validated in Policy Studio. To add your JARs files to the list of runtime dependencies in Policy Studio, perform the following steps:

1. In the Policy Studio main menu, select **Window** > **Preferences** > **Runtime Dependencies**.
2. Click **Add** to select your script JAR file(s) and any other required third-party JARs.
3. Click **Apply** when finished. Copies of these JAR files are added to the `plugins` directory in your Policy Studio installation.
4. You must restart Policy Studio and the server for these changes to take effect. You should restart Policy Studio using the `policystudio -clean` command.

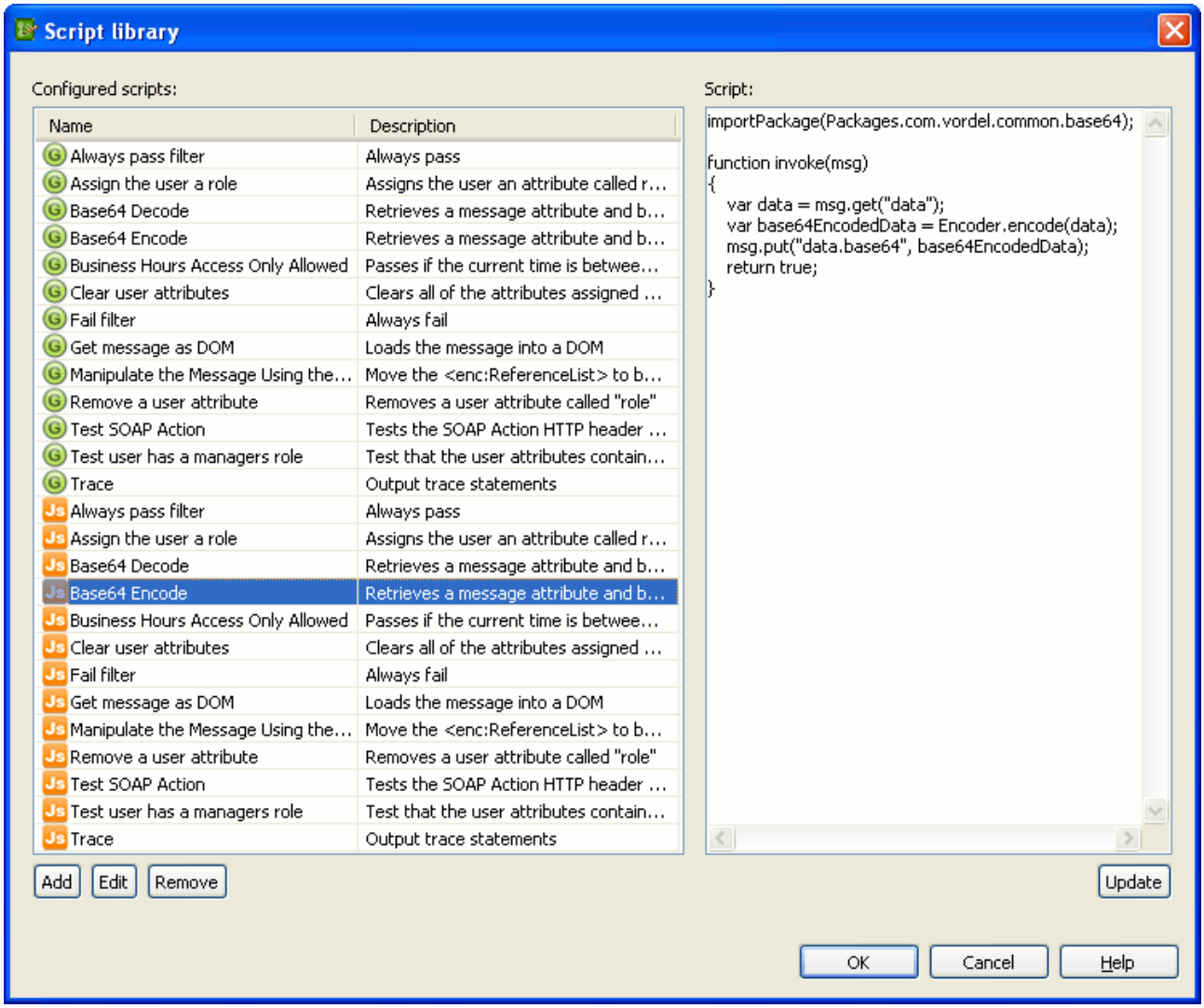
Configuring a Script Filter

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
Always pass filter
Always pass
Assign the user a role
Assigns the user an attribute called r...
Base64 Decode
Retrieves a message attribute and b...
Base64 Encode
Retrieves a message attribute and b...
Business Hours Access Only Allowed
Passes if the current time is betwee...
Clear user attributes
Clears all of the attributes assigned ...
Fail filter
Always fail
Get message as DOM
Loads the message into a DOM
Manipulate the Message Using the...
Move the <enc:ReferenceList> to b...
Remove a user attribute
Removes a user attribute called "role"
Test SOAP Action
Tests the SOAP Action HTTP header ...
Test user has a managers role
Test that the user attributes contain...
Trace
Output trace statements
Always pass filter
Always pass
Assign the user a role
Assigns the user an attribute called r...
Base64 Decode
Retrieves a message attribute and b...
Base64 Encode
Retrieves a message attribute and b...
Business Hours Access Only Allowed
Passes if the current time is betwee...
Clear user attributes
Clears all of the attributes assigned ...
Fail filter
Always fail
Get message as DOM
Loads the message into a DOM
Manipulate the Message Using the...
Move the <enc:ReferenceList> to b...
Remove a user attribute
Removes a user attribute called "role"
Test SOAP Action
Tests the SOAP Action HTTP header ...
Test user has a managers role
Test that the user attributes contain...
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;
}
Add
Edit
Remove
Update
OK
Cancel
Help

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 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 API Gateway can perform certificate validation.

API Gateway uses the following methods/protocols to validate certificates:

- Online Certificate Status Protocol (OCSP) – OCSP is an automated certificate checking network protocol. 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.
- Certificate Revocation List (CRL) – A CRL is a signed list indicating a set of certificates that are no longer considered valid (that is, revoked certificates) by the certificate issuer. 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.
- XML Key Management Services (XKMS) – XKMS is an XML-based protocol for (amongst other things) establishing the trustworthiness of a certificate over the Internet. 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 the validity of a client certificate using any of the following filters:

- *OCSP client*
- *Static CRL certificate validation*
- *Dynamic CRL certificate validation*
- *CRL LDAP validation*
- *XKMS certificate validation*

Note

To validate a certificate using either an OCSP request or CRL lookup, the issuing CA's certificate should be trusted by 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.

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 *Configure 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 API Gateway instance, 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, which apply at the API Gateway instance level. The default is no content encodings. You can also override this settings 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, which displays the active settings. 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:

- General settings in the *API Gateway Administrator Guide*.
- *Configure remote host settings*
- *Configure 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*
- Configuring Scheduled Reports in the *API Gateway Administrator Guide*

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:

```
//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:

```
//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.

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 *Configure HTTP services*.

Configuring Transparent Proxy Mode for Outgoing Calls

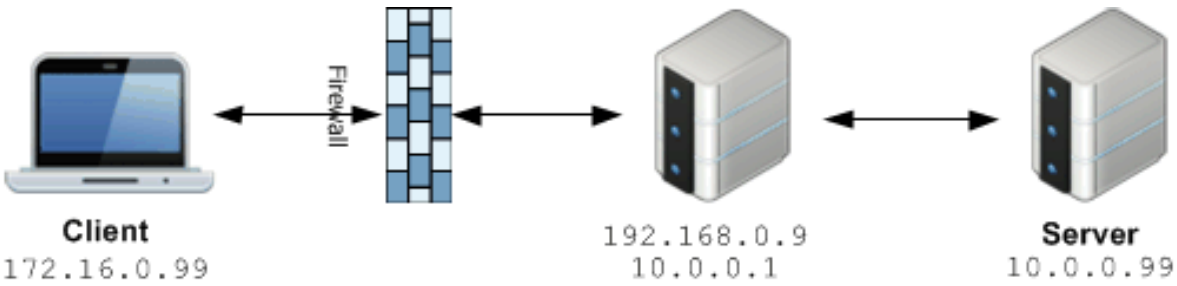
Transparent proxy mode for outgoing calls must be enabled at the level of a connection filter in a policy. To enable transparent proxy mode for outbound calls, perform the following steps:

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* .

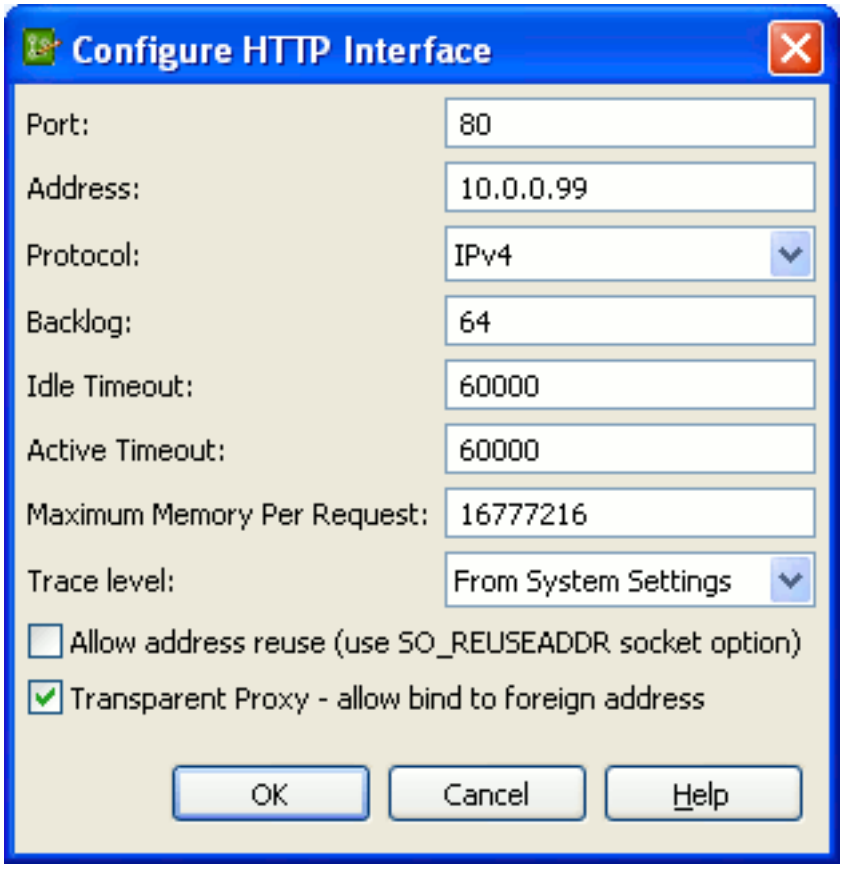
Configuration Example

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.

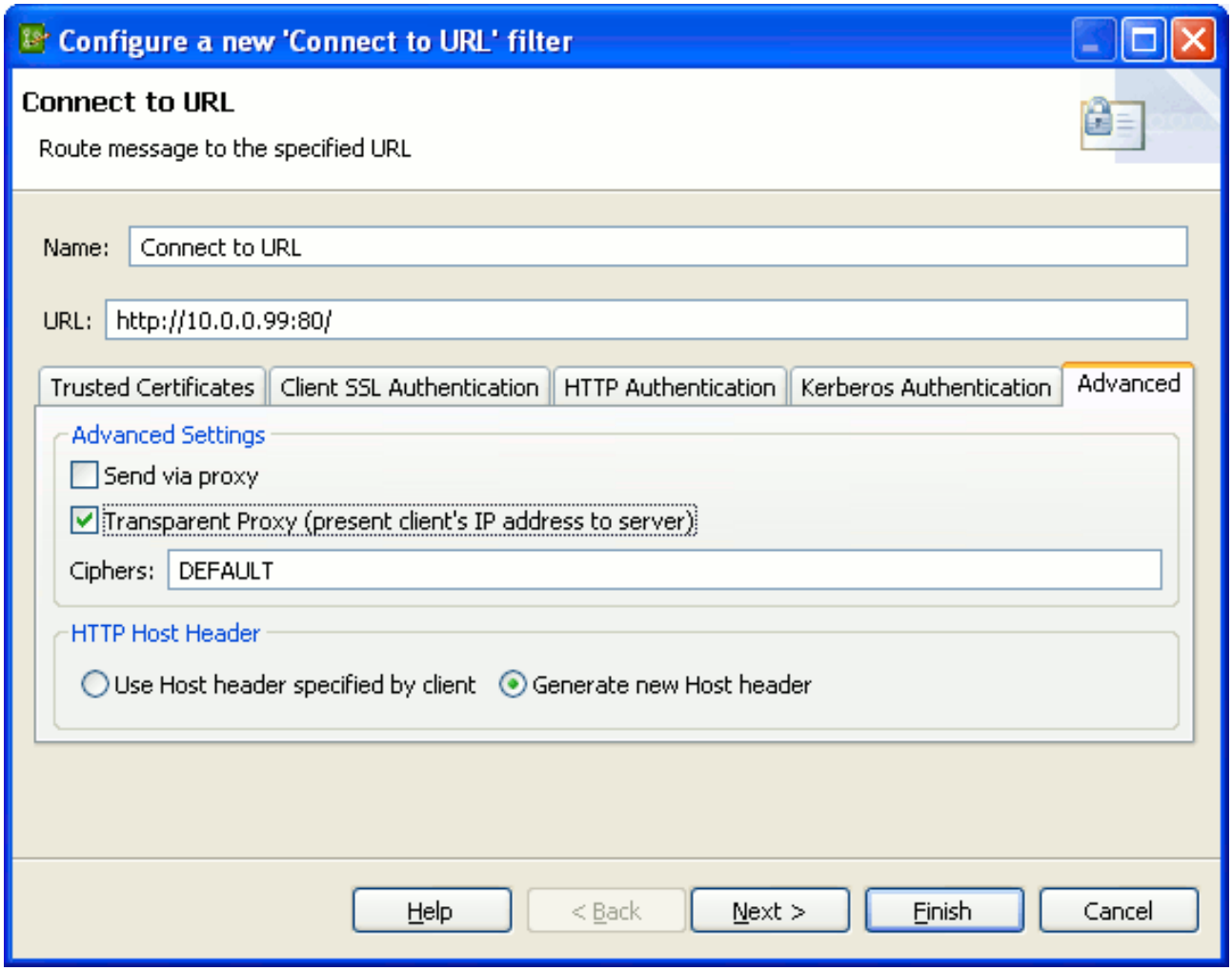
The API Gateway is multi-homed, and sits on both the `192.168.0.0/24` and `10.0.0.0/24` networks. It is configured with a listening interface at address `10.0.0.99:80`, with transparent proxy mode switched on, as shown in the following **Configure HTTP Interface** dialog:



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:



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

You can use WSDL files to register Web services in the **Web Service Repository** and to add WSDL documents and XML schemas to the global cache. 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 *Manage the 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 Policy Studio directly without registering a WSDL file. Under the **Business Services** node in the tree, right-click the **Web Service Repository** node, and select **Browse UDDI Registry**.

Introducing UDDI

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 Policy Studio. You can search the registry to retrieve the WSDL file for a particular service. 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 windows, 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	ntis-gov:naics:1997	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2

Categorization System	Name	tModel Key
(NAICS) 1997 Release		

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 `busines-`

sEntity, 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** window. 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 businesEntity 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 (`cat-`

UDDI Find Qualifier:	Description:
	egoryBag 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`
- `http://HOSTNAME:PORT/uddi/security`

These are the service endpoint URLs that 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 user name required to authenticate to the registry, if required.
Password	Enter the password for this user, if required.

The user name 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, Policy Studio sends this user name and password to the proxy using HTTP Basic authentication.
Password	Enter the password to use with the user name 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 can 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 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 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 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** window.
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 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 Policy Studio must authenticate to the registry), 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 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 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 Service Repository** using Web Services Description Language (WSDL) files. 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 service 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 *Manage the 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 Service Repository** node.
2. Right-click a WSDL node and select **Find in UDDI Registry** to launch the **Find WSDL** wizard.
3. In the **Find WSDL** dialog, 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*.
4. You can select an optional language **Locale** from the list. The default is `No Locale`.
5. Click **Next**. The **WSDL Found in UDDI Registry** window 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`).
6. 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).
7. Click the **Refresh** button to run the search again.
8. 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 Service Repository** to a UDDI registry, perform the following steps:

1. Expand the **Business Services** > **Web Service Repository** tree node.
2. Right-click a WSDL node and select **Publish WSDL to UDDI Registry** to launch the **Publish WSDL Wizard**.
3. Perform the steps in the wizard as 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 service repository, 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 API Gateway. The client can then retrieve the WSDL for the virtualized Web service from API Gateway, without knowing its real location.

This window enables you to optionally override the service address locations in the WSDL file with the virtualized addresses exposed by 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 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 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 API Gateway and exposed at `http://HOST:8080/TestService/StockQuote.svc`, where `HOST` is the machine on which 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 window 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 window 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 window 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 window is displayed only if mapped WSDL entities already exist in the UDDI registry. Otherwise, the wizard skips to step 5. This window 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 se-

	lected 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 window 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 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, 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>11.1.2.3.0</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="11.1.2.3.0">
    <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>11.1.2.3.0</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>11.1.2.3.0</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**
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**
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**
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**
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 ap-

	pear, 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 certificate.
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** **SSL 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** **SSL 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** **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 to 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**
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 element*

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 attribute 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 response. 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	*Retrieve attribute from SAML PDP*

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*

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*

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	*Attribute Authorization*

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 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*

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 API Gateway 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 Client
Description	Retrieves the revocation status of a certificate from an OCSP responder.
Category	Certificate
Required Attributes	**certificates**
Consumed Attributes	
Generated Attributes	

Tutorial	*OCSP client*

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	*Scan with 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 selector expression*

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	*Configure system alerts*

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 end-point. 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 **Return 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.