ORACLE
PEOPLESOFT

Oracle's PeopleTools PeopleBook

PeopleTools 8.52: PeopleSoft Integration Broker

June 2013

ORACLE

PeopleTools 8.52: PeopleSoft Integration Broker
SKU pt8.52tibr-b0613

Copyright © 1988, 2013, Oracle and/or its affiliates. All rights reserved.

Trademark Notice

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

License Restrictions Warranty/Consequential Damages Disclaimer

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

Warranty Disclaimer

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

Restricted Rights Notice

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

Hazardous Applications Notice

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Third Party Content, Products, and Services Disclaimer

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third party content, products and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third party content, products or services.

Contents

Preface

PeopleSoft Integration Broker Preface xxiii

PeopleSoft Integration Broker xxiii
PeopleBooks and the PeopleSoft Online Library xxiii

Chapter 1

Getting Started with PeopleSoft Integration Broker 1

PeopleSoft Integration Broker Overview 1
Implementing PeopleSoft Integration Broker 1
Other Sources of Information 4

Chapter 2

Understanding PeopleSoft Integration Broker 5

Introduction to PeopleSoft Integration Broker 5
Web Services 6
Integration Gateway 6
Integration Engine 6
Integration Gateway Architecture 7
Architecture Elements 7
Connectors 8
Gateway Manager 9
Gateway Services 9
Integration Engine Architecture 11
Service Operations 12
Service Operation Types 12
Operation Types 12
Inbound and Outbound Request Flows 14
Inbound Request Flow 15
Outbound Request Flow 17

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 3

Understanding Messaging 21

Messaging Types 21
Asynchronous Messaging 21
Brokers, Contractors and Queues 21
Messaging System Server Processes 22
Dispatchers and Handlers 23
Asynchronous Service Operation Publication 24
Asynchronous Service Operation Subscription 28
Synchronous Messaging 31
Synchronous Service Operation Publication 31
Synchronous Service Operation Subscription 33

Chapter 4

Understanding PeopleSoft Integration Broker Metadata 35

PeopleSoft Integration Broker Metadata 35
Order of Precedence for Creating Integration Metadata 36

Chapter 5

Understanding Supported Message Structures 39

Supported Message Structures 39
Integration Broker Message Structures 40
Internal Message Format for Request Messages 40
Internal Message Format for Response Messages 49
Local Compression 53
Accessing IBInfo Elements Using PeopleCode 54
PeopleSoft Rowset-Based Message Format 55
Understanding the PeopleSoft Rowset-Based Message Format 56
Rowset-Based Message Template 56
FieldTypes Section 57
MsgData Section 57
PSCAMA 59
Identifying Changes to Field-Level Attributes 61
PeopleSoft Timestamp Format 62
CDATA and Special Characters 62
Schema Restrictions 62
Rowset-Based Message Example 63

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Nonrowset-Based Message Structures 65
XML Messages 65
SOAP-Compliant Messages 66
Non-XML Files 67
Using Nonrowset-Based Messages in Service Operations Exposed as WSDL 68
Message Part Structures 69
Understanding Message Part Structures 69
Rowset-Based Message Parts 69
Nonrowset-Based Message Parts 72
Message Container Structures 72
Example 1: XML Schema of a Container Message with Rowset-Based Message Parts 72
Example 2: XML Schema of a Container Message with Nonrowset-Based Message Parts 73

Chapter 6

Managing Messages 75

Understanding Managing Messages 75
Message Definitions 75
Message Types 76
Naming Conventions for Message Metadata 77
Message Record Structure 77
Underlying Record Definitions 77
Fields Defined as Uppercase 77
Message Aliases and Message Versions 77
Restrictions for Modifying Messages 77
Searching for Message Definitions 78
Adding Message Definitions 79
Understanding Adding Message Definitions 79
Adding Rowset, Nonrowset or Part Message Definitions 79
Adding Document Message Definitions 84
Managing Rowset-Based Messages 87
Understanding Managing Rowset-Based Messages 87
Viewing Rowset-Based Message Structures 88
Inserting Root Records 91
Inserting Child and Peer Records 92
Specifying Record Aliases 94
Deleting Records 94
Excluding Fields from Messages 95
Specifying Field Name Aliases 96
Including Fields in CData Sections in Generated XML 96
Managing XML Message Schemas for Rowset-Based Messages 97
Enforcing Message Record and Field Aliases in Generated WSDL 99
Managing Nonrowset-Based Messages 100

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Managing Nonrowset-Based Messages 100
Adding XML Message Schemas to Nonrowset-Based Messages 100
Editing Nonrowset-Based XML Schemas 101
Deleting Nonrowset-Based XML Message Schemas 101
Managing Message Parts 103
Understanding Message Parts 103
Creating Part Messages 104
Distinguishing Blank from Zero in Rowset-Based Part Messages 104
Reusing Rowset-Based Message Parts 104
Understanding Reusing Rowset-Based Message Parts 104
Reusing Rowset-Based Message Parts by Reference 105
Managing Container Messages 109
Understanding Managing Container Messages 109
Understanding Including Level 0 Rows for Message Parts in Container Messages 110
Adding Message Parts to Container Messages 112
Adding and Getting Container Messages Attributes 115
Generating XML Message Schemas for Container Messages 119
Managing Document Messages 120
Viewing Service Operations that Reference Messages 120
Resolving Inconsistencies in Exported WSDL and WADL Documents 121
Understanding Using Project Copy and Exported WSDL and WADL 121
Viewing Services Operations with Exported WSDL/WADL Inconsistencies 122
Clearing Exported WSDL/WADL Status Flags 123
Renaming and Deleting Message Definitions 125
Renaming Message Definitions 126
Deleting Message Definitions 126
Deleting Messages During Upgrade 126

Chapter 7

Sending and Receiving Messages 127

Understanding Sending and Receiving Messages 127
Prerequisites for Sending and Receiving Messages 127
Messaging Process Flows 128
Understanding Integration PeopleCode 129
Sending and Receiving PeopleCode 129
Application Classes 131
Routing Methods 131
Messaging Methods 135
Messaging PeopleCode 142
Document PeopleCode 143
Generating and Sending Messages 144
Understanding Outbound Messaging 144

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Handling Outbound Asynchronous Message Transmission 145
Handling Outbound Asynchronous Request/Response Message Transmission 147
Handling Outbound Synchronous Transactions 148
Reading Exceptions for Outbound Synchronous Integrations 149
Overriding Synchronous Timeout Intervals at Runtime 150
Handling Cookies 151
Setting and Overriding Target Connector Properties at Runtime 151
Receiving and Processing Messages 155
Handling Inbound Asynchronous Transactions 156
Handling Inbound Asynchronous Request/Response Transactions 169
Handling Inbound Synchronous Transactions 170
Simulating Receiving Messages from External Nodes 173
Processing Inbound Errors 174
Validating Data 174
Using the Exit Built-in Function 175
Using Message Object Functionality With Nonrowset-Based Messages 177
Using the SetXMLDoc Method 177
Using the GetXMLDoc Method 177
Generating Test Messages 178
Working With Message Segments 178
Understanding Message Segments 178
Understanding PeopleCode used to Work with Message Segments 179
Configuring Nodes to Handle Segmented Messages 180
Creating Message Segments 181
Deleting Message Segments 184
Sending and Receiving Segmented Messages between PeopleSoft Systems 184
Sending and Receiving Segmented Messages to/from Third-Party Systems 185
Sending, Receiving, and Correlating Multiple Segmented Messages 187
Accessing Segments in Messages 189
Viewing Message Segment Data 190
Using Restartable Processing for Publishing Large Messages in Batch 190
Populating and Retrieving Document Data 192
Understanding Populating and Retrieving Document Data 192
Instantiating Document Objects 192
Populating Document Data 193
Retrieving Document Data 195
Sending and Receiving Binary Data 198
Understanding Sending and Receiving Binary Data 198
Sending MTOM-Encoded Binary Data 198
Receiving Binary Data 202
Using PeopleCode to Manage REST Service Operations 205
Using PeopleCode to Manage Provider REST Service Operations 206
Using PeopleCode for Consumer REST Service Operations 209
Generating Fully-Qualified URLs for REST Resources 211

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 8

Building Message Schemas 213

Understanding the Message Schema Builder 213
Message Schemas 213
Building, Importing, Modifying and Deleting Message Schemas 214
Selecting and Viewing Data in the Message Schema Builder 214
Selecting Data in the Message Schema Builder 214
Viewing Message Schema Details 216
Viewing XML Message Schema 218
Building Message Schemas for Rowset-Based Messages 218
Building a Message Schema for a Rowset-Based Message 219
Importing Message Schemas for Nonrowset-Based Messages 219
Importing a Message Schema for a Nonrowset-Based Message 219
Modifying Message Schemas 220
Modifying a Message Schema 220
Deleting Message Schemas 221
Understanding Deleting Message Schemas 221
Using the Message Schema Builder Page to Delete Message Schemas 222

Chapter 9

Managing Services 223

Understanding Managing Services 223
Common Elements Used in This Chapter 224
Accessing and Viewing Service Definitions 225
Accessing Service Definitions 225
Viewing WSDL Documents Generated for Services 226
Viewing Service Operation Information 229
Viewing Messages Defined for Service Operations 229
Adding Service Definitions 230
Understanding Naming Services and Service Aliases 230
Adding Service Definitions 230
Adding Service Operations to Service Definitions 232
Understanding Adding Service Operations to Service Definitions 232
Adding Existing Service Operations to Service Definitions 232
Defining New Service Operations for SOAP-Based Service Definitions 233
Creating and Managing Integration Groups 234
Understanding Integration Groups 234
Adding Integration Groups 234
Adding Services to Integration Groups 236

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Adding Integration Subgroups 237
Deleting Services from Integration Groups and Integration Subgroups 238
Renaming and Deleting Integration Groups 238
Copying Integration Groups Using Project Copy 240
Restricting and Enabling Write Access to Service Definitions 240
Understanding Restricting Write Access to Service Definitions 240
Restricting Write Access to Service Definitions 241
Enabling Write Access to Service Definitions 242
Renaming and Deleting Service Definitions 242
Renaming Service Definitions 243
Deleting Service Definitions 244
Activating and Deactivating Service Definitions in Bulk 244

Chapter 10

Managing Service Operations 247

Understanding Managing Service Operations 247
Service Operations 248
Services Operation Types 248
Naming Conventions for Service Operation Metadata 248
Service Operation Aliases 249
Service Operation Versions 249
Monitoring Service Operations 249
Accessing and Viewing Service Operation Definitions 250
Accessing Service Operation Definitions 250
Viewing Service Operation Definitions 252
Adding Service Operation Definitions 255
Configuring Service Operation Definitions 255
Specifying General Service Operation Information 256
Defining Service Operation Version Information 256
Adding Handlers to Service Operations 259
Adding Routing Definitions 259
Activating and Inactivating Routing Definitions 260
Setting Permissions to Service Operations 260
Understanding Setting Permission to Service Operations 260
Setting Permission Access to Service Operations 260
Managing Service Operation Versions 261
Creating Service Operation Versions 261
Using Non-Default Service Operation Versions 262
Attaching Files to Service Operations 262
Understanding Attaching Files to Service Operations 262
Using the FTP Attachment Utility 263
Sending Attachment Information with Service Operations 263

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Processing Attachment Information Included in Service Operations 264
Assigning Multiple Queues to Process Service Operations 265
Understanding Assigning Multiple Queues to Process Service Operations 266
Enabling Multi-Queue Service Operation Processing 266
Specifying Multiple Queues to Process Service Operations 266
Invoking Multiple Service Operations 267
Renaming and Deleting Service Operations 268
Renaming Service Operations 269
Deleting Service Operations 269

Chapter 11

Managing REST Services 271

Understanding Managing REST Services 271
Common Elements Used in this Chapter 271
Accessing and Viewing REST Service Definitions 273
Accessing REST Service Definitions 273
Viewing WADL Documents for REST Services 276
Viewing REST Service Operation Information 278
View Messages Defined for REST Service Operations 278
Adding REST Service Definitions 278
Understanding Naming Services and Service Aliases for REST Service Definitions 279
Adding REST Service Definitions 279
Adding Service Operations to REST Service Definitions 281
Understanding Adding Service Operations to REST Service Definitions 281
Adding Existing Service Operations to REST Service Definitions 281
Adding New Service Operation Definitions for REST Services 281
Restricting and Enabling Write Access to REST Service Definitions 284
Renaming and Deleting REST Service Definitions 284
Activating and Deactivating REST Service Definitions in Bulk 285

Chapter 12

Managing REST Service Operations 287

Understanding Managing REST Service Operations 287
REST Service Operations 288
REST Service Operations Types 288
REST Methods 288
REST Resource Definitions 288
REST Messages 290
Naming Conventions for REST Service Operation Metadata 291
REST Service Operation Aliases 291

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

REST Service Operation Versions 291
WADL Node 291
Monitoring REST Service Operations 291
Accessing and Viewing REST Service Operation Definitions 292
Accessing REST Service Operation Definitions 292
Viewing REST Service Operation Definitions 294
Configuring REST Service Operation Definitions 297
Specifying General Service Operation Information 297
Managing REST Resource Definitions 298
Defining REST Service Operation Version Information 304
Managing Provider REST Service Operations 307
Managing Target Connectors for Provider REST Service Operations 307
Managing Messages for Provider REST Service Operations 308
Securing Provider REST Service Operations 308
Adding Handlers to Provider REST Services Operations 308
Managing Routing Definitions for Provider REST Service Operations 309
Defining Routing Header Properties for Provider REST Service Operations 309
Managing Consumer REST Service Operations 311
Adding Handlers to Consumer REST Service Operations 311
Manage Routing Definitions for Consumer REST Service Operations 312
Securing Consumer REST Service Operations 312
Managing REST Service Operation Versions 315
Setting Compression for REST Service Operations 315
Understanding Setting Compression for REST Service Operations 315
Setting Compression for Provider REST Service Operations 315
Defining Compression for Consumer REST Service Operations 318
Renaming and Deleting REST Service Operations 320

Chapter 13

Managing Service Operation Queues 321

Understanding Service Operation Queues 321
Adding Queue Definitions 321
Applying Queue Partitioning 324
Understanding Queue Partitioning 324
Selecting Partitioning Fields 325
Renaming and Deleting Queues 326
Renaming Queue Definitions 327
Deleting Queue Definitions 328
Deleting Queues During Upgrade 328

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 14

Enabling Runtime Message Schema Validation 329

Understanding Message Schema Validation 329
Message Schema Validation 329
Message Schema Validation and Transformations 329
Message Schema Validation and Part Messages 330
Prerequisites for Validating Message Schemas 330
Selecting Service Operations 330
Selecting a Service Operation 330
Viewing Defined Message Schemas 332
Viewing XML Schemas Defined for Messages 332
Enabling Runtime Message Schema Validation 334
Using the Service Schema Validation Page to Enable Runtime Message Schema Validation 334
Using the Service Operations page to Enable Runtime Message Schema Validation 334

Chapter 15

Creating Component Interface-Based Services 335

Understanding Creating Component Interface-Based Services 335
Naming Conventions Integration Metadata Created 335
User-Defined Method Restrictions 336
Impact of Changing Component Interfaces 337
Prerequisites 337
Selecting Component Interfaces to Expose as Services 337
Selecting Component Interface Methods to Include as Service Operations 339
Generating Component Interface-Based Services 341
Generating Services and Service Operations from Component Interface Methods 341
Inheriting Component Interface Security Permission Lists 343
Adding Message Names and Descriptions to Generated Service Operations 344
Viewing Component Interface-Based Service Definitions 344

Chapter 16

Managing Service Operation Handlers 347

Understanding Service Operation Handlers 347
Service Operation Handler Types 347
Handler Types and Messaging Types 348
Understanding Implementing Handlers 349
Adding Handlers to Service Operations 350

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Adding Handler Definitions to Service Operations 350
Adding a Handler to a Service Operation 351
Specifying General Handler Details 352
Implementing Handlers Using Application Classes 353
Understanding Implementing Handlers Using Application Classes 353
Developing Application Classes for Implementing Handlers 354
Specifying Application Class Implementation Details 355
Implementing Handlers Using Application Engine Programs 356
Understanding Implementing Handlers Using Application Engine Programs 356
Specifying Application Engine Handler Implementation Details 357
Retrieving Service Operation Content from Application Engine Programs 358
Viewing Subscription Contract Status 359
Implementing Handlers Using Component Interfaces 360
Understanding Implementing Handlers Using Component Interfaces 360
Specifying Component Interface Handler Implementation Details 361
Implementing Handlers Using Bulk Load Processing 361
Understanding Implementing Handlers Using the Bulk Load Handler 361
Enabling Transactional Rollback 363
Enabling Table Truncation 364
Specifying XML Record Attribute Values 365
Adding Data Structures for Nonrowset-Based Messages 365
Implementing Handlers Using Deprecated PeopleCode Handlers 367
Understanding the Deprecated PeopleCode Handler 368
Deleting Deprecated PeopleCode Handlers 368

Chapter 17

Managing Service Operation Routing Definitions 371

Understanding Routing Definitions 371
Routing Definitions 371
Routing Types 371
Defining Routing Definitions 372
Methods for Generating and Defining Routing Definitions 373
Routing Definition Naming Conventions 374
Routing Definition External Aliases 375
Service Operation Mapping 375
Graphical Routings View 375
Integration Status 376
Managing System-Generated Routing Definitions 376
Understanding Managing System-Generated Routing Definitions 376
Viewing System-Generated Routing Definition Status 376
Initiating System-Generated Routing Definitions 377
Regenerating System-Generated Routing Definitions 379

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Adding Routing Definitions 379
Understanding Adding Routing Definitions 379
Adding Routing Definitions Using the Routings Component 380
Adding Routing Definitions From Service Operation Definitions 381
Adding Routing Definitions Using the Nodes Component 382
Configuring Routing Definitions 382
Defining General Routing Information 382
Defining Routing Parameters 388
Defining and Overriding Gateway and Connector Properties 393
Defining Routing Properties 395
Using Introspection to Create Routing Definitions 396
Understanding Using Introspection to Create Routing Definitions 397
Prerequisites for Using Introspection to Create Routing Definitions 397
Selecting Service Operations for Which to Create Routing Definitions 397
Selecting Nodes to Introspect 399
Selecting Routing Definitions to Generate 400
Viewing Introspection Results 402
Activating and Inactivating Routing Definitions 403
Understanding Activating and Inactivating Routing Definitions 403
Activating and Inactivating Routing Definitions in the Routing Component 403
Activating and Inactivating Routing Definitions in the Service Operations Component 403
Activating and Inactivating Routing Definitions in the Nodes Component 404
Viewing Routing Definitions in Graphical Format 404
Common Elements Used to View Routing Definitions in Graphical Format 404
Viewing a Routing Definition in Graphical Format 406
Viewing Integration Status and Activating Integration Metadata 407
Understanding Viewing Integration Status and Activating Integration Metadata 407
Viewing Inactive Integration Metadata 408
Activating Integration Metadata Using the Integration Status Page 408
Retrieving Routing Properties Programmatically 409
Searching for Duplicate External Routing Aliases 410
Renaming and Deleting Routing Definitions 411
Renaming Routing Definitions 412
Deleting Routing Definitions 413
Deleting Duplicate Routing Definitions 413

Chapter 18

Applying Filtering, Transformation and Translation 415

Understanding Filtering, Transformation, and Translation 415
Understanding Transform Programs 416
Transform Programs 416
Transformation Programming Languages 417

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Third-Party Considerations 418
Defining Transform Programs 418
Understanding Defining Transform Programs 418
Defining a Transform Program 419
Developing Transform Programs Using PeopleSoft Application Engine 421
Understanding Developing Transform Programs Using PeopleSoft Application Engine 421
Inserting Steps and Actions into Transform Programs 422
Making Working Storage Data Available Globally 423
Preserving Record and Field Aliases 424
Tracing Transform Programs 425
Developing Transforms Using Oracle XSL Mapper 425
Understanding Oracle XSL Mapper 426
Development Considerations 426
Prerequisites for Developing Transforms Using Oracle XSL Mapper 426
Installing Oracle XSL Mapper 427
Specifying the Installation Path and Classpath for Oracle XSL Mapper 427
Launching Oracle XSL Mapper 429
Accessing Oracle JDeveloper Documentation and Online Resources 431
Navigating in Oracle XSL Mapper 432
Mapping Records and Fields 435
Deleting Record and Field Maps 436
Viewing Raw XSLT Code 437
Testing XSL Maps 437
Adding and Modifying XSL Map Code 438
Invoking Transform Programs 439
Accessing Transform Message Data 440
Renaming or Deleting Transform Programs 442
Filtering Messages 442
Understanding Message Filtering 442
PeopleCode Filtering Example 443
Applying Transformations 444
Understanding Transformation 444
Using XSLT for Transformation 445
Applying Message Transformations at the Integration Gateway 446
Understanding Applying Message Transformations at the Integration Gateway 447
Developing and Implementing Gateway-Based Transformation Programs 447
Setting Integration Gateway Properties for Gateway-Based Transformations 448
Understanding Logged Errors 449
Performing Data Translation 450
Understanding Data Translation 450
Defining Codeset Groups 452
Defining Codesets 453
Defining Codeset Values 454
Importing and Exporting Codesets Between Databases 456
Deleting Codesets 457

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Using XSLT for Data Translation 457
XSLT Translation Example 460
PeopleCode Translation Example 462
Rejecting Transformation Programs 464
Terminating Transformation Programs 464

Chapter 19

Managing Error Handling, Logging, Tracing, and Debugging 467

Understanding Error Handling, Logging, Tracing and Debugging 467
Understanding Integration Gateway Error Handling 467
Target Connector Error Handling 467
Listening Connector Error Handling 468
Integration Gateway Exception Types 468
Managing Integration Gateway Message and Error Logging 470
Understanding Message and Error Logging 470
Setting Up Message and Error Logging 470
Viewing Non-English Characters in Integration Gateway Log Files 470
Managing Message Logging 471
Managing Error Logging 472
Managing Application Server Logging and Tracing 473
Debugging Integrations 474
Debugging Handler PeopleCode 474
Handling Common Issues 475

Chapter 20

Providing Services 479

Understanding Providing Services 479
Understanding the Provide Web Service Wizard 479
Understanding Providing WSDL Documents 480
WSDL Features 480
WSDL Document Specifications 480
Supported Operation Types for WSDL Documents 481
Requirements for Nonrowset-Based Message Schemas 481
Locations for Publishing WSDL Documents 482
UDDI Repositories and Endpoints 482
WSDL URL Formats 483
Provided WSDL Documents 483
PartnerLinkType Support 493
Understanding WADL Documents 495
Supported Operation Types for WADL Documents 495

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Locations for Publishing WADL Documents 495
WADL URL Format 496
Provided WADL Documents 496
WSDL Document and WADL Document Versioning 500
Prerequisites for Providing Services 500
Common Elements Used in This Chapter 501
Providing Services 502
Understanding Using the Provide Web Service Wizard 502
Step 1: Select Services to Provide 503
Step 2: Select Service Operations 504
Step 3: View WSDL Documents or WADL Documents 505
Step 4: Specify Publishing Options 507
Step 5: View the WSDL/WADL Generation Log 509
Accessing Generated WSDL Documents and WADL Documents 510
Using WSDL and WADL URLs To Access Generated WSDL and WADL Documents 510
Using the WSDL Repository to Access Generated WSDL and WADL Documents 510
Deleting WSDL and WADL Documents 512
Understanding Deleting WSDL Documents 512
Deleting a WSDL or WADL Document 512

Chapter 21

Consuming Services 515

Understanding Consuming Services 515
Understanding the Consume Web Service Wizard 515
Consume Web Service Wizard Features 515
Operation Types Supported 515
Sources for Consuming WSDL Document 516
Integration Metadata Created by the Consume Web Service Wizard 516
Messages Generated 517
Fault Messages 518
Multiple Root Elements in Message Schemas 518
Delivered Queues and Nodes 518
Binding Style of Consumed WSDL Documents 518
Working with Asynchronous Request/Response Service Operations 519
Prerequisites for Consuming Services 519
Common Elements Used in This Chapter 519
Using the Consume Web Service Wizard 520
Step 1: Select WSDL Source 521
Step 2: Select Service 523
Step 3: Select Service Ports 523
Step 4: Select Service Operations 524
Step 5: Convert Asynchronous Operations 525

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step 6: Rename Operation Messages 526
Step 7: Select a Queue for Asynchronous Operations 528
Step 8: Select the Receiver Node 529
Confirm and View Results 530
Accessing Integration Metadata for Consumed Services 531

Chapter 22

Integrating with BPEL Process-Based Services 535

Understanding Integrating with BPEL Processes 535
Oracle BPEL Process Manager 535
PeopleSoft-Delivered Application Classes for BPEL Integrations 536
Monitoring BPEL Process-Based Integrations 536
Securing BPEL Process-Based Integrations 537
Prerequisites for Integrating with BPEL Processes 537
Configuring the PeopleSoft-Delivered BPEL Node 538
Consuming BPEL Process–Based Services 539
Understanding Consuming BPEL Process-Based Services 539
Deploying BPEL Processes 540
Consuming WSDL Documents from BPEL Processes 540
Consuming Synchronous BPEL Operations 540
Consuming Asynchronous Request/Response BPEL Operations 542
Consuming Asynchronous Fire-and-Forget (One-Way) BPEL Operations 545
Providing PeopleSoft Services to BPEL Processes 548
Understanding Providing PeopleSoft Services to BPEL Processes 548
Providing Synchronous PeopleSoft Operations to BPEL Processes 548
Providing Asynchronous PeopleSoft Request/Response Operations to BPEL Processes 552

Chapter 23

Integrating with Oracle Mediator and Oracle ESB-Based Services 555

Understanding Integrating with Oracle Mediator and Oracle ESB-Based Services 555
Oracle Mediator and Oracle ESB 555
Software Components 556
Securing Oracle Mediator and Oracle ESB-Based Services 557
Prerequisites for Integrating with Oracle Mediator/ESB–Based Services 558
Consuming and Invoking Oracle Mediator/ESB-Based Services 558
Understanding Consuming and Invoking Oracle Mediator/ESB-Based Services 559
Providing Oracle Mediator/ESB–Based Services for Consuming in PeopleSoft 562
Consuming Oracle Mediator/ESB-Based Services 563
Invoking Synchronous Oracle Mediator/ESB-Based Services 563

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Invoking Asynchronous Oracle Mediator/ESB-Based Services 565
Invoking Asynchronous Request/Response Oracle Mediator-Based Services 568
Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB 570
Understanding Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB 570
Prerequisites for Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB 570
Providing PeopleSoft Services 570
Invoking PeopleSoft Services in Oracle Mediator/ESB 571

Chapter 24

Using the Inbound File Loader Utility 573

Understanding the Inbound File Loader Utility 573
File Processing 573
Understanding Development Activities 575
General Development Activities 575
Development Activities for PeopleSoft Integration Broker Processing 575
Creating File Layout Definitions 576
Development Activities for Application Class Processing 577
Prerequisites for Using the Inbound File Loader Utility 580
Setting Up Inbound File Loader Processing Rules 580
Understanding Setting Up Inbound File Loader Processing Rules 580
Setting Up Inbound File Loader Processing Rules 580
Initiating File Processing 583
Understanding Initiating File Processing 583
Initiating Inbound Flat File Processing 584
Testing Inbound Flat File Processing 585

Chapter 25

Copying Integration Metadata between PeopleSoft Databases 587

Copying Integration Metadata Between PeopleSoft Databases 587
Understanding Copying Integration Metadata Between PeopleSoft Databases 587
Understanding Data Dependencies and Relationships for Copying Data 588
Using Data Mover Scripts to Copy Message Schema and WSDL Data 590
Converting WSDL Documents and Message Schemas to Managed Objects 591
Understanding Converting WSDL Documents and Message Schema to Managed Objects 591
Using the Metadata Convert/Schema Convert Page 592
Converting WSDL Documents to Managed Objects 593
Converting Message Schemas to Managed Objects 593
Deleting Data from the Deprecated Data Repository 594
Managing Nodes Copied Between Databases and Upgraded from Earlier PeopleTools Releases 595

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Appendix A

Integration Scenarios 597

Understanding Integration Setup 597
Integrating with PeopleSoft Integration Broker Systems 602
Understanding This Scenario 602
Configuring the System for This Scenario 603
Integrating with PeopleSoft Integration Broker Systems Through Firewalls 604
Understanding This Scenario 604
Configuring the System for This Scenario 606
Integrating with PeopleSoft Integration Broker Systems by Using Hubs 608
Understanding This Scenario 608
Understanding Hub Routing Types 609
Configuring Generic-Routing Hubs 610
Configuring Sender-Specified Routing Hubs 613
Integrating with Third-Party Systems 615
Understanding This Scenario 616
Configuring the System for This Scenario 617
Integrating with Third-Party Systems by Using Remote Gateways 618
Understanding This Scenario 619
Sending Messages to Third-Party Systems 622
Receiving Messages from Third-Party Systems 623
Integrating with PeopleTools 8.47 and Earlier PeopleTools 8.4x Systems 626
Understanding This Scenario 626
Configuring the System for This Scenario 627
Integrating with PeopleTools 8.1x Systems 629
Understanding This Scenario 629
Configuring the System for This Scenario 630

Appendix B

Transformation Example: Integration Between Two PeopleSoft Nodes 633

Understanding This Appendix 633
Using the Example 633
Integration Metadata for This Example 634
Creating Message Definitions 634
Message Definition: PeopleSoft SCM Node 634
Message Definition: PeopleSoft CRM Node 635
Setting Up the Codesets 637
Setting Up the Transformation 639
XSL Walkthrough 641

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Transformation Processing: First Pass 642
Transformation Processing: Second Pass 645
Testing the Transformation 646

Appendix C

Understanding Migrated Integration Metadata 647

Understanding Migrated Integration Metadata 647
Node Objects 647
Channel Objects 648
Message Objects 648
Node Transaction and Relationship Objects 648
Understanding Migrated Integration PeopleCode 649
Application Classes 650
PeopleCode Methods 650
Built-In Functions 651
Other Migrated Constructs 651
Special Characters 651
Correcting Integration PeopleCode That Did Not Migrate 651
Understanding Integration PeopleCode That Did Not Migrate 651
Correcting Non-Migrated Integration PeopleCode 652

Appendix D

Setting PS_FILEDIR and PS_SERVDIR Environment Variables 655

Understanding Setting PS_FILEDIR and PS_SERVDIR Environment Variables 655
Setting PS_FILEDIR and PS_SERVDIR in Microsoft Windows Environments 655
Setting PS_FILEDIR and PS_SERVDIR in UNIX Environments 656

Index 657

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleSoft Integration Broker Preface

This preface provides an overview of the PeopleSoft Integration Broker PeopleBook.

PeopleSoft Integration Broker

PeopleSoft Integration Broker facilitates integrations with PeopleSoft and third-party systems. It features a services-oriented architecture that enables you to expose PeopleSoft business logic as services to PeopleSoft and third-party systems. It also allows you to consume and invoke services from other PeopleSoft and third-party systems. The PeopleSoft Integration Broker services framework supports synchronous and asynchronous messaging, and enables you to use a variety of communication protocols, while managing message structure, message content, and transport disparities

This PeopleBook describes the procedures for using PeopleSoft Integration Broker to develop and administer services. These procedures include defining services, service operations, messages, queues, routings, and transformations.

This PeopleBook also discusses developing the necessary PeopleCode to send, receive, and route service operations. It also discusses how to develop PeopleCode and XSLT code to filter, transform, and translate message content.

Though not intended for use in production environments, code examples are included throughout this PeopleBook to help illustrate technology concepts and the use of product features.

Other PeopleBooks discuss configuring and administering the integration system, monitoring integrations, and testing integrations.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration PeopleBook

PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook

PeopleTools 8.52: Integration Broker Testing Tools and Utilities PeopleBook

PeopleBooks and the PeopleSoft Online Library

A companion PeopleBook called *PeopleBooks and the PeopleSoft Online Library* contains general information, including:

- Understanding the PeopleSoft online library and related documentation.
- How to send PeopleSoft documentation comments and suggestions to Oracle.
- How to access hosted PeopleBooks, downloadable HTML PeopleBooks, and downloadable PDF PeopleBooks as well as documentation updates.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Understanding PeopleBook structure.
- Typographical conventions and visual cues used in PeopleBooks.
- ISO country codes and currency codes.
- PeopleBooks that are common across multiple applications.
- Common elements used in PeopleBooks.
- Navigating the PeopleBooks interface and searching the PeopleSoft online library.
- Displaying and printing screen shots and graphics in PeopleBooks.
- How to manage the locally installed PeopleSoft online library, including web site folders.
- Understanding documentation integration and how to integrate customized documentation into the library.
- Application abbreviations found in application fields.

You can find *PeopleBooks and the PeopleSoft Online Library* in the online PeopleBooks Library for your PeopleTools release.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 1

Getting Started with PeopleSoft Integration Broker

This chapter provides an overview of PeopleSoft Integration Broker and discusses considerations for how to:

- Plan the integration architecture.
- Plan integrations.
- Determine security.
- Plan for support.
- Assess staff skills.

PeopleSoft Integration Broker Overview

This PeopleBook describes using PeopleSoft Integration Broker to:

- Perform asynchronous and synchronous messaging among internal systems and third-party systems.
- Expose PeopleSoft business logic as web services to PeopleSoft and third-party systems.
- Consume and invoke web services from third-party and PeopleSoft systems.

Implementing PeopleSoft Integration Broker

This section provides information to consider before you begin to use PeopleSoft Integration Broker.

Planning the Integration Architecture

The two major components of PeopleSoft Integration Broker are the integration gateway and the integration engine. The integration gateway is a platform that manages the receipt and delivery of messages passed among systems through PeopleSoft Integration Broker. The integration engine is an application server process that routes messages to and from PeopleSoft applications as well as transforms the structure of messages and translates data according to specifications that you define.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When planning the integration architecture, evaluate historical integration data, current data, as well as expected growth and increased traffic. Consider the number of interfaces you have in production and how much system resources they use. Also consider how many of the interfaces will be nightly batch file loads, versus how many will be real-time service-based integrations. Devise simulated real-life integration scenarios where you can estimate the volume and the size of the transactions to a certain degree. Then use this information for benchmarking and stress testing–which should lead to performance tuning, hardware sizing, and so on.

Planning Integrations

In planning the integrations to develop and execute, consider the following:

- Real-time integrations or scheduled integrations.

  Determine if your business needs are best served with real-time integration or scheduled integrations.

  Scheduled batch processing and file loads are discussed in other PeopleBooks.

  See *PeopleTools 8.52: PeopleSoft Process Scheduler*, "Getting Started With PeopleSoft Process Scheduler" and *PeopleTools 8.52: Application Engine*, "Getting Started With Application Engine."

- Inventory the integrations to develop.

  Determine the systems and applications that will participate in each integration.

  Consider dependencies on other systems owned by other groups having concurrent releases, and data dependencies within the context of synchronizing data between systems. Also consider if you will need permission from business owners to integrate with their systems.

- Generic integrations.

  Consider if you can develop generic integrations. Perhaps in your current environment only two systems need to exchange information and they do so in a proprietary way. But consider that one day perhaps additional systems in your enterprise may also need to exchange that information with the source system. Will you need to develop transformations for systems that will be integrating later on? Can you develop the integration in a way so that other systems will be able to consume the service or subscribe to the information without requiring complex transformations?

- Determine the integrations that will require synchronous messaging and those that will asynchronous messaging.

  In PeopleSoft Integration Broker synchronous integrations, all processing stops until a response is received. In PeopleSoft Integration Broker asynchronous integrations, each request is placed in a queue and is processed as soon as the system can accommodate the request.

  Perhaps you may need to stop the processing of fulfilling an order until the system verifies that all requested items are available in inventory. In such a case, a synchronous integration is needed.

  However the processing of support tickets probably should not stop if a system uses integration to add a new ticket to a queue. In such a scenario, an asynchronous integration might be appropriate.

- Prioritize integration development.

  Plan to develop mission-critical integrations first, standard integrations next, and nice-to-have integrations last.

- Determine if data will need transformation or translation.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Plan on using integration simulation tools.

  Plan on using simulation tools such as PeopleSoft Send Master to simulate integrations with external systems that are not under your control. Even when you do control all systems that are being integrated, if you can't get the integration to work using Send Master, you definitely won't be able to get it working from the external system. Test integrations using Send Master before spending hours debugging a system.

  See *PeopleTools 8.52: Integration Broker Testing Utilities and Tools*, "Getting Started with PeopleSoft Integration Testing Utilities and Tools."

Determining Security

Unlike a public web service on the internet that retrieves a stock quote for a given ticker symbol, the web services and integrations in your PeopleSoft applications can expose sensitive information such as financial data. PeopleSoft Integration Broker facilitates transfer of information between systems; however, a security analyst must evaluate security requirements for each individual integration.

For example, security requirements might differ when interfacing with credit card processing vendors, versus publishing salary information out of human resources, versus synchronizing business units between applications, and so on.

Perhaps certain information should be available to the public, including systems outside of your company, such as how many inventory items are available for sale. Other information might be restricted to internal employees only, internal application systems only, or perhaps only certain users of a particular application system.

PeopleSoft Integration Broker allows you to secure each individual integration to the level of security required, as well as all integration data flowing over the wire.

Planning for Support

Develop a support plan for after "go-live." In doing so, consider the following:

- Determine who in your organization will support integration development and administration.
- Determine the type of error-notification and exception handling to implement to meet your support requirements. Consider that while system administrators can resolve communication failure between machines, they may not be able to resolve errors resulting from one system transmitting bad data to another. Analyst intervention may be required to correct the data. Stronger validation at point of data entry will result in fewer calls to a functional analyst to resolve integration issues.

Assessing Staff Skills

Assess the skills of the people who will perform development and administrative functions.

Developers working on the implementation of PeopleSoft Integration Broker should have familiarity, training or experience in the following PeopleSoft areas:

- PeopleTools.
- PeopleCode.
- Application Engine.

In addition, developers should have an understanding and research capabilities in:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Extensible Markup Language (XML).
- XML schema.
- Simple Object Access Protocol (SOAP).
- Hypertext Transfer Protocol (HTTP).
- Web Services Description Language (WSDL).
- Web Application Description Language (WADL).
- Universal Description, Discovery and Integration (UDDI) standard.
- Java programming language.

Other Sources of Information

In addition to the implementation considerations presented in this chapter, take advantage of all PeopleSoft sources of information, including the installation guides, release notes, PeopleBooks, curriculum, and red papers.

See Also

"PeopleSoft Integration Broker Preface," page xxiii

PeopleTools 8.52: Getting Started with PeopleTools PeopleBook

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 2

Understanding PeopleSoft Integration Broker

This chapter provides overview information about PeopleSoft Integration Broker and discusses:

- Integration gateway architecture.
- Integration engine architecture.
- Services.
- Inbound and outbound request flows.

Important! PeopleSoft Integration Broker interacts with a wide variety of third-party products. This PeopleBook is not an authoritative source of information about any third-party product. Most third-party products are delivered with their own documentation, which you should use as the primary source for information about them. This PeopleBook provides guidance that enables you to determine the configuration settings that PeopleSoft Integration Broker requires to work with third-party products. It does not address all configuration permutations. Examples of settings and data relative to a third-party product may not be correct for your particular situation. To properly configure PeopleSoft Integration Broker, you must apply your own expertise and obtain the most accurate and current information about third-party products.

Introduction to PeopleSoft Integration Broker

PeopleSoft Integration Broker is a middleware technology that:

- Performs asynchronous and synchronous messaging among internal systems and third-party systems.
- Exposes PeopleSoft business logic as web services to PeopleSoft and third-party systems.
- Consumes and invokes web services from third-party and PeopleSoft systems.

PeopleSoft Integration Broker enables you to perform these integrations among internal systems and third-party integration partners, while managing data structure, data format and transport disparities. Because of its modular design, you can reuse many elements that you develop for integrations.

PeopleSoft Integration Broker consists of two subsystems: the integration gateway and the integration engine. The integration gateway resides on a PeopleSoft web server, and the integration engine is installed on an application server as part of the PeopleSoft application.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Web Services

PeopleSoft Integration Broker enables you to provide web services to other PeopleSoft systems and external integration partners by generating Web Services Description Language (WSDL) documents and Web Application Definition Language (WADL) document from integration metadata. PeopleSoft supports providing WSDL documents and WADL documents to the PeopleSoft WSDL repository. The system also supports providing WSDL documents to Universal Description, Discovery, and Integration (UDDI) repositories.

The system enables you to consume WSDL documents from other PeopleSoft and third-party systems, and automatically creates integration metadata based on the consumed WSDL documents for processing integrations. You can consume WSDL documents from other PeopleSoft systems, UDDI repositories, WSDL URLs, and Web Services Inspection Language (WSIL) URLs.

Integration Gateway

The integration gateway is a platform that manages the receipt and delivery of messages passed among systems through PeopleSoft Integration Broker. It supports the leading TCP/IP application protocols used in the marketplace today and provides extensible interfaces to develop new connectors for communication with legacy, enterprise resource planning, and internet-based systems.

Additional features include:

- Backward compatibility for Extensible Markup Language (XML) links and PeopleSoft Application Messaging.
- Listening connectors and target connectors that transport messages between integration participants and the integration engine.

  Note. This feature also enables you to build your own connectors to complement those delivered with PeopleSoft Integration Broker.
- Basic logging information concerning message receipt, delivery, and errors.
- Connection persistence with continuous open feeds to external systems through connectors, with full failover capabilities.
- Transport protocol and message format management so that when messages reach the integration engine, they have a PeopleSoft-compatible message format.

See Also

Chapter 2, "Understanding PeopleSoft Integration Broker," Integration Gateway Architecture, page 7

Integration Engine

The integration engine runs on the PeopleSoft application server. Rather than communicating directly with other applications, the integration engine sends and receives messages through one or more separately installed integration gateways.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The integration engine:

- Uses a modular architecture, so it can treat gateways as black boxes and communicate with them using standard connectors.
- Adapts elements of an existing integration to produce a new integration with only minor adjustments.
- Handles messages containing data in a variety of formats. Formats include the PeopleSoft rowset-based message format, and nonrowset-based message structures including , XML document object model messages, Simple Object Access Protocol (SOAP) messages, and non-XML files.
- Sends and receives messages asynchronously (like email) or synchronously (suspending activity to wait for a response).
- Applies message transmission type and routing based on specifications that you define in a PeopleSoft Pure Internet Architecture component.
- By developing and applying application engine transform programs, the application engine can transform message structure and translate data content according to specifications that you define in PeopleSoft Pure Internet Architecture components.

  You develop transform application engine programs in PeopleCode or Extensible Stylesheet Language Transformation (XSLT) code.

  These specifications can be reused for other integrations.
- Handles security features such as authentication, nonrepudiation, and cookies.

See Also

Chapter 2, "Understanding PeopleSoft Integration Broker," Integration Engine Architecture, page 11

Integration Gateway Architecture

This section discusses:

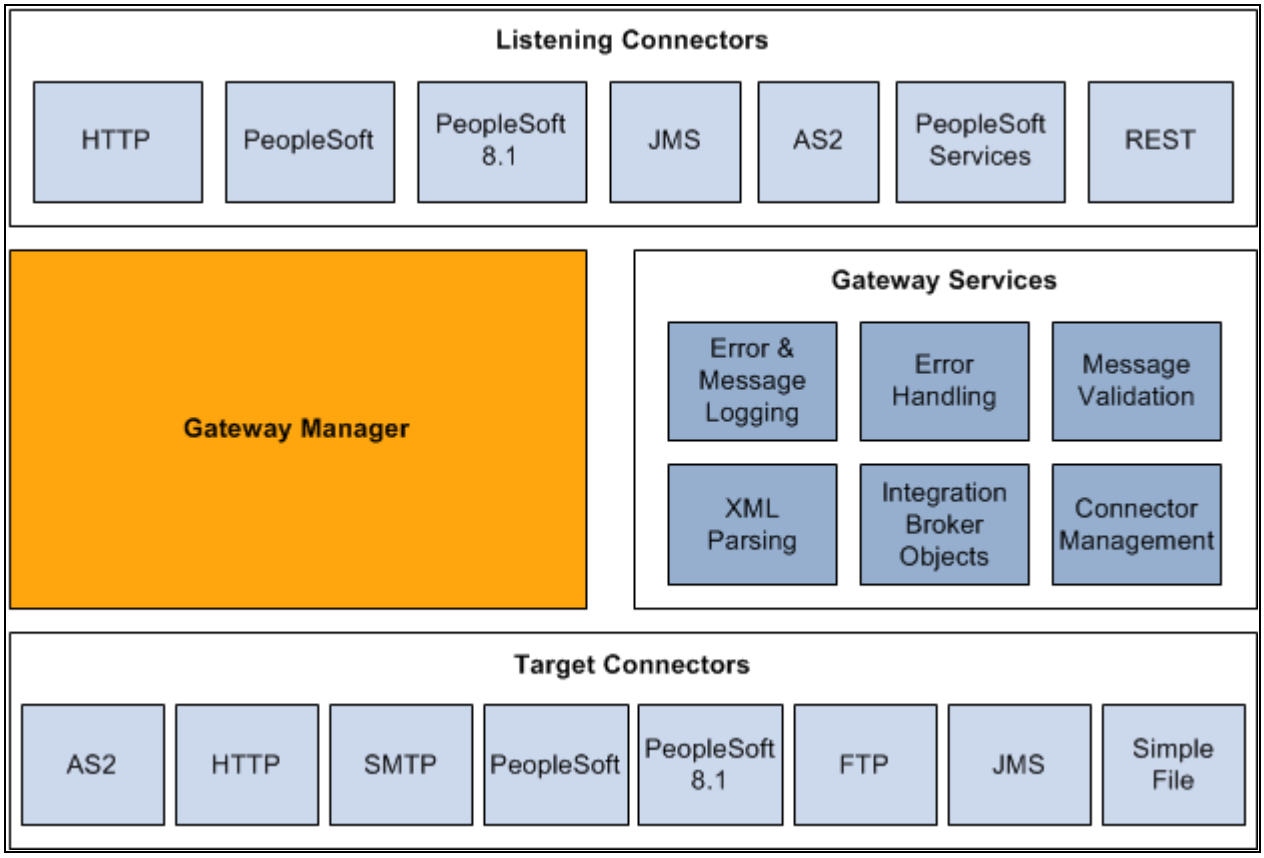
- Architecture components.
- Connectors.
- Gateway manager.
- Gateway services.

Architecture Elements

You use an integration gateway to receive and send messages among integration participant systems.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Listening connectors receive incoming messages and deliver the incoming requests to the gateway manager, which is a dispatcher for messages that flow through an integration gateway. The gateway manager determines which target connector to use to properly deliver the messages to their intended recipients. The target connector then delivers the messages to the intended recipients using the recipients' preferred protocols.



Integration gateway architecture

Connectors

Listening connectors and target connectors transport messages between integration participants and the integration gateway. These connectors support asynchronous and synchronous message handling. Many connectors are configurable at the integration gateway and system levels.

Listening Connectors

Listening connectors receive incoming data streams and perform services based on the content of the stream. They are invoked externally by other PeopleSoft systems and third-party systems.

Target Connectors

Target connectors initiate communication with other PeopleSoft systems or third-party systems. A target connector might not receive a response from the target system during each operation, but every transmission requires a low-level acknowledgment.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleSoft Integration Broker Connector SDK

The integration gateway provides a fully extensible model for developing new connectors built to the interface specification of the PeopleSoft Integration Broker software development kit (SDK) by PeopleSoft customers, consultants, and application developers.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Using Listening Connectors and Target Connectors"

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Using the Integration Broker Connector SDK"

Gateway Manager

The gateway manager processes every message that flows through an integration gateway and maintains links to the other major integration gateway components, including target connectors, listening connectors, and each of the gateway services.

Listening connectors invoke the gateway manager when they receive a request. The gateway manager uses the messaging objects IBRequest and IBResponse to determine how to route each request.

The gateway manager uses a number of the gateway services during this stage to perform operations such as message validation. The gateway manager then invokes the appropriate target connector based on the content of the message object and waits for a reply from the target connector. When the reply is received, the gateway manager forwards the reply to the calling listening connector.

If an error occurs, the gateway manager uses the error handling service and works with the service to prepare an error reply for the listening connector.

Gateway Services

This section describes the gateway services that the gateway manager uses.

XML Parsing

Most IBRequest objects and IBResponse objects that are processed in the system contain a content section that represents the actual business content sent.

Most of the time, these content sections contain XML data. Consequently, often connectors must parse and traverse XML. The standard Java XML objects are cumbersome for manipulating XML, so the integration gateway includes an XML parsing service consisting of objects that provide an intuitive interface for manipulating XML objects. This service is delivered as a set of three classes: XmlDocument, XmlNode and XmlNodeList.

See *PeopleTools 8.52: PeopleCode API Reference PeopleBook*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Integration Broker Objects

Two objects comprise the messaging objects service in the integration gateway:

- IBRequest
- IBResponse

These objects represent the request and response that enter and exit PeopleSoft Integration Broker.

See Chapter 5, "Understanding Supported Message Structures," page 39.

Connector Management

The connector management service is a composite of several services that manage connectors. The gateway processes each IBRequest to determine the appropriate connector to call in each situation. This is primarily a message routing function that has varying levels of complexity abstracted from the connectors. The connector management service also processes the IBResponse returned by each connector.

WS-Security

WS-Security is an extension to the concept of the SOAP envelope header that enables applications to construct secure SOAP message exchanges. It also provides a means for associating security tokens with messages.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Implementing Web Services Security.

Error and Service Operation Logging

Most components in the system use a standard error logging interface.

Each PeopleSoft-delivered connector uses the logging API in the same fashion, ensuring that an administrator can quickly drill down on problems or simply review the logs to see the IBRequest object, the IBResponse object, and even the raw data exchanged with integration participants.

See Chapter 19, "Managing Error Handling, Logging, Tracing, and Debugging," page 467.

Error Handling

The integration gateway provides a standard error handling interface that is exposed to each connector. This service provides error handling and error logging for most connectors delivered with PeopleSoft Integration Broker.

Message Validation

Messages that pass into PeopleSoft Integration Broker must contain certain elements to be processed. Because the integration gateway is the first component that processes messages sent to a PeopleSoft application, it performs basic validation—such as making sure that the message identifies its requestor and service operation name—to ensure that the integration engine and the target application can process them.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

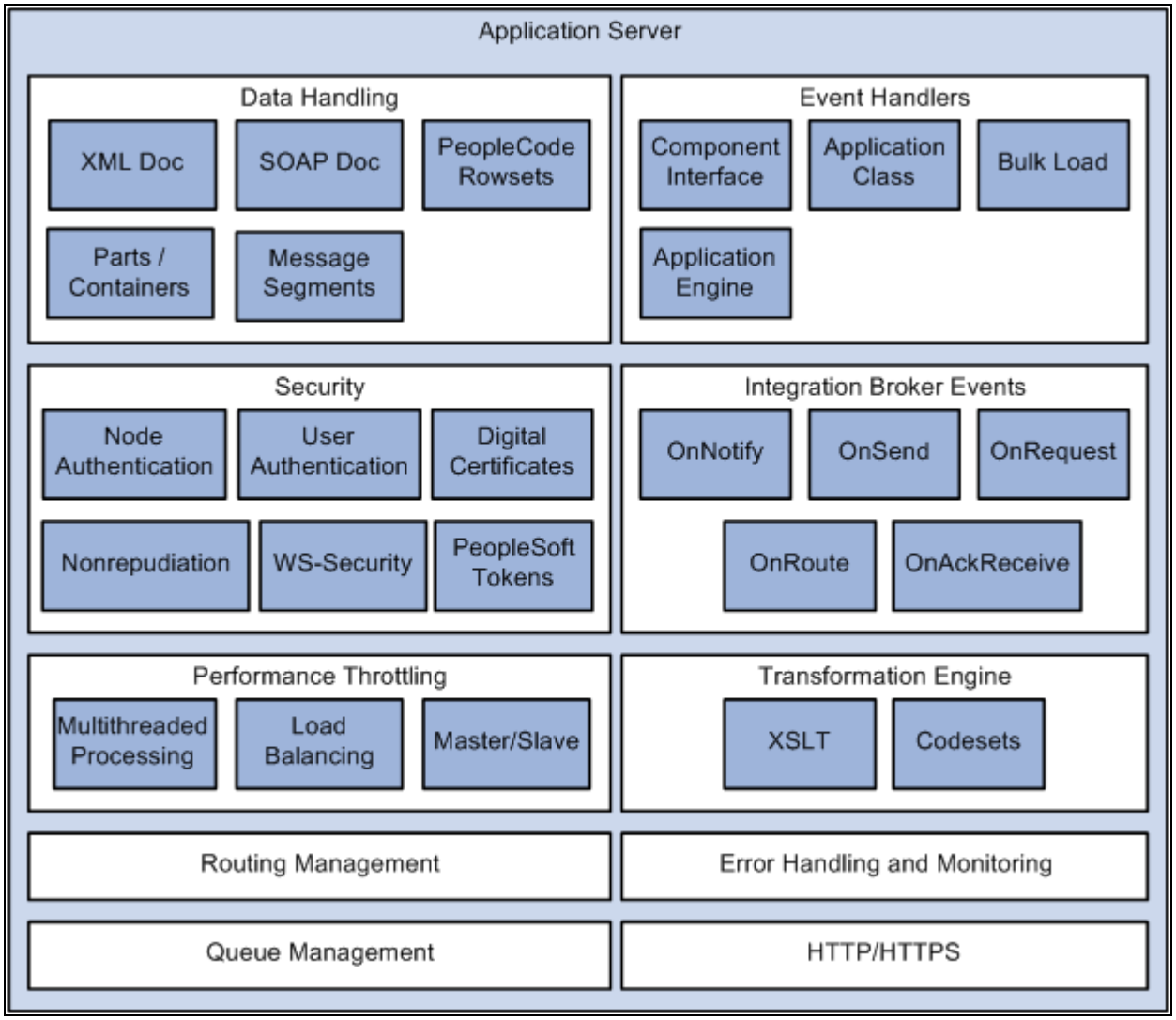
Integration Engine Architecture

The integration engine uses a variety of PeopleTools elements to create, implement, manage, and enhance integrations. Its modular architecture separates integration development activities from administrative activities.

The integration engine is a combination of PeopleSoft Application Designer definitions, PeopleSoft Pure Internet Architecture definitions, PeopleCode, and XSLT code, along with the underlying mechanisms that tie all these elements together. The underlying mechanisms include the request handlers that process both inbound and outbound messages according to the specifications in the development and administrative elements.

The integration engine resides on the PeopleSoft application server.

The following diagram shows the integration components that reside on the integration engine and the types of processing it performs.



Integration engine architecture

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operations

A service operation in the PeopleSoft system contains the processing logic for an integration and determines if the integration is to be processed synchronously or asynchronously. A service operation definition contains the following definitions:

- *Message.* A message contains the payload of the integration.
- *XML message schema.* Message schemas provide the physical description of the data that is being sent, including descriptions of fields, field types, field lengths, and so on.
- *Handler.* A service operation handler contains the processing logic for the service operation.
- *Routing.* A routing definition specifies the direction of the integration (inbound or outbound), routing alias names, transformations, and more.

Service Operation Types

PeopleSoft Integration Broker supports four types of service operations:

- Asynchronous one-way.
- Asynchronous request/response.
- Asynchronous to synchronous.
- Synchronous

Note. In this section the term *transaction* is used to describe the exchange of data between integration partners.

When PeopleSoft Integration Broker sends a service operation, the receiving system returns a response back to the sender. With asynchronous transactions, the response is automatically generated by the integration gateway, and it serves only to notify the sending system of the transmission status of the request . It is processed automatically by the application server, which uses that status information to update the Service Operations Monitor. With synchronous transactions, however, the response includes the content that is requested by the sending system, and it must be generated and returned by the receiving system.

Operation Types

PeopleSoft Integration Broker supports the operation types listed in the table.

For any operation type, the application must invoke PeopleCode, a component interface or data mover script to generate and send a service operation, or to receive and process a service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Operation Type	*Routing*	*Actions*
Asynchronous — One Way.	Outbound.	1. The application generates and sends a request. 2. One or more target system receives and processes the request.
Asynchronous — Request/Response.	Outbound.	1. The application generates and sends a request. 2. The target system receives and processes the request. 3. Sometime later the target system sends a response which contains the transaction ID from the original request. This ID serves as the correlation ID. 4. The application processes the response using the correlation ID to map it back to the original request. The message sent back is a response in the form of a request.
Asynchronous to Synchronous.	Outbound.	1. The application generates and sends a request. 2. A single target system receives and processes the request, then generates and sends a response. 3. The application receives and processes the response.
Synchronous.	Outbound.	1. The application generates and sends a request. 2. The application suspends activity and waits for a response. 3. A single target system receives and processes the request, then generates and sends a response. 4. The application resumes its activity and receives and processes the response.
Asynchronous — One way.	Inbound.	1. A source system generates and sends a request. 2. The application receives and processes the request.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Operation Type	*Routing*	*Actions*
Asynchronous — Request/Response.	Inbound.	1. A source system generates and sends a request. 2. The application receives and processes the request. 3. Sometime later the application sends a response back to the source system. The response includes a unique identifier from the original request, which serves as a correlation ID. 4. The source system processes the response using the correlation ID to map it back to the original request.
Asynchronous to Synchronous.	Inbound.	1. A source system generates and sends a request. 2. The application receives and processes the request, then generates and sends a response. 3. The source system receives and processes the response.
Synchronous.	Inbound.	1. A source system generates and sends a request. 2. The source system suspends activity and waits for a response. 3. The application receives and processes the request, then generates and sends a response. 4. The source system resumes its activity and receives and processes the response.

See Also

Chapter 10, "Managing Service Operations," Services Operation Types, page 248

Inbound and Outbound Request Flows

This section discusses how inbound and outbound service operation flow through the architecture components of PeopleSoft Integration Broker.

The PeopleSoft messaging architecture is discussed in greater detail in the Understanding Messaging chapter of this PeopleBook.

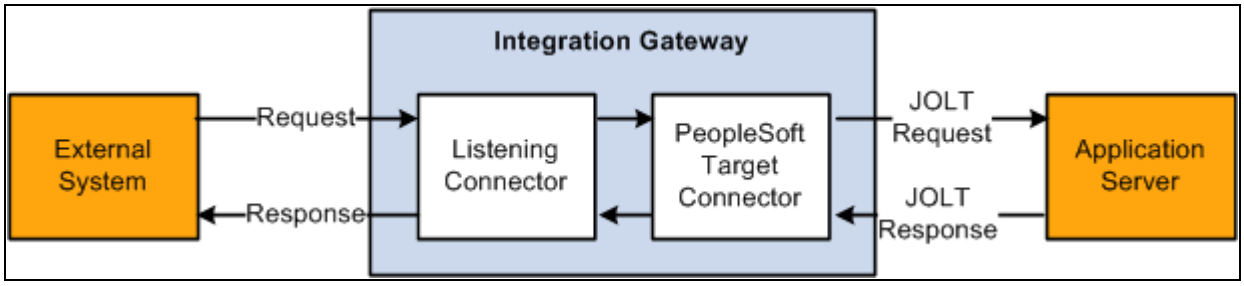
 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 3, "Understanding Messaging," page 21

Inbound Request Flow

This section describes the flow of a typical inbound request from an external system through PeopleSoft Integration Broker.



Flow of an inbound request through PeopleSoft Integration Broker

After the incoming request has been received by the integration gateway, the flow through PeopleSoft Integration Broker is the same, regardless of the listening connector used. With this in mind, no specific listening connector will be discussed here. The scenario is simple: a request is sent into the gateway, which then passes it on to the application server. The application server processes the request, and returns a response.

Step 1: External System Sends a Request to PeopleSoft Integration Broker

The first step is that an external system sends a request to PeopleSoft Integration Broker. The external system can be another PeopleSoft system or a third-party system.

Step 2: Request is Received by the Listening Connector

When a request is received by a listening connector, the first thing that the connector does is write the request to the gateway log file. (The gateway's integration properties file is used to set the logging level, which controls what is actually written to the log. If messages are not being seen in the log file, check to ensure that the log level is set correctly.) The request is written exactly as it is received. This is very useful in that it presents exactly what was sent on the wire, before the connector normalizes the service operation for use by the application server.

The connector then attempts to populate an internal request class with the particulars from the received request.

A term often used in conjunction with listening connectors is *credentials.* Incoming requests are thought to have two logical parts: the credentials and the body. The credentials can be thought of as the information required by PeopleSoft Integration Broker to process and deliver the payload of the message. The payload is located in the body. Since the credentials are separate from the body, the integration gateway does not need to parse or otherwise examine the request body for information on how to route it.

A request without credentials cannot be processed. If the integration gateway receives such a request an error will occur and an error message will be returned to the requestor.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step 3: Request is Processed by the PeopleSoft Target Connector

In order for a request to be sent from the gateway to the application server, it must pass through the PeopleSoft target connector. This connector has two major responsibilities: it serializes the request to a string, and sends that string via a JOLT connection to the application server.

All communication between the gateway and the application server is done via the use of Multipurpose Internet Mail Extensions (MIME) messages. When the request is received by the connector, it builds a MIME message. Typically the MIME message will only have two sections. In the first, the credentials are stored in an XML document in a specific format. The second section stores the body.

At this point the request is in a standard format understood by both the gateway and the application server. All requests must eventually resolve to this format before they can be sent to the application server for processing. This format effectively isolates the application server from the protocols supported by the gateway; for the most part, there is no information present about what listening connector was initially invoked by the external request.

One credential element that may be present is the one for cookies. Obviously if this is set, the application server would be right in assuming that the request came through the HTTP listening connector. However, as a general rule the application server is isolated from the details of the protocol and the general broker code on the server does not care what listening connector was used for a given request.

Once the MIME message has been built, it is written to the gateway log.

Finally, the connector looks up the JOLT connection properties from the integration properties file and attempts to send the MIME to the application server. If these properties are not set up correctly, the gateway will be unable to send requests. This is a common source of error, so care should be taken when configuring this file.

An important point to keep in mind is that even though the MIME request to the application server may appear in the gateway log file, the actual request may not have made it to the application server, since the log entry is written before the service operation is sent. If a communication error occurs, the entry will still be present in the log file. However, if this situation occurs an exception will be thrown and an error log entry will also be created.

Step 4: Request is Received by the Application Server

When the MIME request is received by the application server, the system parses it into a request object. The MIME structure is not propagated into the server.

Assuming the request parses without error, the application server pre-processes it.

Pre-processing involves:

- Authenticating the service operation, depending on the authentication scheme configured. If the request fails authentication, an error is returned.
- Determining the direction of the service operation, by looking at the external alias on the routing definition that is associated with the service operation.
- Determining the runtime handler to invoke. Currently, there are three handler types supported by the integration broker: Ping, Synchronous, and Asynchronous. The service operation type determines the handler code to invoke. Synchronous service operations are passed to sync-specific code, and asynchronous service operations are passed to the publish/subscribe subsystem.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Once a request has been passed to its respective handler, further processing is dictated by the data and PeopleCode specific to a particular system. Or in the case of hub configurations, the request may immediately be routed to another external system.

Step 5: Response is Returned by the Application Server

Regardless of how the request is processed, a response must be returned by the application server to the gateway in the same thread of execution. The connection between the gateway and the application server is essentially synchronous, independent of the type of the service operation type. When the gateway sends a request to the application server, it expects and must get a response.

In the case of synchronous processing, the generation of the response is blocked by the processing of the request. A response cannot be generated until the service operation runs to completion. There may be a noticeable delay in receiving the response, depending on the processing required by the OnRequest method or if the request is being sent out of the broker to an external system for additional processing.

Asynchronous requests behave differently. Unlike synchronous requests, there is no blocking. A response is generated for an asynchronous request as soon as the request is placed on the publication queue. Because of this, a response generated for an asynchronous request is not a response in the strictest sense of the term. Such responses should really be considered acknowledgments that the pub/sub system has received the request. Receipt of such a response is not a guarantee that any applicable notification PeopleCode has been successfully run.

Responses are converted to the MIME standard by the application server, and are returned to the gateway.

Step 6: Response is Received by the PeopleSoft Target Connector

As soon as the MIME response is received by the PeopleSoft target connector, it is written to the gateway log file.

The MIME response is then parsed back into a gateway request object, and is then returned to the listening connector.

Step 7: Response is Received by the Listening Connector

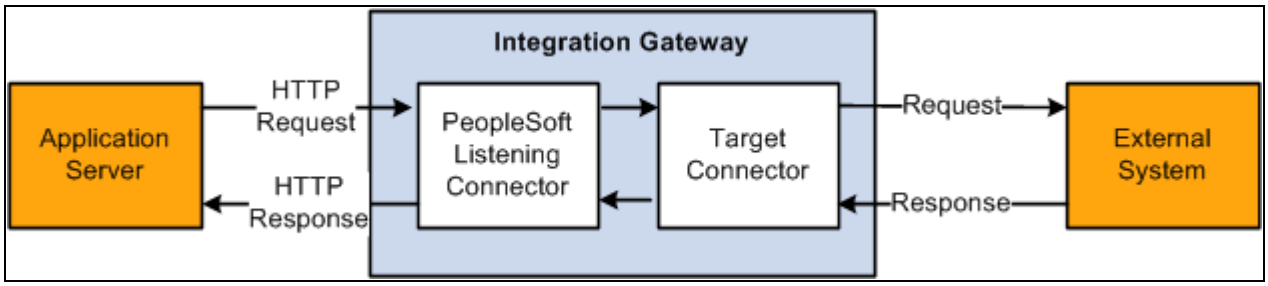
The response object is returned to the listening connector, upon which the response is mapped to a response suitable for the given protocol.

It should be emphasized that, from the viewpoint of the listening connector, the processing of requests is done synchronously. A request is received by a listening connector which then coverts it to a suitable format, makes a blocking call to the gateway to handle the message, and ultimately gets a response back all in the same thread of execution.

Outbound Request Flow

The following diagram shows an outgoing request through PeopleSoft Integration Broker.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Outgoing request through PeopleSoft Integration Broker to an external system

There are several scenarios that might result in a request being sent out of the broker. Requests can be sent in PeopleCode by using the Publish or SyncRequest methods of the Integration Broker class.

Regardless of how the request is created, the mechanism for sending it out of the broker is the same, and the flow is the same regardless of the specific outgoing target connector you invoke.

Step 1: Application Server Generates Request

Once an outgoing request has been generated, the application server must perform some basic processing before it can be sent out.

The application server looks at the request, and extracts the information about the node that it is being sent to.

If target connector information was not supplied via PeopleCode or as part of the routing, then the node name is then used to look up the name of the gateway to use, the target connector to use on that gateway, as well as any specific connector properties that need to be passed to the connector in order to handle the request. If this information is not found, an error will occur.

The application server modifies the outgoing request with the appropriate connector information.

The request is then converted to the MIME standard format, and is sent to the gateway over an HTTP connection.

The request must be sent to the PeopleSoft listening connector on the gateway. The application server uses the value of the Gateway URL defined for the given gateway. If this URL is not valid or does not point to the PeopleSoft listening connector, the application server will be unable to send the request.

Step 2: Request is Received by the PeopleSoft Listening Connector

When the MIME request is received by the PeopleSoft listening connector, it is written to the gateway log file.

The request is converted from MIME format to a gateway request object.

The connector then examines the request to determine what target connector the request is to be sent to; that target connector is then invoked.

Step 3: Request is Received by the Target Connector

The target connector validates the request. Each connector requires certain properties to be set, otherwise the request cannot be sent. For example, the HTTP target connector requires that the PrimaryURL be set. If any mandatory connector properties are missing or are invalid, an error will be thrown.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The target connector then converts the request into whatever format is required by the protocol.

The modified request is then written to the gateway log, and then sent out.

Step 4: Response is Received by the Target Connector

The response received by the target connector is written to the gateway log, and the response is used to build a gateway response object, which is then returned to the PeopleSoft listening connector.

Step 5: Response is Received by the PeopleSoft Listening Connector

The response object is then converted to the MIME standard format by the connector.

The MIME response is then written to the gateway log file, and is then returned to the application server.

Interactions with the gateway are always synchronous. If a request is sent to the gateway, a response should be expected.

Step 2 is an HTTP POST request made of the gateway, and the response created here in Step 5 is returned in response to that HTTP request. The HTTP connection is open for the duration of the processing for that request.

The response object is returned to the listening connector, upon which the response is mapped to a response suitable for the given protocol.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 3

Understanding Messaging

This chapter discusses:

- Asynchronous messaging.
- Synchronous messaging.

Note. For compatibility with previous PeopleTools releases, the PeopleSoft Integration Broker services-oriented architecture introduced in PeopleTools 8.48 overlays the messaging architecture from earlier PeopleTools 8.4x releases.

Messaging Types

PeopleSoft Integration Broker supports asynchronous and synchronous messaging.

Synchronous messaging	In synchronous messaging, a message is sent to a target system. The sending system must receive a response from the target system before it continues to process additional messages.
Asynchronous messaging	In asynchronous messaging, a message is sent to a target system. However, the sending system does not need to receive a response from the target system before it can continue processing messages. This type of messaging is also referred to as fire-and-forget messaging.

The remainder of this chapter discusses the PeopleSoft Integration Broker architecture for these messaging types.

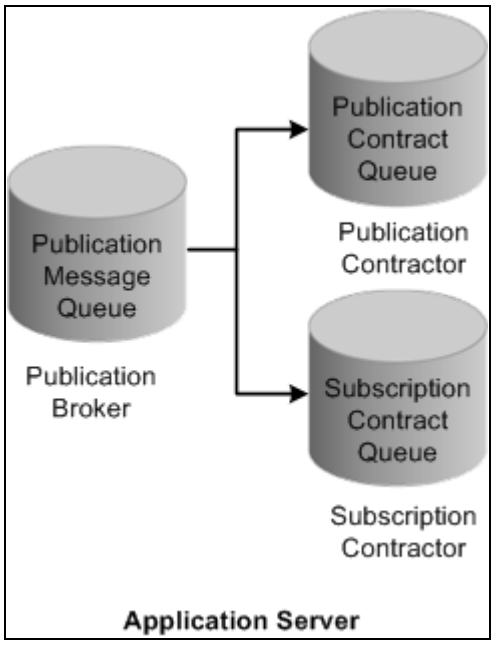
Asynchronous Messaging

This section discusses the PeopleSoft Integration Broker asynchronous messaging architecture.

Brokers, Contractors and Queues

The publication broker, publication contractor, and subscription contractor services are the primary application server elements required for asynchronous messaging. The publication broker service routes the workload to both contractor server processes, as illustrated in the following diagram:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Brokers, contractors, and queues

Publication broker	Acts as the routing mechanism. When an asynchronous service operation arrives in its queue, the publication broker service runs the defined routing rules. If the service operation needs to be published to a remote node, it routes the service operation to the publication contractor service. If the service operation is subscribed to on the local node, then the publication broker routes the service operation to the subscription contractor service. Routing involves submitting either a subscription or publication contract to the appropriate contractor, followed by an asynchronous call to the contractor service notifying it that work is waiting in the queue.
Publication contractor	References the publication contract submitted by the publication broker service and performs an HTTP post of the publication service operation to the integration gateway. When the integration gateway sends a reply indicating that it received the publication service operation, the publication contractor service updates the publication contract with the status of subscription processing *(Done* or *Retry)*.
Subscription contractor	References the subscription contract submitted by the publication broker service and runs the appropriate notification PeopleCode. Then it updates the subscription contract concerning the status of the subscription processing.

Messaging System Server Processes

The application server offers six server processes to handle asynchronous service operations. They work in pairs to provide three primary services:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service	*Server Processes*
Publication broker	• Broker dispatcher (PSBRKDSP) • Broker handler (PSBRKHND)
Publication contractor	• Publication dispatcher (PSPUBDSP) • Publication handler (PSPUBHND)
Subscription contractor	• Subscription dispatcher (PSSUBDSP) • Subscription handler (PSSUBHND)

Dispatchers and Handlers

Each of the publication broker, publication contractor, and subscription contractor is comprised of two individual server processes that work together to handle incoming requests. One server process functions as a dispatcher, while the other functions as a handler.

This relationship is analogous to the way that the application server handles workstation connections and requests. To handle the incoming client requests, the application server has a listener and a handler (or a pool of handlers). The listener receives the incoming requests and then routes them to an available handler.

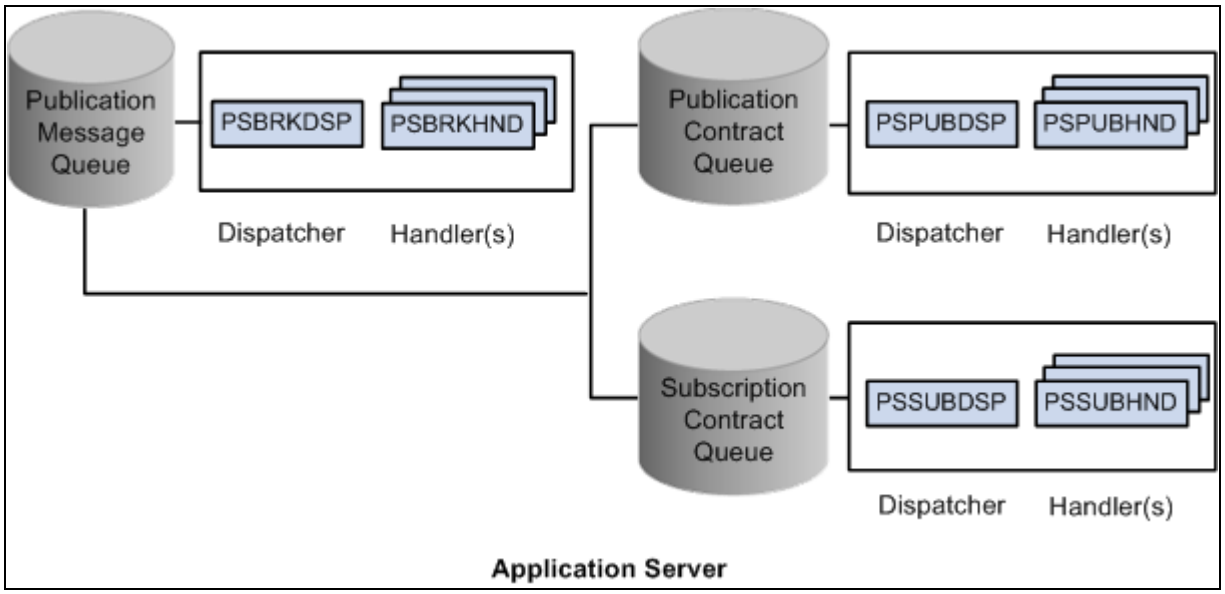
Typically, one listener serves many handlers. The relationship between the dispatcher and the handlers is analogous to the relationship between the Jolt Server Listener (JSL) and the Jolt Server handler (JSH). In the case of the application messaging server processes, the dispatcher functions as the listener, and the handler as similar to the JSH.

For the services discussed in this section (publication contractor, subscription contractor, and publication broker) there are *at least* two server processes: a single dispatcher and one or more handlers. The PS*xxx*DSP server process is the dispatcher, and the PS*xxx*HND server process is the handler.

Note. The *xxx* represents BRK, PUB, or SUB. For example, in the case of the publication broker, PSBRKDSP is the dispatcher and PSBRKHND is the handler.

This diagram shows the messaging server processes grouped by their functions in the messaging architecture:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Dispatchers and handlers

Asynchronous Service Operation Publication

This section discusses:

- Asynchronous publish of a service operation instance.
- Asynchronous publish of a publication contract.

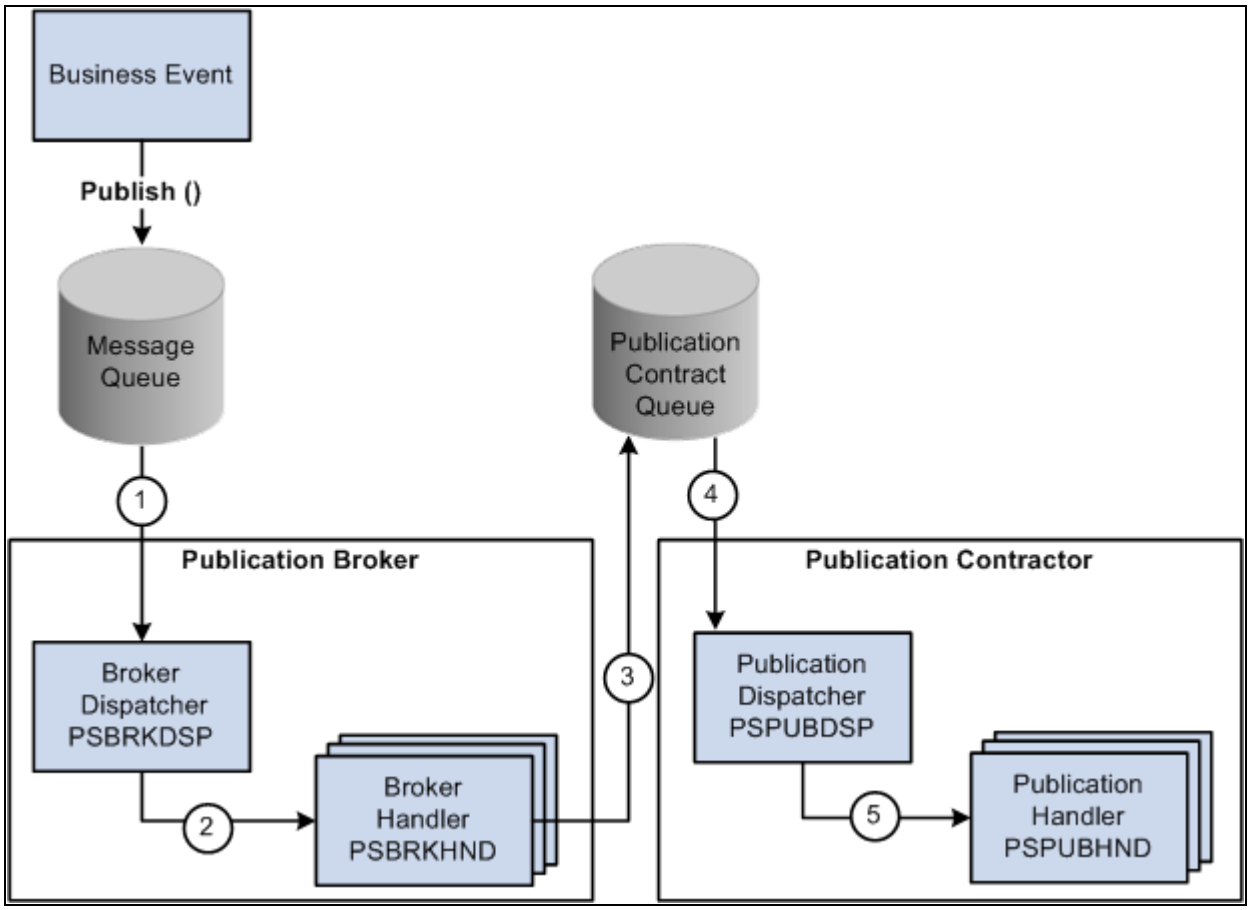
Understanding Asynchronous Service Operation Publication

This section describes the flow of an asynchronous service operation publication through PeopleSoft Integration Broker, as well as the status of the service operations as they appear in Service Operations Monitor.

Asynchronous Publish of Service Operation Instances

This diagram shows an asynchronous publish of a service operation instance in the messaging system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Asynchronous publication of an operation instance

The following table describes the processing steps of an asynchronous publication of a service operation instance in PeopleSoft Integration Broker:

Step	***Process***
1	The service operation is published and enters the message queue. The instance is written to the PSAPMSGPUBHDR table in the database, but is not yet dispatched. The broker dispatcher process picks up the service operation instance from its queue. During this stage, the service operation instance status in the Service Operations Monitor is *New*.
2	The broker dispatcher process passes the service operation instance to the broker handler process. During this stage, the service operation instance status in the Service Operations Monitor is *Started*.
3	The broker handler process accepts the service operation instance, reads the data, and runs the routing rules to determine where the publication needs to be delivered. The broker handler process then writes a publication contract in the PSAPMSGPUBCON table and notifies the publication contractor service that it has an item to process. During this stage, the service operation instance status in the Service Operation Monitor is *Working*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

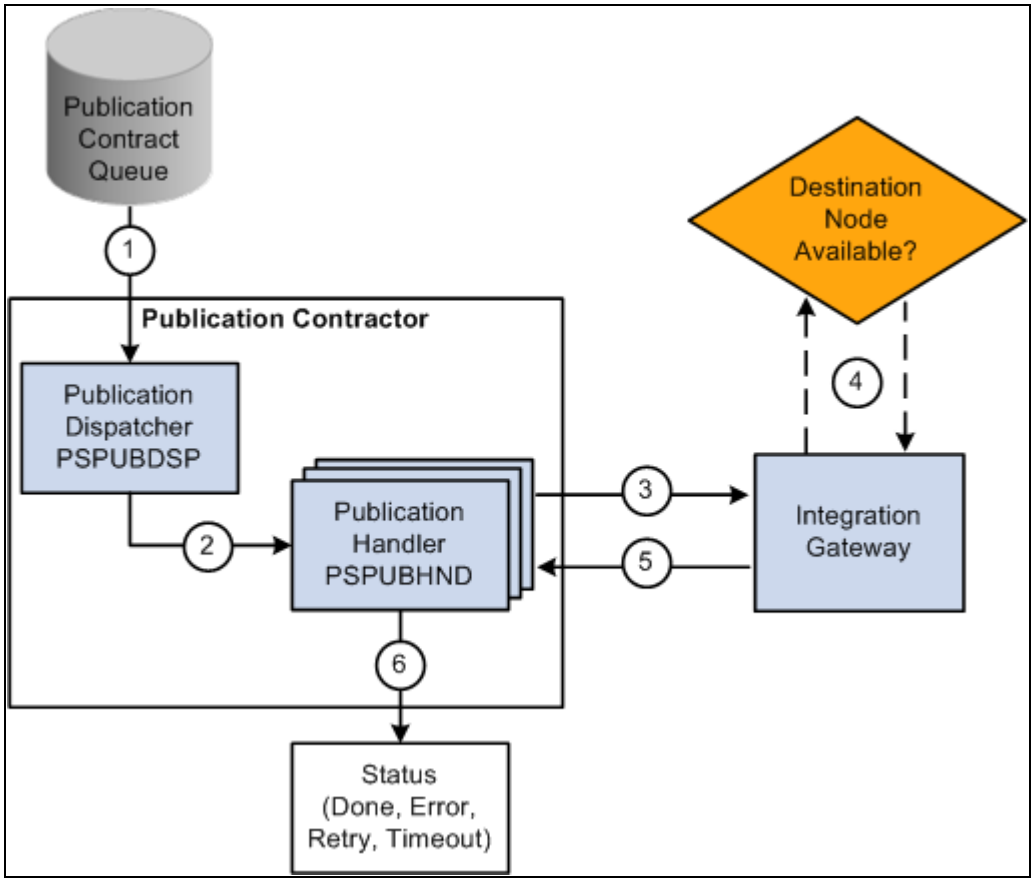
Step	Process
4	After the service operation is stored in the publication contact queue, the status of the publication contract in the Service Operations Monitor is *New*, the service operation instance status is *Done*, and the publication dispatcher process picks up the publication contract from its queue.
5	The publication dispatcher process passes the service operation instance to the publication handler process. During this stage, the publication contract status in the Service Operations Monitor is *Started.*

You view service operation instance status information on the Operation Instances page of the Service Operations Monitor. To access the page select PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Asynchronous Services, Operation Instances.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Monitoring Asynchronous Service Operation Instances.

Asynchronous Publish of Publication Contracts

This diagram shows the flow of an asynchronous publication contract through the messaging system:



Asynchronous publish of a publication contract

The following table describes the processing steps of an asynchronous publish of a publication contract in PeopleSoft Integration Broker:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step	Process
1	The publication dispatcher picks up the publication contract from the publication contract queue.
2	The publication contract is written to the PSAPMSGPUBCON table in the database, but is not yet dispatched. The publication dispatcher process passes the publication contract to the publication handler process. At this stage the status of the publication contract in the Service Operation Monitor is *Started.*
3	The publication handler process accepts the publication contract and attempts to deliver the service operation to the integration gateway. At this stage, the status of the publication contract in the Service Operations Monitor is *Working.*
4	The integration gateway attempts to pass the publication contract to the destination node.
5	The integration gateway passes the status of the publication contract back to the publication handler.
6	The publication handler updates the Service Operations Monitor with the status of the publication contract. The typical statuses that displays in the Service Operations Monitor are: • *Done.* The subscribing node successfully received the contract. • *Timeout.* The system timed out before the transaction processing was completed. • *Retry.* The system encountered and error. The retry is automatic. When service operations have *Retry* status, the service operations are not resent until an internal ping is successful. This ping is similar to a node ping. The publication Contract dispatcher, as part of its on idle processing, pings a node that is in *Retry* status and verifies if the connection is reestablished. When the ping is successful the publication Contract dispatcher resends the service operation. The service operation goes back to the publication handler process and returns to *Working* status.

You can view the status information for the publication contract using the publication Contracts page in the Service Operations Monitor. To access the page, select PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Asynchronous Services, Publication Contracts.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Monitoring Publication Contracts.

The Service Operations Monitor may display statuses for publication contracts other than those discussed in this section.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Asynchronous Service Operation Statuses.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Asynchronous Service Operation Subscription

This section discusses:

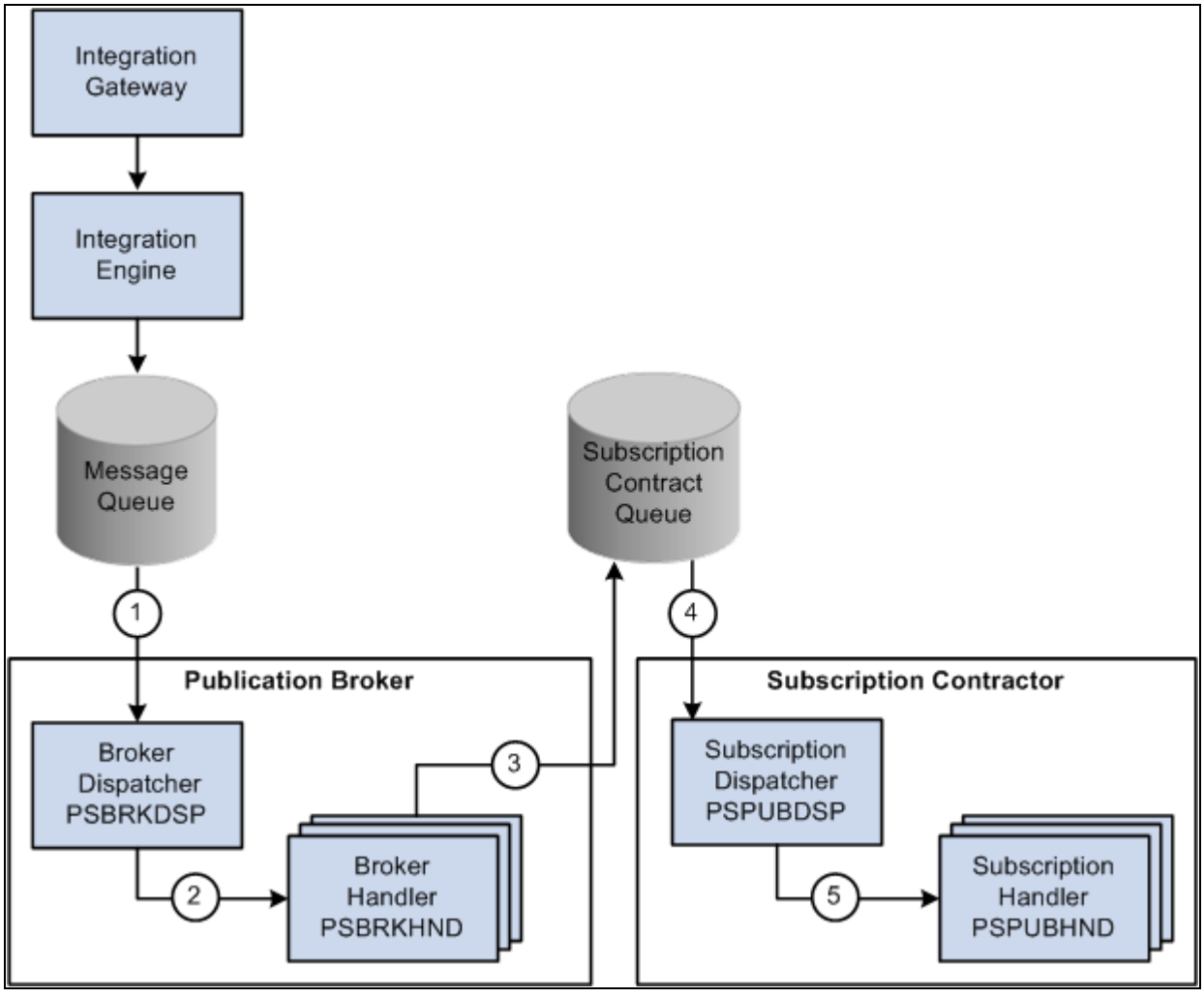
- Asynchronous subscription of a service operation instance.
- Asynchronous subscription contracts.

Understanding Asynchronous Service Operation Subscription

This section describes the flow of an asynchronous service operation subscription through PeopleSoft Integration Broker, as well as the service operation status at each stage of the process.

Asynchronous Subscription of Service Operation Instances

This diagram illustrates the flow of an asynchronous service operation subscription through PeopleSoft Integration Broker:



Asynchronous subscription of a service operation instance

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following table describes the processing steps of an asynchronous subscription of a service operation instance in PeopleSoft Integration Broker:

Step	*Process*
1	The service operation enters the message queue. The instance is written to the database, but not yet dispatched The broker dispatcher process picks up the service operation instance from its queue. During this stage, the status of the service operation instance in the Service Operations Monitor is *New*.
2	The broker dispatcher process passes the service operation instance to the broker handler process. During this stage, the status of the service operation instance in the Service Operations Monitor is *Started.*
3	The broker handler process accepts the service operation instance, reads the data, and runs the subscription routing rules to determine if the service operation needs to be processed locally. During this stage, the status of the service operation instance in the Service Operations Monitor is *Working.*
4	The broker handler process then writes a subscription contract in the PSAPMSGPUBCON table (the subscription contract queue) and notifies the subscription contractor service that it has an item to process. During this stage, the status of the service operation instance in the Service Operations Monitor is *Working.*
5	Once the service operation is stored in the subscription contact queue, the status of the service operation instance in the Service Operations Monitor is *Done*. Processing of the subscription contract begins as the subscription dispatcher process picks up the subscription contract from its queue, and the status of the subscription contract in the Service Operations Monitor is *New*. In this example, at the point when the status of the asynchronous service operation instance is *Done*, the subscription contract status is *New*. Asynchronous subscription contract processing is described in the next section.

You can view service operation instance status on the Operation Instances page of the Service Operations Monitor. To access this page, select PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Asynchronous Services, Operation Instances.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Monitoring Asynchronous Service Operation Instances.

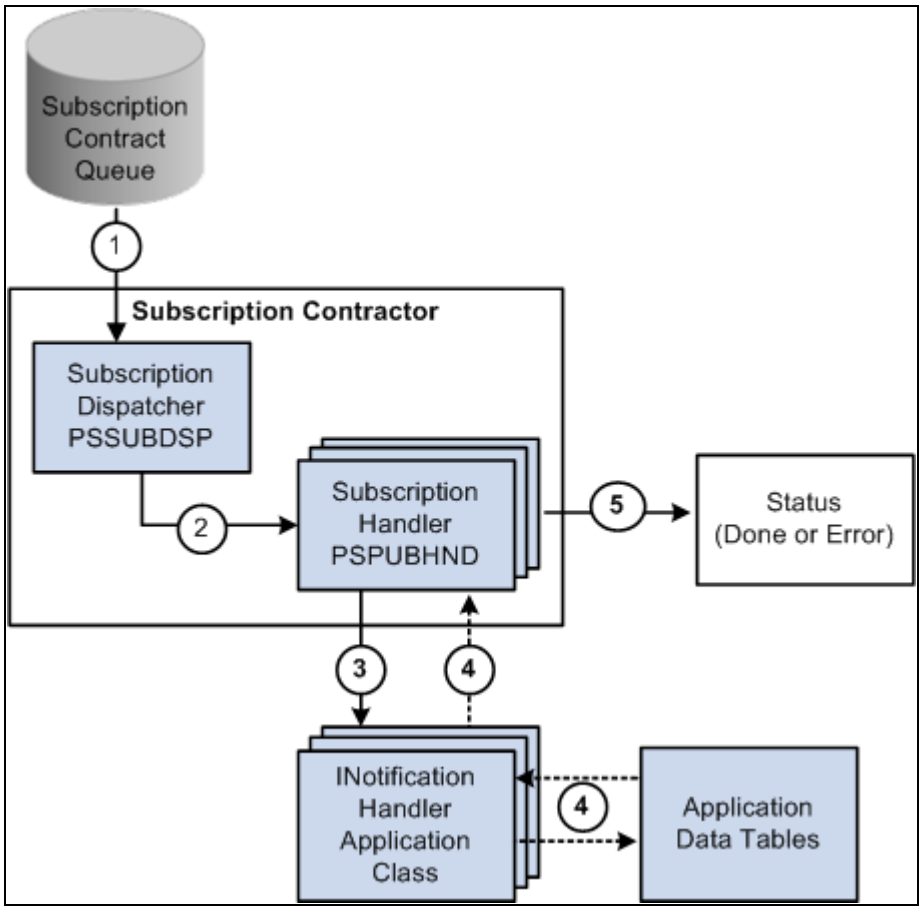
The Service Operations Monitor may display statuses for subscription instances other than those discussed in this section.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Asynchronous Service Operation Statuses.

Asynchronous Subscription Contract

This diagram shows the flow of an asynchronous subscription contract:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Asynchronous subscription contract

The following table describes the processing steps of an asynchronous subscription contract in PeopleSoft Integration Broker:

Step	***Process***
1	The subscription dispatchers picks up the contract from the subscription contract queue.
2	The subscription dispatcher process passes the subscription contract to the subscription handler process. At this stage the status of the subscription contract in the Service Operations Monitor is *Started.*
3	The subscription handler process accepts the subscription contract and runs the notification PeopleCode.
4	In the example shown in the diagram, the notification PeopleCode then uses the service operation data to update application data tables. However, the notification PeopleCode can use the service operation data as input to look up information, create and publish another service operation, and so forth. At this stage, the status of the publication contract in the Service Operations Monitor is *Working.*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step	Process
5	The subscription handler passes the status of the subscription contract to the Service Operations Monitor. The typical statuses that display in the Service Operations Monitor for an asynchronous subscription contract are: • *Done*. The notification PeopleCode ran successfully. • *Error*. An error occurred.

To view status information for subscription contracts, use the Subscription Contracts page in the Services Operation Monitor. To access the page select PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Asynchronous Services, Subscription Contracts.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Monitoring Subscription Contracts.

The Service Operations Monitor may display statuses for subscription contracts other than those discussed in this section.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations," Asynchronous Service Operation Statuses.

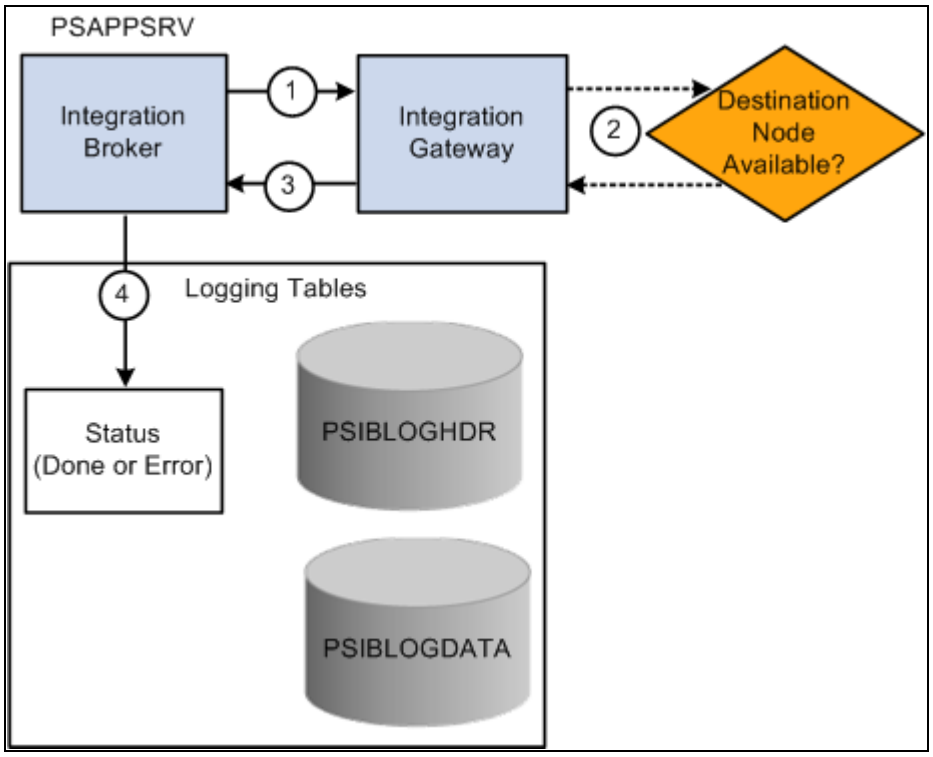
Synchronous Messaging

This section discusses synchronous messaging in PeopleSoft Integration Broker.

Synchronous Service Operation Publication

This diagram illustrates using PeopleSoft Integration Broker to consume a synchronous service operation:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Synchronous service operation publication

The following table describes the processing steps for a synchronous publication of a service operation in PeopleSoft Integration Broker:

Step	***Process***
1	The integration engine sends the service operation to the integration gateway.
2	The integration gateway attempts to deliver the service operation to the destination node.
3	The integration gateway sends back the status information to the integration engine
4	The integration engine updates the database tables as well as sends the status information to the Service Operations Monitor. The possible statuses in the Service Operations Monitor for a synchronous publication are: • *Done.* The integration gateway was able to deliver the service operation to the destination node. • *Error.* The integration gateway was not able to deliver the service operation to the destination node.

You can view the status information for the invocation in the Service Operations Monitor using the Synchronous Services page. To access the page select PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Synchronous Services.

For status information for synchronous integrations to be available in the Service Operations Monitor, you must set the Log Detail parameter in the routing definition for the service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

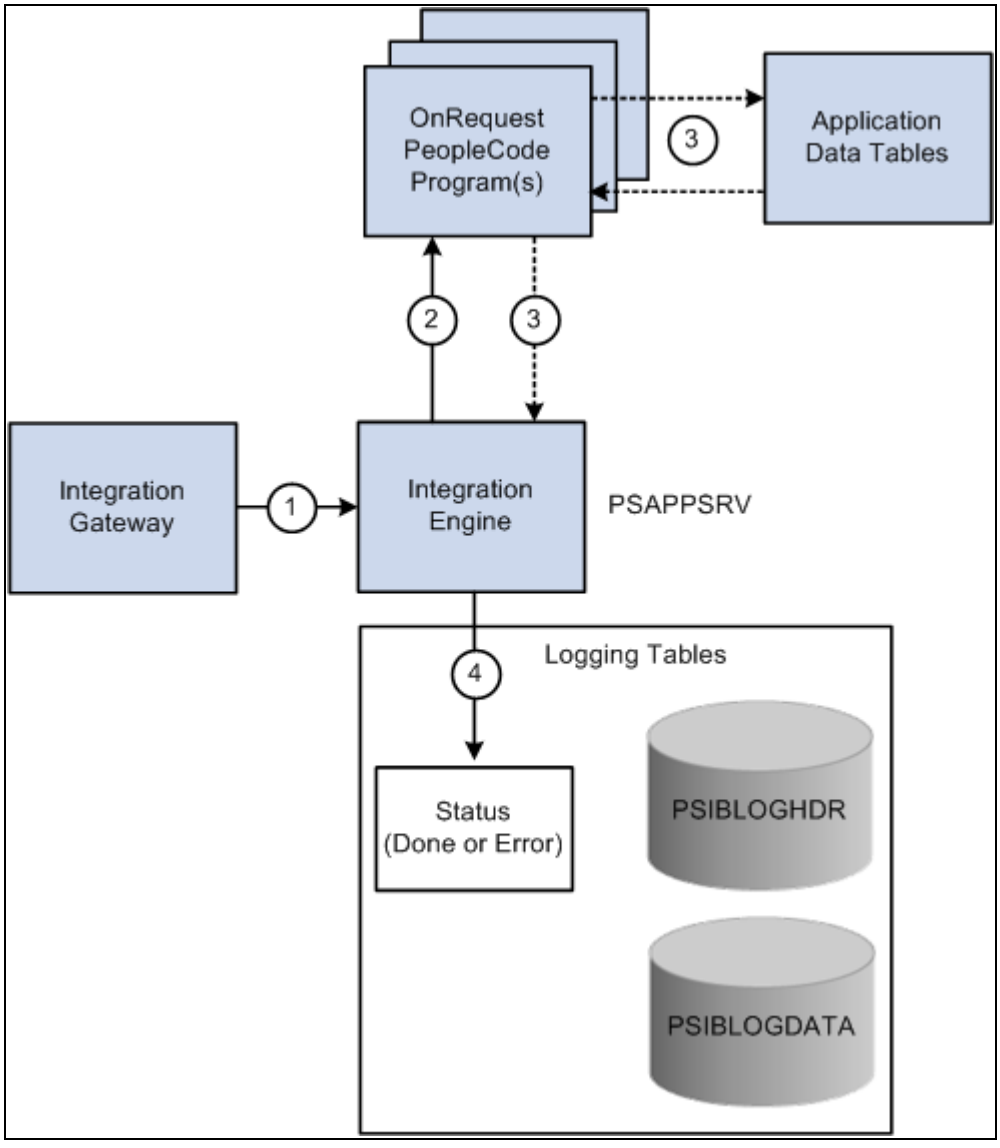
See Also

Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Monitoring Synchronous Service Operations"

Synchronous Service Operation Subscription

This diagram illustrates providing a synchronous service operation through PeopleSoft Integration Broker:



Synchronous service operation subscription

The following table describes the processing steps of a synchronous service operation subscription in PeopleSoft Integration Broker:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step	Process
1	The integration gateway passes an inbound synchronous service operation to the integration engine.
2	The integration engine runs an OnRequest PeopleCode event program.
3	The OnRequest PeopleCode program attempts to update the application data tables.
4	The integration engine updates the database tables as well as sends the status information to the Service Operations Monitor. The possible statuses in the Service Operations Monitor for a synchronous publication are: • *Done.* The integration gateway was able to deliver the service operation to the destination node. • *Error.* The integration gateway was not able to deliver the service operation to the destination node.

For status information for synchronous integrations to be available in the Service Operations Monitor, you must set the Log Detail parameter in the routing definition for the service operation.

You can view the status information for the publication in the Service Operations Monitor by using the Synchronous Services page. Access this page by selecting PeopleTools, Integration Broker, Service Operations Monitor, Monitor, Synchronous Services.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Monitoring Synchronous Service Operations"

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 4

Understanding PeopleSoft Integration Broker Metadata

This chapter provides a high-level overview of the integration metadata that you need to create to use PeopleSoft Integration Broker. This chapter discusses:

- Integration metadata
- Order of precedence for creating integration metadata

PeopleSoft Integration Broker Metadata

You use the following integration metadata to create and implement integrations using PeopleSoft Integration Broker

Integration PeopleCode	You use integration PeopleCode to send and receive messages, route messages and manipulate message content.
Integration gateway definitions	This definition is an application's internal representation of an installed integration gateway. An application requires at least the local gateway, through which it can send and receive messages. Multiple nodes can share the same local gateway, which might be the only gateway that you need for all of the integrations.
Message definitions	Message definitions provide the physical description of the data that is being sent, including fields, field types, and field lengths.
Node definitions	Nodes represent any organization, application or system that will play a part in integrations. For example, nodes can represent customers, business units, suppliers, other trading partners, external or third-party software systems, and so on. Node definitions define the locations to or from which messages can be routed. Because an application can send messages to itself, a *default local node* definition that represents the application is delivered as part of the integration engine. Each PeopleSoft installation must have one, and only one, default local node
Queue definitions	Queues group asynchronous services for processing. In addition, they can dictate the order of processing of the asynchronous service operations .

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Routing definitions	Routing definitions determine the sender and receiver of an integration. Routing definitions allow you to specify inbound and outbound transformations that enable you to transform data structures into those that the sending or receiving systems can understand.
Service definitions	Service definitions group service operations into logical groups or categories.
Service operation definitions	Service operations define the processing logic of an integration. They specify the inbound, outbound and fault messages associated with an integration, the integration PeopleCode to invoke, and the routing to use.
Transformation programs	A transformation or transform program is a type of Application Engine program that you develop and specify as part of a routing definition. PeopleSoft Integration Broker supports the use of Extensible Stylesheet Language Transformation (XSLT) code and PeopleCode for developing transform programs. Transform programs can transform, filter and translate data.

Order of Precedence for Creating Integration Metadata

Create integration metadata in the following order:

1. Integration gateway definition.
2. Node definition.
3. Message definition.
4. Integration PeopleCode.
5. Transformation programs.
6. Queue definition.
7. Service definition.
8. Service operation definition.
9. Handler definition.
10. Routing definition.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Defining Integration Gateways

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Chapter 6, "Managing Messages," page 75

Chapter 18, "Applying Filtering, Transformation and Translation," page 415

Chapter 13, "Managing Service Operation Queues," page 321

Chapter 7, "Sending and Receiving Messages," page 127

Chapter 9, "Managing Services," page 223

Chapter 10, "Managing Service Operations," page 247

Chapter 17, "Managing Service Operation Routing Definitions," page 371

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 5

Understanding Supported Message Structures

This chapter discusses:

- Internal message format for request messages.
- Internal message format for response messages.
- Accessing IBInfo elements using PeopleCode.
- Rowset-based message structure.
- PSCAMA.
- Identifying changes to field-level attributes.
- Nonrowset-based message structures.
- XML DOM-Compliant messages.
- SOAP-Compliant messages.
- Non-XML messages.
- Message part structures.
- Message container structures.

Supported Message Structures

This chapter discusses the message structures used by PeopleSoft Integration Broker to exchange request and response messages between the integration gateway and the application server, between other PeopleSoft systems, and between third-party integration partners.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Integration Broker Message Structures

This section discusses the internal message formats for request messages and response messages, local compression, and how to access IBInfo elements.

Internal Message Format for Request Messages

This section discusses the format used to exchange request messages between the integration gateway and the application server. These messages are frequently referred to as IBRequest messages.

The Multipurpose Internet Mail Extension standard (MIME) is used as the basic structure for internal messaging. MIME has several advantages in that the standard is well-known and supported, and because it is text-based, it is human readable and easily serializable.

Messages using the internal format display in the integration gateway log file. Since the log file is a valuable tool for debugging, anyone reading the file will need to understand how the messages are structured.

Every request message contains three parts:

Headers	The first part of a request message contains headers which describe the attributes of the whole message.
IBInfo (Integration Broker Information)	The IBInfo (Integration Broker Information) section contains the credentials of the request as well as all other information required by the PeopleSoft Integration Broker to process the message. The IBInfo for a request has a specific XML structure which is used for all request messages in the system, regardless if the message is being sent to the application server or to the integration gateway.
Content section	The final section contains the message body of the original request. This is the payload and is what is ultimately delivered to the final destination.

The following is an example of a request message in the PeopleSoft internal MIME format:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Message-ID: <-123.123.123.123@nowhere >
Mime-Version: 1.0
Content-Type: multipart/related; boundary="Integration_Server_MIME_Boundary"
Content-ID: PeopleSoft-Internal-Mime-Message
PeopleSoft-ToolsRelease: 8.52

--Integration_Server_MIME_Boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: IBInfo
Content-Disposition: inline

<?xml version="1.0" ?>
  <IBInfo>
   <TransactionID>
     <![CDATA[ caa3a040-bde5-11da-914c-ecaede80d83b]]>
  </TransactionID>
  <ExternalOperationName>
  <![CDATA[ QE_FLIGHTPLAN_TRANSFORM.VERSION_1]]>
  </ExternalOperationName>
  <OperationType>async</OperationType>
  <From>
    <RequestingNode>
       <![CDATA[ QE_LOCAL]]>
    </RequestingNode>
    <RequestingNodeDescription>
       <![CDATA[ ]]>
    </RequestingNodeDescription>
    <NodePassword>
       <![CDATA[ password]]>
    </NodePassword>
    <ExternalUserName>
    <![CDATA[ ]]>
    </ExternalUserName>
    <ExternalUserPassword>
    <![CDATA[ ]]>
    </ExternalUserPassword>
    <AuthToken>
       <![CDATA[ owAAAAQDAgEBAAAAvAIAAAAAAAAsAAAABABTaGRyAk4AbQg4AC4AMQ
        AwABTFZOonLEjJaPtR6v02oadvRUoSq2MAAAAFAFNkYXRhV3icHYhNDkAwGERfEQ
        srFyFN0cZSaGz8xAmcwA0dzug3yZv53gMUeWaM+s1IV11EFnZOysjBSv2bm01mZl
        L3Dqt4GrETHSHtQCs6cWBM2ybr9fMBbP0LSQ==]]>
    </AuthToken>
    <WSA-ReplyTo>
       <![CDATA[ ]]>
    </WSA-ReplyTo>
    <NodeDN>
       <![CDATA[ ]]>
    </NodeDN>
    <OrigUser>
       <![CDATA[ QEDMO]]>
    </OrigUser>
    <OrigNode>
       <![CDATA[ QE_LOCAL]]>
    </OrigNode>
    <OrigProcess>
       <![CDATA[ QE_FLIGHTDATA]]>
    </OrigProcess>
    <OrigTimeStamp>2006-03-27T15:02:39.280000-0800</OrigTimeStamp>
    <DirectGatewayRequest />
    <SyncServiceTimeout />
    <ExternalMessageID>
       <![CDATA[ ]]>
    </ExternalMessageID>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<SegmentsUnOrder>N</SegmentsUnOrder>
<ConversationID>
   <![CDATA[ ]]>
</ConversationID>
<WSA-MessageID>
   <![CDATA[ ]]>
</WSA-MessageID>
<InReplyToID>
   <![CDATA[ ]]>
</InReplyToID>
<DataChunk>
   <![CDATA[ ]]>
</DataChunk>
<DataChunkCount>
   <![CDATA[ ]]>
</DataChunkCount>
</From>
<WS-Security>
  <WSTokenType>
     <![CDATA[ ]]>
  </WSTokenType>
</WS-Security>
<To>
  <DestinationNode>
      <![CDATA[ QE_IBTGT]]>
  </DestinationNode>
  <FinalDestinationNode>
     <![CDATA[ ]]>
  </FinalDestinationNode>
  <AppServerDomain>
     <![CDATA[ ]]>
  </AppServerDomain>
</To>
<Cookies>
   <![CDATA[ ]]>
</Cookies>
<PathInfo>
   <![CDATA[ ]]>
</PathInfo>
<HttpSession>
  <SessionID>
     <![CDATA[ ]]>
  </SessionID>
</HttpSession>
<QStrArgs />
<ContentSections>
   <ContentSection>
      <ID>ContentSection0</ID>
      <NonRepudiation>N</NonRepudiation>
      <Headers>
        <version>
           <![CDATA[ VERSION_1]]>
        </version>
        <encoding>
           <![CDATA[ base64(deflate)]]>
        </encoding>
        <encodedlength>
           <![CDATA[ 948(709)]]>
        </encodedlength>
        <length>
           <![CDATA[ 2840]]>
        </length>
      </Headers>
   </ContentSection>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
     </ContentSections>
    <PublishNode>
       <![CDATA[ QE_LOCAL]]>
    </PublishNode>
    <Queue>
       <![CDATA[ QE_FLIGHTPLAN_CHNL]]>
    </Queue>
    <SubQueue>
       <![CDATA[ c9de8110-bde5-11da-be79-846b4e717ebf]]>
    </SubQueue>
    <VisitedNodes>
       <![CDATA[ QE_LOCAL|]]>
    </VisitedNodes>
    <Connector>
       <ConnectorName>
          <![CDATA[ PSFTTARGET]]>
       </ConnectorName>
       <ConnectorClassName>
          <![CDATA[ PeopleSoftTargetConnector]]>
       </ConnectorClassName>
       <RemoteFrameworkURL>
          <![CDATA[ ]]>
       </RemoteFrameworkURL>
       <ConnectorParameters>
          <ConnectorParam>
             <Name>
               <![CDATA[ URL]]>
             </Name>
             <Value>
                <![CDATA[ ]]>
             </Value>
          </ConnectorParam>
       </ConnectorParameters>
      <ConnectorHeaders />
   </Connector>
  <AttachmentSection ResponseAsAttachment="N" />
</IBInfo>

--Integration_Server_MIME_Boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: ContentSection0
Content-Disposition: inline

eJydll1P2zAUhu8n7T+g/gAKFdt6UYxOHCe1FH9gO1RwE1UMTUiMTRRN27+fk7SpvwItd/F57c
TvG+c5WVz9/fl08ufhZfP46/lycn56NrlCnz8trklTVLRcGlkBb4wCrguhGFoUjw9P382/3w8b
tJ+Ug4GT+6f1ZnM5UZNOAMxrlhF18mrnXk76wWTaaUzTRhOuxU7FS9hpopBxEbC5lSSuY6gqTU
seK/q6hlyJhIIFY+fp8iwuE8yCYk40VpXgZVCfeml0i1eSN1IRTYw+Ih1twFDBG569KUuhU/KK
gBQ8HVdR37VGeUHDzW9FIdtbx9oK1JJAnnDsWOxmcrihZbfBIxyv4FYKys2Y5YyAolHgVrDHsk
z4hMpQU+cJpYJRxb7REam1nnz8DakEpoYSHWs2iPakJCLbR7T1kIPaZnhEZv0yTSqCnXcWT2EiZ
alfLSQZ0TNQGkPiJHQLR2I3pYFU3V5yTqrWX/yq7iirzTIWpCoS2blh9esVA0b4Mcm9831BORIJ
TWy//9q0JDg8AlNnc2ghbZrMQ6TFalnbuBocflZQ59S0yAvjz0C3J3hsHdOlPQ/XFkLhUZVKYKJ
1Q7n1zjGJbMs6q6heNqquSEMTN+Y2Em69hCZ7X/bCbQts8yNfv67Rwrysnzfr+1fbWg5rFmj+bT
7ro9tV/H6B7C1UN8GpbdsGmp9eXHRaO9i3DVScz7f37IZue0Bfv1z0DxwqQ49AGXRKPxh6BMrwU
J7tegSyiRRduR3se8TAgrehiBKmXSh6GHTQ5+POR1yANQ9lIb48ZH0YUykXAaYCMKVg5AEohI4L
mhAuHlAGiHwQHCkvDjhcVAx4iJAwPfAv4KCHOX0/7PRRdw87utfFU53bp1X4K/MmvRDh5WLqFhy
CJajVz4+qLr4SyEJnFjZhLeSWzyqPTx6KpgOh9k6D/9z/1gQ1Ww==
--Integration_Server_MIME_Boundary--
```

IBRequest Header Section

The first part of a request message contains headers which describe the attributes of the whole message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Message-ID: <-123.123.123.123@nowhere >
Mime-Version: 1.0
Content-Type: multipart/related; boundary="Integration_Server_MIME_Boundary"
Content-ID: PeopleSoft-Internal-Mime-Message
PeopleSoft-ToolsRelease: 8.52

--Integration_Server_MIME_Boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: IBInfo
Content-Disposition: inline
```

IBRequest IBInfo Section

The following example shows an IBInfo section for a request message that was sent from the application server to the integration gateway (formatted for easier reading):

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0" ?>
  <IBInfo>
   <TransactionID>
     <![CDATA[ caa3a040-bde5-11da-914c-ecaede80d83b]]>
  </TransactionID>
  <ExternalOperationName>
  <![CDATA[ QE_FLIGHTPLAN_TRANSFORM.VERSION_1]]>
  </ExternalOperationName>
  <OperationType>async</OperationType>
  <From>
    <RequestingNode>
       <![CDATA[ QE_LOCAL]]>
    </RequestingNode>
    <RequestingNodeDescription>
       <![CDATA[ ]]>
    </RequestingNodeDescription>
    <NodePassword>
       <![CDATA[ password]]>
    </NodePassword>
    <ExternalUserName>
    <![CDATA[ ]]>
    </ExternalUserName>
    <ExternalUserPassword>
    <![CDATA[ ]]>
    </ExternalUserPassword>
    <AuthToken>
       <![CDATA[ owAAAAQDAgEBAAAAvAIAAAAAAAAsAAAABABTaGRyAk4AbQg4AC4AMQ
        AwABTFZOonLEjJaPtR6v02oadvRUoSq2MAAAAFAFNkYXRhV3icHYhNDkAwGERfEQ
        srFyFN0cZSaGz8xAmcwA0dzug3yZv53gMUeWaM+s1IV11EFnZOysjBSv2bm01mZl
        L3Dqt4GrETHSHtQCs6cWBM2ybr9fMBbP0LSQ==]]>
    </AuthToken>
    <WSA-ReplyTo>
       <![CDATA[ ]]>
    </WSA-ReplyTo>
    <NodeDN>
       <![CDATA[ ]]>
    </NodeDN>
    <OrigUser>
       <![CDATA[ QEDMO]]>
    </OrigUser>
    <OrigNode>
       <![CDATA[ QE_LOCAL]]>
    </OrigNode>
    <OrigProcess>
       <![CDATA[ QE_FLIGHTDATA]]>
    </OrigProcess>
    <OrigTimeStamp>2006-03-27T15:02:39.280000-0800</OrigTimeStamp>
    <DirectGatewayRequest />
    <SyncServiceTimeout />
    <ExternalMessageID>
       <![CDATA[ ]]>
    </ExternalMessageID>
    <SegmentsUnOrder>N
    </SegmentsUnOrder>
    <ConversationID>
       <![CDATA[ ]]>
    </ConversationID>
    <WSA-MessageID>
       <![CDATA[ ]]>
    </WSA-MessageID>
    <InReplyToID>
       <![CDATA[ ]]>
    </InReplyToID>
    <DataChunk>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
   <![CDATA[ ]]>
</DataChunk>
<DataChunkCount>
   <![CDATA[ ]]>
</DataChunkCount>
</From>
<WS-Security>
  <WSTokenType>
     <![CDATA[ ]]>
  </WSTokenType>
</WS-Security>
<To>
  <DestinationNode>
      <![CDATA[ QE_IBTGT]]>
  </DestinationNode>
  <FinalDestinationNode>
     <![CDATA[ ]]>
  </FinalDestinationNode>
  <AppServerDomain>
     <![CDATA[ ]]>
  </AppServerDomain>
</To>
<Cookies>
   <![CDATA[ ]]>
</Cookies>
<PathInfo>
   <![CDATA[ ]]>
</PathInfo>
<HttpSession>
  <SessionID>
     <![CDATA[ ]]>
  </SessionID>
</HttpSession>
<QStrArgs />
<ContentSections>
   <ContentSection>
      <ID>ContentSection0</ID>
      <NonRepudiation>N</NonRepudiation>
      <Headers>
        <version>
           <![CDATA[ VERSION_1]]>
        </version>
        <encoding>
           <![CDATA[ base64(deflate)]]>
        </encoding>
        <encodedlength>
           <![CDATA[ 948(709)]]>
        </encodedlength>
        <length>
           <![CDATA[ 2840]]>
        </length>
      </Headers>
   </ContentSection>
 </ContentSections>
<PublishNode>
   <![CDATA[ QE_LOCAL]]>
</PublishNode>
<Queue>
   <![CDATA[ QE_FLIGHTPLAN_CHNL]]>
</Queue>
<SubQueue>
   <![CDATA[ c9de8110-bde5-11da-be79-846b4e717ebf]]>
</SubQueue>
<VisitedNodes>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      <![CDATA[ QE_LOCAL|]]>
    </VisitedNodes>
    <Connector>
      <ConnectorName>
        <![CDATA[ PSFTTARGET]]>
      </ConnectorName>
      <ConnectorClassName>
        <![CDATA[ PeopleSoftTargetConnector]]>
      </ConnectorClassName>
      <RemoteFrameworkURL>
        <![CDATA[ ]]>
      </RemoteFrameworkURL>
      <ConnectorParameters>
        <ConnectorParam>
          <Name>
            <![CDATA[ URL]]>
          </Name>
          <Value>
            <![CDATA[ ]]>
          </Value>
        </ConnectorParam>
      </ConnectorParameters>
     <ConnectorHeaders />
  </Connector>
  <AttachmentSection ResponseAsAttachment="N" />
</IBInfo>
```

While the basic structure is the same for all requests, not all elements are always required. An example of this is the Connector section. The Connector XML is used to tell the integration gateway to route a message to the named target connector. It also lists configuration parameters for the outbound request. This XML would only be seen in requests sent from the application server to the integration gateway. For requests going in the other direction, the section would be empty.

Note. The only element that is always required is *ExternalOperationName.*

The following is a list of the most important elements that may appear in the IBInfo section:

Element	***Description***
IBInfo / ExternalOperationName	The name of the requested service operation.
IBInfo / Operation Type	(Optional.) This is the type of service operation. The valid values are: asynchronous, synchronous and ping.
IBInfo / TransactionID	(Optional.) The transaction ID is used to uniquely identify a request.
IBInfo / From / RequestingNode	(Optional.) The requesting node is the node that sent the request to the current system.
IBInfo / From / Password	(Optional.) This is the password for the requesting node.
IBInfo / From / DN	(Optional.) For incoming requests, the DN gives the Distinguished Name extracted from the certificate authentication process.
IBInfo / From / OrigNode	(Optional.) For requests that cross multiple nodes, OrigNode is used to identify the node that initiated the request.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Element	*Description*
IBInfo / From / OrigTimeStamp	(Optional.) This timestamp corresponds to the time that the request was created. For requests that cross nodes, this is the time that the first request was created.
IBInfo / To / DestinationNode	(Optional.) This is the node to which the request will be delivered.
IBInfo / To / FinalDestinationNode	(Optional.) In cases where the message will be passed across several nodes, this value specifies the ultimate target of the message.
IBInfo / QStrArgs	(Optional.) Specific to incoming HTTP requests. These are the query string parameters found when the request was received by the HTTP listening connector.
IBInfo / Cookies	(Optional.) Specific to incoming HTTP requests. This is cookie string found when the request was received by the HTTP listening connector.
IBInfo / PathInfo	(Optional.) Specific to incoming HTTP requests. This is the path information extracted from the request.
IBInfo / ContentSections / ContentSection	(Optional.) This node provides metadata about the text present in the ContentSection.
IBInfo / ContentSections / ContentSection / ID	(Optional.) The index number of the content section.
IBInfo / ContentSections / ContentSection / NonRepudiation	(Optional.) Indicates as to whether nonrepudiation should be performed.
IBInfo / ContentSections / ContentSection / Headers	(Optional.) Provides additional information about the data.
IBInfo / PublishingNode	(Optional.) The node that published the message.
IBInfo / Queue	(Optional.) The queue to which the service operation was published.
IBInfo / InternalInfo / AppMsg / SubQueue	(Optional.) The subqueue to which the service operation was published.
IBInfo / InternalInfo / AppMsg / VisitedNodes	(Optional.) The list of nodes that have already received this message. This is useful when a message is being propagated across multiple nodes.
IBInfo / InternalInfo / AppMsg / PublicationID	(Optional.) The publication ID for this message.
IBInfo / Connector	(Optional.) Connector information instructs the gateway as to how to process the request.
IBInfo / Connector / ConnectorName	(Optional.) This is the proper name of the target connector to invoke to send the message.
IBInfo / Connector / ConnectorClassName	(Optional.) This is the class name of the target connector to invoke.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Element	*Description*
IBInfo / Connector / ConnectorParameters	(Optional.) Connector parameters are processing instructions for the target connector to be invoked.
IBInfo / Connector / ConnectorHeaders	(Optional.) Connector headers provide further metadata about the contents of the message to be sent.

IBRequest Content Section

The content section of a request message features the message body.

```
--Integration_Server_MIME_Boundary
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: ContentSection0
Content-Disposition: inline
eJydll1P2zAUhu8n7T+g/gAKFdt6UYxOHCe1FH9gO1RwE1UMTUiMTRRN27+fk7SpvwItd/F5
eJydll1P2zAUhu8n7T+7c TvG+c5WVz9/fl08ufhZfP46/lycn56NrlCnz8trklTVLRcGlkBb
4wCrguhGFoUjw9P382/3w TvG+8btJ+Ug4GT+6f1ZnM5UZNOAMxrlhF18mrnXk76wWTaaUzTRh
OuxU7FS9hpopBxEbC5lSSuY6gqtJ+Ug4GT+TUseK/q6hlyJhIIFY+fp8iwuE8yCYk40VpXgZVC
feml0i1eSN1IRTYw+Ih1twFDBG569KUuhU/KKgBQ8HVdR37VGeUHDzW9FIdtbx9oK1JJAnnDsWO
xmcrihZbfBIxyv4FYKys2Y5YyAolHgVrDHskz4hMpQU+cJpYJRxb7REam1nnz8DakEpoYSHWs2i
PakJCLbR7T1kIPaZnhEZv0yTSqCnXcWT2z4hMpQU+EiZalfLSQZ0TNQGkPiJHQLR2I3pYFU3V5y
TqrWX/yq7iirzTIWpCoS2blh9esVA0b4Mcm9831BORIJTWy//9q0JDg8AlNnc2ghbZrMQ6TFalnb
uBocflZQ59S0yAvjz0C3J3hsHdOlPQ/XFkLhUZVKYKJ1Q7n1zjGJbMs6q6heNqquSEMTN+Y2Em69
hCZ7X/bCbQts8yNfv67Rwrysnzfr+1fbWg5rFmj1Q7n1zjGJbMs6q6heNqquSEMTN++bT
7ro9tV/H6B7C1UN8GpbdsGmp9eXHRaO9i3DVScz7f37IZue0Bfv1z0DxwqQ49AGXRKPxh6BMrwU
J7tegSyiRRduR3se8TAgrehiBKmXSh6GHTQ5+POR1yANQ9lIb48ZH0YUykXAaYCMKVg5AEoh
J7tegSyiRRduR3se8TAgrehiBKmXSh6GHTQ5+I4LmhAuHlAGiHwQHCkvDjhcVAx4iJAwPfAv4KCHOX0
/7PRRdw87utfFU53bp1X4K/MmvRDh5WLqFhyCJajVz4+qLr4SyEJnFjZhLeSWzyqPTx6KpgOh9k6D
/9z/1gQ1Ww==
```

Internal Message Format for Response Messages

The internal format for response messages parallels that for request messages, and has the same basic MIME structure. These messages are frequently referred to as IBResponse messages.

There are three logical components to a MIME response message: the IBResponse header section, the IBInfo section, and the Content section.

The following code shows an example of a response message:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Message-ID: <32004392.1143500580241.JavaMail.KCOLLIN2@PLE-KCOLLIN2>
Date: Mon, 27 Mar 2006 15:03:00 -0800 (PST)
Mime-Version: 1.0
Content-Type: multipart/related;
 boundary="----=_Part_4_9069393.1143500580221"
Content-ID: PeopleSoft-Integration-Broker-Internal-Mime-Message
PeopleSoft-ToolsRelease: 8.52

------=_Part_4_9069393.1143500580221
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-Disposition: inline
Content-ID: IBInfo

<?xml version="1.0"?><IBInfo><Status><StatusCode>0</StatusCode>
<MsgSet>158</MsgSet>

<MsgID>10000</MsgID><DefaultTitle>Integration Broker Response
Message</DefaultTitle>

</Status><ContentSections><ContentSection><ID>ContentSection0</ID>
<NonRepudiation>N</NonRepudiation></ContentSection></ContentSections></IBInfo>
------=_Part_4_7210339.1008355101202
```

IBResponse Header

The first part of a response message contains headers which describe the attributes of the whole message.

```
Message-ID: <32004392.1143500580241.JavaMail.KCOLLIN2@PLE-KCOLLIN2>
Date: Mon, 27 Mar 2006 15:03:00 -0800 (PST)
Mime-Version: 1.0
Content-Type: multipart/related;
 boundary="----=_Part_4_9069393.1143500580221"
Content-ID: PeopleSoft-Integration-Broker-Internal-Mime-Message
PeopleSoft-ToolsRelease: 8.52
```

IBResponse IBInfo Section

The format for the XML for the IBInfo for a response message is different than that for the request message. The following is a sample (formatted for easier reading):

```
<?xml version="1.0"?>
<IBInfo>
   <Status>
      <StatusCode>0</StatusCode>
      <MsgSet>158</MsgSet>
      <MsgID>10000</MsgID>
      <DefaultMsg>OK</DefaultMsg>
      <DefaultTitle>Integration Broker Response Message</DefaultTitle>
   </Status>
   <ContentSections>
      <ContentSection>
      <ID>ContentSection0</ID>
      <NonRepudiation>N</NonRepudiation>
      </ContentSection>
   </ContentSections>
</IBInfo>
```

The following is the list of all the elements that may be present in the IBInfo for a response:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Element	Description
IBInfo / Status / StatusCode	Describes the result of the request. The possible values are: • *0* (zero). Request successfully processed. • *10*. Temporary error occurred. Request can be resent. • *20*. Fatal error occurred. Do not resend request. • *30*. Request message is a duplicate of a message previously received.
IBInfo / Status / MsgSet	The MessageSetNumber for this message in the Message Catalog. Message set number 158 is assigned to the PeopleSoft Integration Broker.
IBInfo / Status / MsgID	The Message Number for this message in the Message Catalog. If no errors occurred during the processing of the request, the MsgID will be set to the value '10000'.
IBInfo / Status / DefaultTitle	Used if the message catalog is unavailable. This value corresponds to the "Message Text" for a given entry in the message catalog.
IBInfo / Status / DefaultMsg	Used if the message catalog is unavailable. This value corresponds to the "Explanation" for a given entry in the message catalog.
IBInfo / Status / Parameters	Parameters may be used to provide additional information for error responses.
IBInfo / ContentSection	A description of the content section returned with the response. **Note.** Not all response messages will have a content section. The structure of the content section and all child elements is the same as was seen in the request IBInfo.

IBResponse Content Section

The content section of a response message features the message body only when working with SyncRequests

```
<?xml version="1.0"?>
<TestXml>This is a sample response message.</TestXml>
```

Error Codes and Message Catalog Entries

A response message may contain data relating to the processing of the request message, or it may contain error information if there were problems in fulfilling the request.

The status code describes the nature of the response message. The following table describes possible request message status codes and their meaning.

Value	Meaning	Description
0	Success	The message transport and processing were successful.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Value	*Meaning*	*Description*
10	Retry	The transport was not successful. PeopleSoft Integration Broker will perform its retry logic and send the message again.
20	Error	An error occurred.
30	Duplicate message	The transaction ID for the message has already been received.
40	Acknowledgement error	This status is used for SOAP messages and indicates that the contents of the data is not proper, but the transport was successful.
50	Acknowledgement hold	Used for asynchronous chunking of messages from PeopleSoft to PeopleSoft nodes when sending multiple message segments.

All PeopleSoft Integration Broker error messages are stored in the message catalog. A short and long description for every error can be found there. Catalog entries are given a number, and this number is used in the response messages.

Here is a sample error message:

```
Message-ID: <32004392.1143500580241.JavaMail.KCOLLIN2@PLE-KCOLLIN2>
Date: Mon, 27 Mar 2006 15:03:00 -0800 (PST)
Mime-Version: 1.0
Content-Type: multipart/related;
 boundary="----=_Part_4_9069393.1143500580221"
Content-ID: PeopleSoft-Integration-Broker-Internal-Mime-Message
PeopleSoft-ToolsRelease: 8.52

------=_Part_25_2235074.1008270392277
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-Disposition: inline
Content-ID: IBInfo

<?xml version="1.0"?><IBInfo><Status><StatusCode>10</StatusCode>
<MsgSet>158</MsgSet>

<MsgID>10721</MsgID><Parameters count="1"><Parm>404</Parm></Parameters>
<DefaultTitle>Integration Gateway Error</DefaultTitle></Status></IBInfo>
------=_Part_25_2235074.1008270392277--
```

All PeopleSoft Integration Broker errors use message set *158.* The actual error seen here is *10721*. Going to the message catalog, the description for message set *158*, error *10721* is:

```
Message Text: Integration Gateway - External System Contact Error

Explanation: Integration Gateway was not able to contact the external system.
The network location specified may be incorrect, or the site is permanently
or temporarily down.
```

Therefore this error was created by the integration gateway when it tried to send a request message to an external system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Local Compression

This section provides an overview of local compression and discusses how to:

- Set local compression for asynchronous messages.
- Set local compression for synchronous messages.
- Override local compression for synchronous messages.

Understanding Local Compression

The integration engine compresses and base64–encodes messages destined for the PeopleSoft listening connector on its local integration gateway.

Setting Local Compression for Asynchronous Messages

Asynchronous messages are always compressed and base64 encoded when sent to the integration gateway. There are no settings you need to make.

Setting Local Compression for Synchronous Messages

In PSAdmin you can set a threshold message size above which the system compresses synchronous messages. The setting is shown here:

```
Values for config section - Integration Broker
    Min Message Size For Compression=10000

Do you want to change any values (y/n)? [n]:
```

The value is the message size in bytes; the default value is *10000* (10 kilobytes). You can specify a setting of *0* to compress all messages.

To turn off compression, set the value to *-1*.

Warning! Turning compression off can negatively impact system performance when transporting synchronous messages greater than 1 MB. As a result, you should turn off compression only during integration development and testing.

Note. This setting does not affect the compression of messages that the integration gateway sends using its target connectors.

Overridding Local Compression for Synchronous Messages

You can override the PSAdmin message compression setting for synchronous messages at the transaction level. The following method on the IBInfo object in the Message class is provided for this purpose:

```
&MSG.IBInfo.CompressionOverride
```

The valid parameters for this method are: %IntBroker_Compress, %IntBroker_UnCompress, and %IntBroker_Compress_Reset.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See *PeopleTools 8.52: PeopleCode API Reference*, "Message Classes."

See Also

PeopleTools 8.52: System and Server Administration, "Setting Application Server Domain Parameters"

Accessing IBInfo Elements Using PeopleCode

You can use the PeopleCode Message object to access IBRequest and IBResponse IBInfo data.

The following example demonstrates reading target connector information on a notification method for a rowset-based asynchronous message.

```
method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   /* Variable Declaration */

   integer &i;
   string &strReturn;
   rowset &RS;

   For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfConnectorProperties()
   /* get Query arguments */

      &strReturn = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgName(&i);
      &strReturn = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgValue(&i);

   End-For;

   /* access the content data */

&RS = &MSG.GetRowset();

end-method;
```

The following example demonstrates reading target connector information on notification method for a nonrowset-based asynchronous message.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   /* Variable Declaration */

   integer &i;

   string &&strReturn;

   xmldoc &xmldoc;

   For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfConnectorProperties()

      &strReturn = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgName(&i);
      &strReturn = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgValue(&i);

   End-For;

   /* access the content data */

&xmldoc = &MSG.GetXmlDoc();

end-method;
```

If an HTTP header is passed with a dollar sign ($), PeopleSoft Integration Broker converts the dollar sign to an underscore (_).

PeopleSoft Rowset-Based Message Format

This section discusses the PeopleSoft rowset-based message format and discusses:

- FieldTypes section of a rowset-based message.
- MsgData section of a rowset-based message.
- PeopleSoft rowset-based message example.
- PeopleSoft timestamp format.
- CDATA and special characters.
- Schema restrictions.

This section also provides an example of a rowset-based message.

See Also

Chapter 5, "Understanding Supported Message Structures," Message Part Structures, page 69

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding the PeopleSoft Rowset-Based Message Format

To work with rowset-based messages—the PeopleSoft native format—you define a message in the PeopleSoft Pure Internet Architecture, insert records into the message definition in a hierarchical structure, and then populate the message and manipulate its contents by using the PeopleCode Rowset and Message classes. Externally, the message is transmitted as XML with a prescribed PeopleSoft schema.

The PeopleSoft message schema includes a PSCAMA record, which PeopleTools adds to every level of the message structure to convey basic information about the message and its data rows.

The Rowset and IntBroker classes are recommended for messaging between PeopleSoft applications. If a message is populated with data from a PeopleSoft application's database or component buffer, the Message class is best for handling that data.

Record and Field Aliases

You can specify an alias for any record or field in a rowset-based message definition. Each node participating in a transaction maintains its own independent definition of the message and its versions, including record and field names and their aliases.

When you send a message with an alias defined and the message is converted to XML for sending, only the alias appears in the XML. If you don't specify an alias, the original name is used. If the service operation is routed to multiple target nodes with different record or field naming schemes, you create for each target a separate service operation version with its own aliases and send each version separately.

The only requirement for a successful transaction is that the record and field names in the XML match either the original names or the aliases that are defined for that message and version at the node receiving the message. This behavior applies to both request and synchronous response messages, but typically only the source node applies aliases to accommodate the target node's naming scheme in both directions.

In a synchronous transaction, the request and response messages can be completely different from each other. Upon receiving a synchronous request, the target node generates and sends a response message. Upon receiving the response, the source node uses its defined aliases to find and reapply its original record and field names. The resulting inbound message contains the original names that were defined at the source node, not the aliases. Therefore, both the sending and receiving PeopleCode at the source node should expect to work with the source node's original record and field names.

See Also

Chapter 5, "Understanding Supported Message Structures," PSCAMA, page 59

Chapter 7, "Sending and Receiving Messages," Understanding Integration PeopleCode, page 129

Chapter 18, "Applying Filtering, Transformation and Translation," page 415

Rowset-Based Message Template

The following template shows the overall structure of a message in the PeopleSoft rowset-based message format:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
   <psft_message_name>
      <FieldTypes>
         ...
      </FieldTypes>
      <MsgData>
         <Transaction>
            ...
         </Transaction>
      </MsgData>
   </psft_message_name>
```

Note. *Psft_message_name* is the name of the message definition in the PeopleSoft database. Integration Broker inserts this message content into a standard PeopleSoft XML message wrapper for transmission.

FieldTypes Section

Each PeopleSoft message includes *field type* information. Fieldtype information conveys the name of each data record and its constituent fields, along with each field's data type. Your receiving application can use this information to validate data types. The field type information is contained in the *FieldTypes* section of the message.

There are two FieldTypes tags:

- Each record tag consists of the name of a record, followed by a *class* attribute with a single valid value: *R.* The record tag encloses that record's field tags.
- Each field tag consists of the name of a field, followed by a *type* attribute with three valid values: *CHAR* for a character field, *DATE* for a date field, and *NUMBER* for a numeric field.

Following is a simple FieldTypes template.

```
<FieldTypes>
   <recordname1 class="R">
      <fieldname1 type="CHAR"/>
      <fieldname2 type="DATE"/>
      <fieldname3 type="NUMBER"/>
   </recordname1>
   <recordname2 class="R">
      <fieldname4 type="NUMBER"/>
   </recordname2>
<FieldTypes>
```

Note. Third-party sending applications must include a valid FieldTypes section in each message. The PeopleSoft system expects fieldtype information for each record and field in the message.

MsgData Section

In addition to field type information, each PeopleSoft message contains data content in the *MsgData* section of the message. Between the MsgData tags are one or more *Transaction* sections. Each transaction represents one row of data.

Between the Transaction tags is a rowset hierarchy of records and fields. The record tags at each level contain the fields for that record, followed by any records at the next lower level.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The last record within a transaction is a fully specified PeopleSoft Common Application Message Attributes (PSCAMA) record, which provides information about the entire transaction. Immediately following the closing tag of *every* record below level 0 is a PSCAMA record containing only the AUDIT_ACTN field that specifies the action for that record.

Simple MsgData Template

Following is a simple MsgData template.

Note. The PSCAMA PUBLISH_RULE_ID and MSGNODENAME fields (shown emphasized) are used internally by certain PeopleSoft utilities, but third-party systems can generally ignore them and don't need to include them in messages.

```
<MsgData>
   <Transaction>
      <level0recname1 class="R">
         <fieldname1>value</fieldname1>
         <fieldname2>value</fieldname2>
         <level1recname1 class="R">
            <fieldname3>value</fieldname3>
            <fieldname4>value</fieldname4>
         </level1recname1>
         <PSCAMA class="R">
            <AUDIT_ACTN>value</AUDIT_ACTN>
         </PSCAMA>
         <level1recname2 class="R">
            <fieldname5>value</fieldname5>
         </level1recname2>
         <PSCAMA class="R">
            <AUDIT_ACTN>value</AUDIT_ACTN>
         </PSCAMA>
      </level0recname1>
      <level0recname2 class="R">
         <fieldname6>value</fieldname6>
      </level0recname2>
      <PSCAMA class="R">
         <LANGUAGE_CD>value</LANGUAGE_CD>
         <AUDIT_ACTN>value</AUDIT_ACTN>
         <BASE_LANGUAGE_CD>value</BASE_LANGUAGE_CD>
         <MSG_SEQ_FLG>value</MSG_SEQ_FLG>
         <PROCESS_INSTANCE>value</PROCESS_INSTANCE>
        <PUBLISH_RULE_ID>value</PUBLISH_RULE_ID>
        <MSGNODENAME>value</MSGNODENAME>
      </PSCAMA>
   <Transaction>
</MsgData>
```

See Also

Chapter 5, "Understanding Supported Message Structures," PSCAMA, page 59

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PSCAMA

PeopleTools adds the PSCAMA record to every level of the message structure during processing. It isn't accessible in the message definition, but you can reference it as part of the Message object in the sending and receiving PeopleCode, and you can see it in the Integration Broker Monitor. PeopleCode processes this record the same way as any other record.

Note. PSCAMA records are automatically included in messages only if you insert database records to define the message structure. You can use the PeopleCode XmlDoc class to handle an inbound message containing PSCAMA records, but the PeopleCode Message class is much better suited for this.

PSCAMA contains fields that are common to all messages. The <PSCAMA> tag repeats for each row in each level of the transaction section of the message. The sender can set PSCAMA fields to provide basic information about the message; for example, to indicate the message language or the type of transaction a row represents. When receiving a message, your PeopleCode should inspect the PSCAMA records for this information and respond accordingly.

PSCAMA Record Definition

The PSCAMA record definition includes the following fields:

LANGUAGE_CD Indicates the language in which the message is generated, so the receiving application can take that information into account when processing the message. When sending a message with component PeopleCode, the system sets this field to the user's default language code.

AUDIT_ACTN Identifies each row of data as an Add, Change, or Delete action.

BASE_LANGUAGE_CD (Optional.) Indicates the base language of the sending database. This is used by generic, full-table subscription PeopleCode to help determine which tables to update.

MSG_SEQ_FLG (Optional.) Supports the transmission of large transactions that may span multiple messages. Indicates whether the message is a header *(H)* or trailer *(T)* or contains data (blank). The header and trailer messages don't contain message data. The receiving system can use this information to determine the start and end of the set of messages and initiate processes accordingly. For example, the header message might cause staging tables to be cleared, while the trailer might indicate that all of the data has been received and an update job should be initiated.

PROCESS_INSTANCE (Optional.) Process instance of the batch job that created the message. Along with the sending node and publication ID, the receiving node can use this to identify a group of messages from the sending node.

PUBLISH_RULE_ID (Optional.) Indicates the publish rule that is invoked to create the message. This is used by routing PeopleCode to locate the appropriate chunking rule, which then determines to which nodes the message should be sent. Third-party applications can ignore this field.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

MSGNODENAME (Optional.) The node to which the message should be sent. This field is passed to the Publish utility by the Application Engine program. Routing PeopleCode must look for a value in this field and return that value to the application server. Third-party applications can ignore this field.

Language Codes

Each message can contain only one language code (the LANGUAGE_CD field) in the first PSCAMA record.

PeopleSoft language codes contain three characters and are mapped to corresponding International Organization for Standardization (ISO) locale codes in an external properties file. This mapping enables the PeopleSoft Pure Internet Architecture to derive certain defaults from the ISO locales that are stored in a user's browser settings. Your PeopleSoft application is delivered with a set of predefined language codes; you can define your own codes, as well.

Note. There can be only one language code for the entire message. To send messages in multiple languages, send multiple messages.

See *PeopleTools 8.52: Global Technology*, "Controlling International Preferences."

Audit Action Codes

A PSCAMA record appears following each row of the message. At a minimum, it contains an audit action code (the AUDIT_ACTN field), denoting the action to be applied to the data row. The audit action is required so that the receiving system knows how to process the incoming data.

The valid audit action codes match those that are used in the PeopleSoft audit trail processing: *A,C,D,K,N,O,* and blank.

The audit action values are set by the system or by the sending PeopleCode to specify that a record should be added, changed, or deleted:

Audit Action Code	*Description*
A	Add a noneffective or effective-dated row. To add an effective-dated row, the value is *A.* If you populate the row data by using the CopyRowsetDeltaOriginal method in the PeopleCode Message class, an additional record is created with an audit action value of *O*, containing the original values of the current effective-dated row.
C	Change non-key values in a row.
D	Delete a row.
K	If you change at least one key value in a row (in addition to any non-key values) and then populate the row data by using the CopyRowsetDeltaOriginal or CopyRowsetDelta methods in the Message class, an additional record is created with an audit action value of *K*, containing the original values of the current effective-dated row.
N	Change at least one key value in a row (in addition to any non-key values).

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Audit Action Code	Description
O	If you change non-key values in a row and populate the row data by using the CopyRowsetDeltaOriginal method in the Message class, an additional record is created with an audit action value of *O*, containing the original values of the current effective-dated row.
Blank	Default value. If a row's content hasn't changed, the value is blank. This audit action code is also used to tag the parents of rows that have changed.

Other PSCAMA Considerations

You can update values on the PSCAMA record just like any other record in the message definition before sending the message.

The receiving process can access the fields in this record just like any other fields in the message.

The size of the PSCAMA record varies. In particular, notice a difference between the first PSCAMA record and the ones that follow. The first record contains all of the fields, while the other PSCAMA records have only the AUDIT_ACTN field, which is the only field that can differ for each row of data.

Although the first PSCAMA record is always present, not all of the remaining PSCAMA records are sent in the message. If a <PSCAMA> tag is not included for a specific row, you can assume that the row hasn't changed. In addition, if the <AUDIT_ACTN> tag is blank or null, you can also assume the row hasn't changed.

If you're receiving a message that has incremental changes, only the rows that have changed *and their parent rows* are present in the message.

You can view an example of an outbound message with the PSCAMA records inserted by testing your message definition using the Schema Tester.

See *PeopleTools 8.52: Integration Broker Testing Utilities and Tools*, "Using the Schema Tester Utility."

Identifying Changes to Field-Level Attributes

When sending and receiving messages, all blank data values get stripped. As a result, you cannot determine if a field value is blank by definition, or if its value was stripped in the messaging process.

The PeopleCode CopyRowset functions CopyRowset, CopyRowsetDelta and CopyRowsetDeltaOriginal, feature an IsChanged attribute that automatically gets set to identify fields that have been changed. Any field that has been changed displays the attribute *IsChanged="Y"*.

Note. The IsChanged attribute applies only to rowset-based messages. For rowset-based message parts, use the Message Part Default Indicator field to distinguish blanks from zeros in part messages. The IsChanged attribute does not apply to nonrowset-based messages, including nonrowset-based container messages and nonrowset-based part messages.

For example:

```
<QE_ACNUMBER IsChanged="Y">2</QE_ACNUMBER>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Fields that had data and then were blanked contain the IsChanged attribute.

For example:

```
<DESCRLONG IsChanged="Y"/>
```

Fields that were always blank and thus were not changed do not feature this attribute. For example:

```
<QE_NAVDESC/>
```

If you are writing subscription PeopleCode you reference the IsChanged value of the field in the message rowset, as always. However, the blanks appear with the attribute *IsChanged="Y"*.

See Also

Chapter 6, "Managing Messages," Distinguishing Blank from Zero in Rowset-Based Part Messages, page 104

PeopleSoft Timestamp Format

The PeopleSoft format for all timestamps is ISO-8601. If any message fields are type timestamp, the following format is used:

```
CCYY-MM-DDTHH:MM:SS.ssssss+/-hhmm
```

Note. The ISO format specifies that the `+/-hhmm` parameter is optional, but PeopleSoft requires it. All date and time stamps in the header and the body of the message must include the Greenwich Mean Time (GMT) offset as `+/-hhmm`. This ensures that the timestamp is correctly understood by the receiving application.

CDATA and Special Characters

Consider the following points regarding rowset-based messages:

- You cannot use CDATA in message XML if you plan to use GetRowSet to parse the message.
- When using the ampersand character (&) in a string, it must be URL-encoded. For example: *&amp.* Passing only the ampersand character results in a PeopleCode error when you get the rowset values.
- Other special characters are best passed encoded as well, such as *&gt* for "<" and *&lt* ">."

Schema Restrictions

For stronger schema validation control, some PeopleSoft field types have certain implicit restrictions regarding the format of field data that is acceptable in a runtime message. These restrictions appear in message schema.

The restrictions apply to fields having the following formats.

- Mixed case.
- Name.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Phone number.
- Social security number.
- Uppercase.
- Zip code.

Note. These restrictions apply to rowset-based messages and rowset-based message parts.

The restrictions for each are shown in the following example:

```
<xsd:simpleType name="BASE_LANGUAGE_CD_TypeDef">
   <xsd:annotation>
     <xsd:documentation>BASE_LANGUAGE_CD is a character of length 3.
       Allows Uppercase characters including numbers
     </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="3"/>
     <xsd:whiteSpace value="preserve"/>
     <xsd:pattern value="([A-Z]|[0-9]|\p{Z}|\p{P}|\p{Lu})*"/>
   </xsd:restriction>
 </xsd:simpleType>
```

Rowset-Based Message Example

The message data is enclosed in a tag with the name of the message, and consists of one FieldTypes section followed by one MsgData section. The FieldTypes section describes the records and fields that appear in the MsgData section, which contains the actual data.

Note. The PSCAMA record requires field type information just like any other record.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<SDK_BUS_EXP_APPR_MSG>
   <FieldTypes>
      <SDK_BUS_EXP_PER class="R">
         <SDK_EMPLID type="CHAR"/>
         <SDK_EXP_PER_DT type="DATE"/>
         <SDK_SUBMIT_FLG type="CHAR"/>
         <SDK_INTL_FLG type="CHAR"/>
         <SDK_APPR_STATUS type="CHAR"/>
         <SDK_APPR_INSTANCE type="NUMBER"/>
         <SDK_DESCR type="CHAR"/>
         <SDK_COMMENTS type="CHAR"/>
      </SDK_BUS_EXP_PER>
      <SDK_DERIVED class="R">
         <SDK_BUS_EXP_SUM type="NUMBER"/>
      </SDK_DERIVED>
      <SDK_BUS_EXP_DTL class="R">
         <SDK_CHARGE_DT type="DATE"/>
         <SDK_EXPENSE_CD type="CHAR"/>
         <SDK_EXPENSE_AMT type="NUMBER"/>
         <SDK_CURRENCY_CD type="CHAR"/>
         <SDK_BUS_PURPOSE type="CHAR"/>
         <SDK_DEPTID type="CHAR"/>
      </SDK_BUS_EXP_DTL>
      <PSCAMA class="R">
         <LANGUAGE_CD type="CHAR"/>
         <AUDIT_ACTN type="CHAR"/>
         <BASE_LANGUAGE_CD type="CHAR"/>
         <MSG_SEQ_FLG type="CHAR"/>
         <PROCESS_INSTANCE type="NUMBER"/>
      </PSCAMA>
   </FieldTypes>
   <MsgData>
      <Transaction>
         <SDK_BUS_EXP_PER class="R">
            <SDK_EMPLID>8001</SDK_EMPLID>
            <SDK_EXP_PER_DT>1998-08-22</SDK_EXP_PER_DT>
            <SDK_SUBMIT_FLG>N</SDK_SUBMIT_FLG>
            <SDK_INTL_FLG>N</SDK_INTL_FLG>
            <SDK_APPR_STATUS>P</SDK_APPR_STATUS>
            <SDK_APPR_INSTANCE>0</SDK_APPR_INSTANCE>
            <SDK_DESCR>Regional Users Group Meeting</SDK_DESCR>
            <SDK_COMMENTS>Attending Northeast Regional Users Group
            Meeting and presented new release functionality.
            </SDK_COMMENTS>
            <SDK_BUS_EXP_DTL class="R">
               <SDK_CHARGE_DT>1998-08-22</SDK_CHARGE_DT>
               <SDK_EXPENSE_CD>10</SDK_EXPENSE_CD>
               <SDK_EXPENSE_AMT>45.690</SDK_EXPENSE_AMT>
               <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
               <SDK_BUS_PURPOSE>Drive to Meeting</SDK_BUS_PURPOSE>
               <SDK_DEPTID>10100</SDK_DEPTID>
            </SDK_BUS_EXP_DTL>
            <PSCAMA class="R">
               <AUDIT_ACTN>A</AUDIT_ACTN>
            </PSCAMA>
            <SDK_BUS_EXP_DTL class="R">
               <SDK_CHARGE_DT>1998-08-22</SDK_CHARGE_DT>
               <SDK_EXPENSE_CD>09</SDK_EXPENSE_CD>
               <SDK_EXPENSE_AMT>12.440</SDK_EXPENSE_AMT>
               <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
               <SDK_BUS_PURPOSE>City Parking</SDK_BUS_PURPOSE>
               <SDK_DEPTID>10100</SDK_DEPTID>
            </SDK_BUS_EXP_DTL>
            <PSCAMA class="R">
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
                  <AUDIT_ACTN>A</AUDIT_ACTN>
               </PSCAMA>
            </SDK_BUS_EXP_PER>
            <SDK_DERIVED class="R">
               <SDK_BUS_EXP_SUM>58.13</SDK_BUS_EXP_SUM>
            </SDK_DERIVED>
            <PSCAMA class="R">
               <LANGUAGE_CD>ENG</LANGUAGE_CD>
               <AUDIT_ACTN>A</AUDIT_ACTN>
               <BASE_LANGUAGE_CD>ENG</BASE_LANGUAGE_CD>
               <MSG_SEQ_FLG></MSG_SEQ_FLG>
               <PROCESS_INSTANCE>0</PROCESS_INSTANCE>
            </PSCAMA>
         </Transaction>
      </MsgData>
</SDK_BUS_EXP_APPR_MSG>
```

Nonrowset-Based Message Structures

This section discusses nonrowset-based message structures that you can use with PeopleSoft Integration Broker. This section discusses:

- XML messages.
- SOAP-compliant messages.
- Non-XML files.

XML Messages

The World Wide Web Consortium (W3C) has established a Document Object Model (DOM) for accessing and manipulating structured data. The DOM specifies a standardized application programming interface (API) that provides a consistent, familiar way to work with almost any XML data. This API—the XML DOM—enables you to create, retrieve, navigate, and modify messages.

You define an XML message in the PeopleSoft Pure Internet Architecture by either uploading an XML file or entering an XML schema definition. The following example shows an XML message schema:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace=
"http://xmlns.oracle.com/Common/schemas/COMPANY" xmlns="http://xmlns.
oracle.com/Common/schemas/COMPANY" elementFormDefault="qualified">
  <xsd:element name="Company" type="CompanyType"/>
  <xsd:complexType name="CompanyType">
     <xsd:sequence>
        <xsd:element name="Person" type="PersonType"/>
        <xsd:element name="Product" type="ProductType"/>
     </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="PersonType">
     <xsd:sequence>
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="SSN" type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ProductType">
     <xsd:sequence>
        <xsd:element name="Type" type="xsd:string"/>
     </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Then populate the message and manipulate its contents by using the PeopleCode XmlDoc class and built-in functions, which reflect the XML DOM.

Note. You can use the XmlDoc class to access inbound, rowset-based messages; however, the PeopleCode Message and Rowset classes handle the PeopleSoft native format more easily.

Use the XmlDoc class if *any* of the following is true:

- The message structure doesn't fit the PeopleSoft rowset model.
- The message data doesn't come from PeopleSoft database records.
- The third-party source or target node requires non-XML message data.

Although you can use the XmlDoc class to generate or process messages that use the SOAP protocol, the PeopleCode SoapDoc class is more efficient and is strongly recommended.

Note. Non-XML message data must be embedded in an XML wrapper, which you send and receive by using the XmlDoc class.

SOAP-Compliant Messages

The W3C SOAP specification defines synchronous transactions in a distributed web environment. SOAP is appropriate for Universal Description, Discovery, and Integration (UDDI) interactions, or to interact with SOAP-compliant servers.

You define a message in PeopleSoft Application Designer without inserting any records to define its structure, then populate the message and manipulate its contents by using the PeopleCode SoapDoc class and built-in functions, which comply with the W3C SOAP specification. The SoapDoc class is well-suited for messages that are populated with SOAP-compliant XML data.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

SoapDoc complies with the W3C XML DOM specification. The SoapDoc class is based on the PeopleCode XmlDoc class, with some identical methods and properties. To send and receive SoapDoc messages, you must convert them to XmlDoc objects and use the XMLDoc built-in functions, SyncRequestXmlDoc and GetMessageXmlDoc. SoapDoc provides a property for handling the conversion easily.

Use the SoapDoc class if *all* of the following are true:

- The third-party source or target node requires SOAP-compliant messages.
- The third-party source or target node requires synchronous transactions.
- The message conforms to the SOAP specification.

See Also

Chapter 7, "Sending and Receiving Messages," Generating and Sending Messages, page 144

Chapter 7, "Sending and Receiving Messages," Receiving and Processing Messages, page 155

Non-XML Files

To send non-XML files through PeopleSoft Integration Broker to their destination, you must wrap them in the PeopleSoft non-XML message element, CDATA. However, when you send messages to third-party systems, the recipient systems may not be able to interpret that element.

The Message class features a method, SetXMLDoc, that you can use to remove the tags upon execution of the method. Another alternative to removing the tags is to write a transformation to do so. If you do not use either option, the data remains in the wrapper through to the destination.

The following code example shows a non-XML file wrapped in the PeopleSoft non-XML message element, PsNonXmL, for transport through PeopleSoft Integration Broker:

Note. The element PsNonXml is not case-sensitive.

```
<?xml version="1.0"?>
<AsyncRequest>
  <data PsNonXml="Yes">
    <![CDATA[<?xml version="1.0"?>101 123456789
    12345678902 0510145 60094101First Bank First Bank 5200 University
    000001 PPDDIRECT PAY020510020510000112345678000000162200000111 222
    0000001000USA0000001 USA0000001 000000111000000162712345 6
    789131415511 0000001000 University 0123456780000
    002 820000000200123457890000000010000000000001000 123456780000001
    900000100000100000002001234578900000000100000000000100099999999999
    999999999999999999999999999999999999999999999999999999999999999999
    999999999999999999999999999999999999999999999999999999999999999999
    999999999999999999999999999999999999999999999999999999999999999999
    999999999999999999999999999999999999999999999999999999999999999999
    999999999999999999999999999999999999999999999999999999999999999999
    99999999999999999999999999999999999999
   ]]>
    </data>
</AsyncRequest>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following example shows an alternative way to wrap a non-XML file in the PeopleSoft non-XML message element for transport through PeopleSoft Integration Broker:

```
<?xml version="1.0"?>
<AsyncRequest PsNonXml = 'Yes'>
<![CDATA[<?xml version="1.0"?>101 123456789 12345678902
0510145 60094101First Bank First Bank 5200 University 000001 PPDDIRECT
PAY020510020510000112345678000000162200000111 222
0000001000USA0000001 USA0000001 000000111000000162712345 6
789131415511 0000001000 University 0123456780000
002 820000000200123457890000000010000000000001000 123456780000001
900000100000100000002001234578900000000100000000000100099999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999
]]>
</AsyncRequest>
```

The following example shows using the SetXMLDoc method to remove the PsNonXML wrapper from a message:

```
// create xmldoc with PSNonXml tag and load it into the message

// create an instance of the Message object
&MSG = CreateMessage(OPERATION.QE_F18_ASYNC_XMLDOC);

// Load the Message object via the SetXmlDoc method with the xmldoc data.
// This will also remove the PSNonXml wrapper
&MSG.SetXmlDoc(&XmlDoc);

// perform a publish for the nonrowset-based message
%IntBroker.Publish(&MSG);
```

As stated previously in this section, if you do not use the SetXMLDoc method to remove the PsNonXml wrapper, you must write a transformation to physically remove the tags.

See Also

PeopleTools 8.52: PeopleCode API Reference, "Message Classes"

Chapter 18, "Applying Filtering, Transformation and Translation," page 415

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Using Listening Connectors and Target Connectors," Complying With Message Formatting and Transmission Requirements

Using Nonrowset-Based Messages in Service Operations Exposed as WSDL

If a nonrowset-based message is used in a service operation which will be exposed as a WSDL to third parties, the schema cannot be empty. The schema has to have at least the header elements, as shown in the following example:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"/>
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Message Part Structures

This section discusses:

- Rowset-based message parts.
- Nonrowset-based message parts.

Understanding Message Part Structures

Message parts are rowset-based messages or nonrowset-based messages that you designate as a part message on the message definition. Message parts are used in container messages

Message parts can be re-used in multiple containers.

All parts in a container must be of the same type (Rowset-based or Nonrowset-based).

You create messages using the Message Builder page in the PeopleSoft Pure Internet Architecture.

See Also

Chapter 5, "Understanding Supported Message Structures," PeopleSoft Rowset-Based Message Format, page 55

Chapter 5, "Understanding Supported Message Structures," Nonrowset-Based Message Structures, page 65

Chapter 6, "Managing Messages," page 75

Rowset-Based Message Parts

Rowset-based message parts provide all the ease of use of using rowsets, yet the generated XML message is industry standard and not PeopleSoft proprietary. Rowset-based message parts, like nonrowset-based parts, can only be part of a container type message.

These are the benefits of using Rowset-based parts:

- The XML schema generated is standard XML and not the PeopleSoft message format. Rowset-based message parts do not have a PSCAMA section, FieldTypes section, IsChanged attributes, and so forth.
- The message API for rowset-based parts is simple to use and understand.
- XML serialization and deserialization to and from part rowset is provided by Integration Broker framework.
- You can use a CopyRowSet type method to populate the rowset from another rowset (component rowset).

The following example shows a sample schema from a rowset-based message part:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<xsd:schema elementFormDefault="qualified" targetNamespace="http://xmlns.
oracle.com/Enterprise/Tools/schemas/Part_1.V1" xmlns="http://xmlns.oracle.
com/Enterprise/Tools/schemas/Part_1.V1" xmlns:xsd="http://www.w3.org/
2001/XMLSchema">
  <xsd:element name="Part_1" type="Part_1_TypeShape"/>
  <xsd:complexType name="Part_1_TypeShape">
    <xsd:sequence>
      <xsd:element name="First_Part" type="First_PartMsgDataRecord_TypeShape"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="First_PartMsgDataRecord_TypeShape">
    <xsd:sequence>
      <xsd:element name="QE_ACNUMBER" type="QE_ACNUMBER_TypeDef"/>
      <xsd:element name="QE_WAYPOINT_NBR" type="QE_WAYPOINT_NBR_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_BEARING" type="QE_BEARING_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_RANGE" type="QE_RANGE_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_ALTITUDE" type="QE_ALTITUDE_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_LATITUDE" type="QE_LATITUDE_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_LONGITUDE" type="QE_LONGITUDE_TypeDef"/>
      <xsd:element name="QE_HEADING" type="QE_HEADING_TypeDef"/>
      <xsd:element name="QE_VELOCITIES" type="QE_VELOCITIES_TypeDef"/>
      <xsd:element minOccurs="0" name="QE_NAVDESC" type="QE_NAVDESC_TypeDef"/>
    </xsd:sequence>
    <xsd:attribute fixed="R" name="class" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:simpleType name="QE_ACNUMBER_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_ACNUMBER is a number of length 10 with a decimal
        position of 0</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:integer">
      <xsd:totalDigits value="10"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_WAYPOINT_NBR_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_WAYPOINT_NBR is a number of length 3 with a decimal
        position of 0</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:integer">
      <xsd:totalDigits value="3"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_BEARING_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_BEARING is a character of length 10</xsd:⇒
documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="10"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_RANGE_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_RANGE is a character of length 10</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="10"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_ALTITUDE_TypeDef">
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
    <xsd:annotation>
      <xsd:documentation>QE_ALTITUDE is a character of length 10</xsd:⇒
documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="10"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_LATITUDE_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_LATITUDE is a character of length 15
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="15"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_LONGITUDE_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_LONGITUDE is a character of length 15
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="15"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_HEADING_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_HEADING is a character of length 4
     </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="MAG"/>
      <xsd:enumeration value="TRUE"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_VELOCITIES_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_VELOCITIES is a character of length 4
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ADC"/>
      <xsd:enumeration value="GPS"/>
      <xsd:enumeration value="INS"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="QE_NAVDESC_TypeDef">
    <xsd:annotation>
      <xsd:documentation>QE_NAVDESC is a character of length 30
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="30"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Nonrowset-Based Message Parts

A nonrowset-based message part schema is similar to a regular nonrowset-based message, however a nonrowset-based message part can be reused in multiple containers.

Message Container Structures

Message container structures hold rowset-based or nonrowset-based message part structures. All message parts assigned to a container must of the same type, rowset-based or nonrowset-based.

A message container is always a nonrowset-based message.

You create container messages using the Message Builder in the PeopleSoft Pure Internet Architecture.

See Also

Chapter 5, "Understanding Supported Message Structures," Nonrowset-Based Message Structures, page 65

Chapter 6, "Managing Messages," page 75

Example 1: XML Schema of a Container Message with Rowset-Based Message Parts

The following example shows a sample schema of a container message with three rowset-based message parts:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<xsd:schema elementFormDefault="unqualified" targetNamespace="http://xmlns.
oracle.com/Enterprise/Tools/schemas/Part_Container.V1"
xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/Part_Container.V1"
xmlns:Part_1.V1="http://xmlns.oracle.com/Enterprise/Tools/schemas/Part_1.V1"
xmlns:Part_2.V1="http://xmlns.oracle.com/Enterprise/Tools/schemas/Part_2.V1"
xmlns:Part_3.V1="http://xmlns.oracle.com/Enterprise/Tools/schemas/Part_3.V1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/schemas/
    Part_1.V1" schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
     ServiceListeningConnector?Operation=GetSchema&amp;xsd=Part_1.V1"/>
  <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/schemas/
     Part_3.V1" schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
     ServiceListeningConnector?Operation=GetSchema&amp;xsd=Part_3.V1"/>
  <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/schemas/
     Part_2.V1" schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
     ServiceListeningConnector?Operation=GetSchema&amp;xsd=Part_2.V1"/>
  <xsd:element name="Part_Container" type="Part_ContainerType"/>
  <xsd:complexType name="Part_ContainerType">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Part_1" type="
         Part_1.V1:Part_1_TypeShape"/>
      <xsd:element maxOccurs="10" minOccurs="0" name="Part_3" type="Part_3.V1:
         Part_3_TypeShape"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Part_2" type="
         Part_2.V1:Part_2_TypeShape"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example 2: XML Schema of a Container Message with Nonrowset-Based Message Parts

The following example shows a sample schema from a container message that contains three nonrowset-based parts:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<xsd:schema elementFormDefault="unqualified" targetNamespace="http://xmlns.
 oracle.com/Enterprise/Tools/schemas/NonRowSetContainer.v1"
xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/NonRowSetContainer.v1"
xmlns:Part_One_NonRowset.v1="http://xmlns.oracle.com/Enterprise/Tools/
   schemas/Part_One.v1"
xmlns:Part_Three_NonRowset.v1="http://xmlns.oracle.com/Enterprise/Tools/
   schemas/Part_Two.v1"
xmlns:Part_Two_NonRowset.v1="http://xmlns.oracle.com/Enterprise/Tools/
   schemas/Part_Three.v1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
    ServiceListeningConnector?Operation=GetSchema&amp;xsd=Part_One_NonRowset.v1"/>
  <xsd:import schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
    ServiceListeningConnector?Operation=GetSchema&amp;xsd=Part_Two_NonRowset.v1"/>
  <xsd:import schemaLocation="http://kcollin2042803:5000/PSIGW/PeopleSoft
     ServiceListening Connector?Operation=GetSchema&amp;xsd=Part_Three_Non
Rowset.v1"/>
  <xsd:element name="NonRowSetContainer" type="NonRowSetContainerType"/>
  <xsd:complexType name="NonRowSetContainerType">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Part_One_NonRowset"
        type="Part_One_NonRowset.v1:Part_One_TypeShape"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Part_Two_NonRowset"
        type="Part_Two_NonRowset.v1:Part_Two_TypeShape"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Part_Three_NonRowset"
         type="Part_Three_NonRowset.v1:Part_Three_TypeShape"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 6

Managing Messages

This chapter provides an overview of managing messages and discusses how to:

- Search for message definitions.
- Add message definitions.
- Manage rowset-based messages.
- Manage nonrowset-based messages.
- Manage message parts.
- Reuse message parts.
- Manage container messages.
- Manage document messages.
- View service operations that reference a message.
- Resolve inconsistencies in exported WSDL and WADL documents.
- Rename and delete message definitions.
- Delete messages during upgrade.

Understanding Managing Messages

This section provides an overview of messages.

Message Definitions

Message definitions provide the physical description of the data that is being sent, including fields, field types, and field lengths. You create message definitions in the PeopleSoft Internet Architecture.

Note. Messages are shapes that describe the contents of a service operation transaction. Unlike prior PeopleTools releases, messages do not contain any processing logic. All processing logic is defined in service operations, using service operation handlers.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Message Types

Five types of messages are available:

Rowset-based messages For hierarchical data that is based on PeopleSoft records, you create a message definition by assembling records, organizing them into a hierarchy, and selecting fields from those records to include in the message. The result is a rowset that doesn't need to match an existing rowset structure in the application. Use the PeopleCode Rowset and operation classes to generate, send, receive, and process these messages.

Nonrowset-based messages These messages can have virtually any structure and content. You create a message definition, but you do not insert any records. The message definition serves as a placeholder for the actual message. Use the PeopleCode XmlDoc and operation classes to generate, send, receive, and process these messages. If you're handling Simple Object Access Protocol (SOAP) compliant data, you can also use the SoapDoc class to generate and process these messages.

Container messages A container message is a nonrowset-based message that holds one or more part messages.

A container message must contain all rowset-based messages *or* all nonrowset-based message parts.

Message parts Message parts are rowset-based messages or nonrowset-based messages that you designate as a part message, to be used in a container message.

Document messages A document messages are messages based on the PeopleSoft XML document technology. You create and manage these messages in the PeopleSoft Document Builder, either by creating the XML documents from the ground up, importing them from schema definitions, or from PeopleSoft records.

The following table describes when to use a given message type:

Message Type	***When to Use***
Rowset-based message.	All PeopleSoft-to-PeopleSoft integrations.
Nonrowset-based message.	Integrations with third-party systems.
Container message with rowset-based message parts.	Exposing PeopleSoft services to third-party systems.
Container message with nonrowset-based message parts.	Exposing PeopleSoft services to third-party systems and need to provide nested parts.
Document message.	Integrations with third-party systems.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Naming Conventions for Message Metadata

When naming the following message metadata, names cannot start with "xml," digits or special characters:

- Message names.
- Message aliases.
- Record aliases.
- Field aliases.

Message Record Structure

If a message handles PeopleSoft record data, that is, a rowset-based message, you must insert records in the message definition in an appropriate hierarchical structure.

However, if the message data doesn't map to a record hierarchy, do not insert any records. You supply or receive the data and its structure from an external source, using the PeopleCode XmlDoc or SoapDoc classes.

See Chapter 7, "Sending and Receiving Messages," page 127.

Underlying Record Definitions

Records that you insert in a message definition have live references to the original record definitions. Any change that you make to an underlying record definition is automatically reflected by a change in the corresponding record in the message definition.

Fields Defined as Uppercase

If a message definition includes character fields that are defined as uppercase, then character data in those fields is automatically converted to uppercase when the message is received. This happens when the receiving PeopleCode inserts the message data in a rowset, and it overrides any previous changes in the data, including transformation and data translation.

Message Aliases and Message Versions

Message aliases are read-only once you save the message definition. As a result, once you create a message alias for a message definition, any subsequent versions of the message that you create use the original alias.

Restrictions for Modifying Messages

This section lists the conditions under which a message may become restricted and read-only. This list applies to all message types, including rowset-based messages, nonrowset-based messages, container messages, part messages, and subpart messages.

You cannot modify a message if one or more of the following conditions exists:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- The service to which a message is contained in a restricted service.
- The message is used internally by the system. For example, the delivered IB_GENERIC message is read-only and is used internally by the system.
- The message is referenced in the runtime tables.

  To work around this, you must remove any entries in the runtime tables that reference the message.
- The message is used in a service operation where WSDL documents have been generated.
- The message is used in a service operation that has validation enabled.

Searching for Message Definitions

To search for an existing message definition in the system use the Messages - Search page (IB_MSGSEARCH). To access the page select PeopleTools, Integration Broker, Integration Setup, Messages. The following example shows the page:

Messages – Search page

To search for a message definition:

1. Access the Messages - Search page (PeopleTools, Integration Broker, Integration Setup, Messages).

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Search for a message definition.

   You can search for message definition in one of two ways:

   - Enter search criteria in one or more of the following fields, and then click the Search button:

     Type From the dropdown list, select the type of message for which you are searching. The options are:

     - *Container.*
     - *Document.*
     - *Nonrowset.*
     - *Part nonrowset.*
     - *Part rowset.*
     - *Rowset.*

     Message Name Enter the name of the message.

     Version Enter the version of the message.

   - Click the Search button to display all message definitions in the system.

   The results appear in the Search Results grid.

3. Click the name of the message to view.

Adding Message Definitions

This section discusses how to:

- Add rowset, nonrowset and part message definitions.
- Add document message definitions.

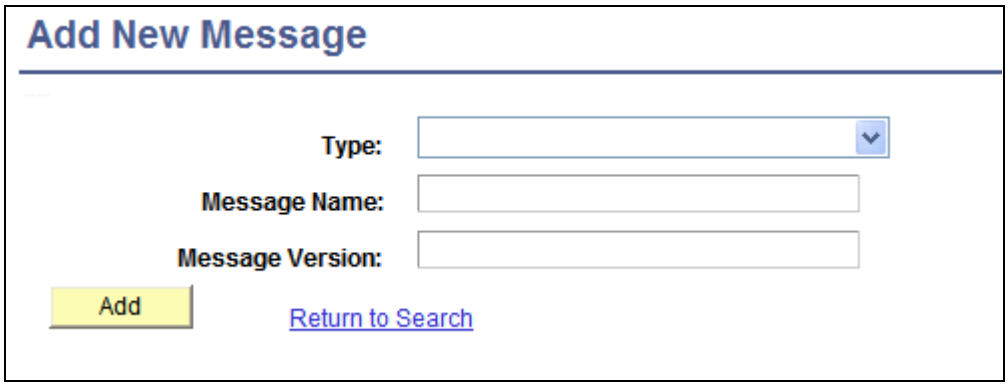
Understanding Adding Message Definitions

When you add a message definition to the system you first give the message a name and specify a message version. After doing so, you can then define additional aspects of the message definition.

Adding Rowset, Nonrowset or Part Message Definitions

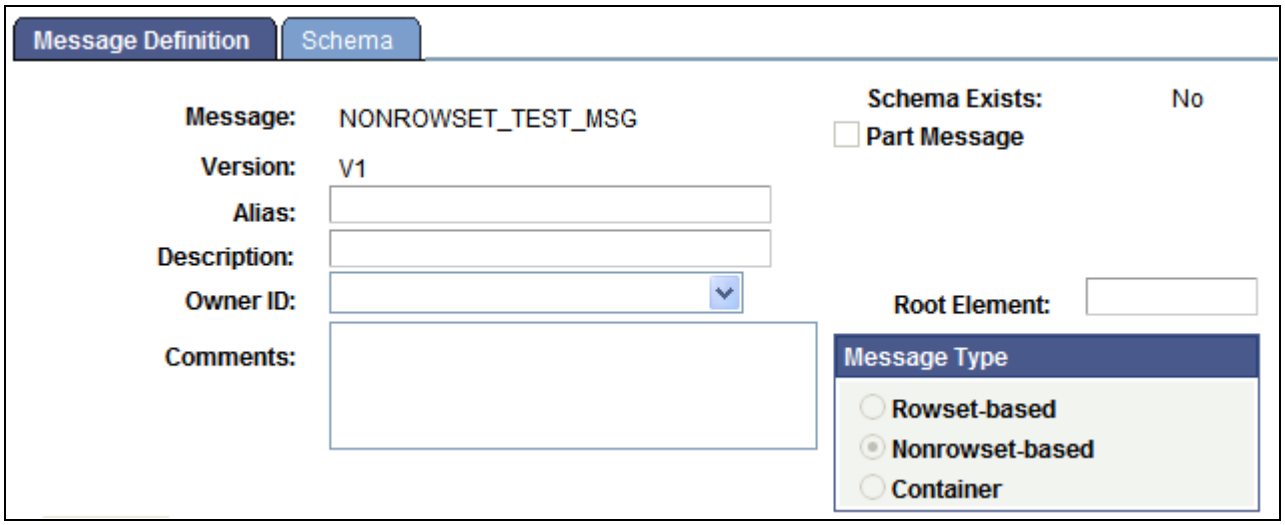
The following example shows the Add New Message page (IB_MSGSEARCH_ADD) that you use to name a new message definition and assign a version to it:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Add New Message page

The following example shows the Message - Message Definition page (IB_MESSAGE_BUILDER) that you use to configure a message after you create the message definition:



Message - Message Definition page when the message type is set to Nonrowset-based.

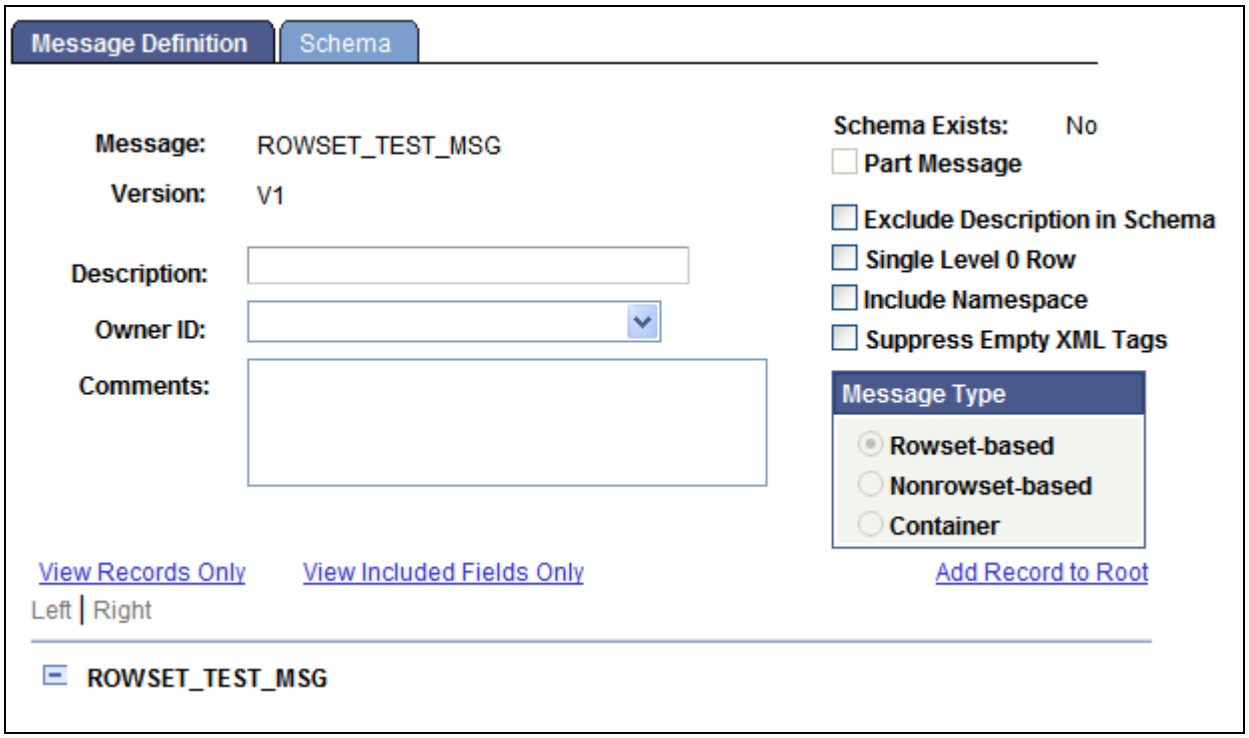
Different options appear on the Message–Message Definition page, depending on the type of message that you are defining.

By default the message type is set to *Nonrowset-based* as shown in the previous example.

If you select a *Rowset-based* or *Container message* type, additional options appear on the page with which you can work.

The following example shows the Message–Message Definition page when you select *Rowset-based* as the message type:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

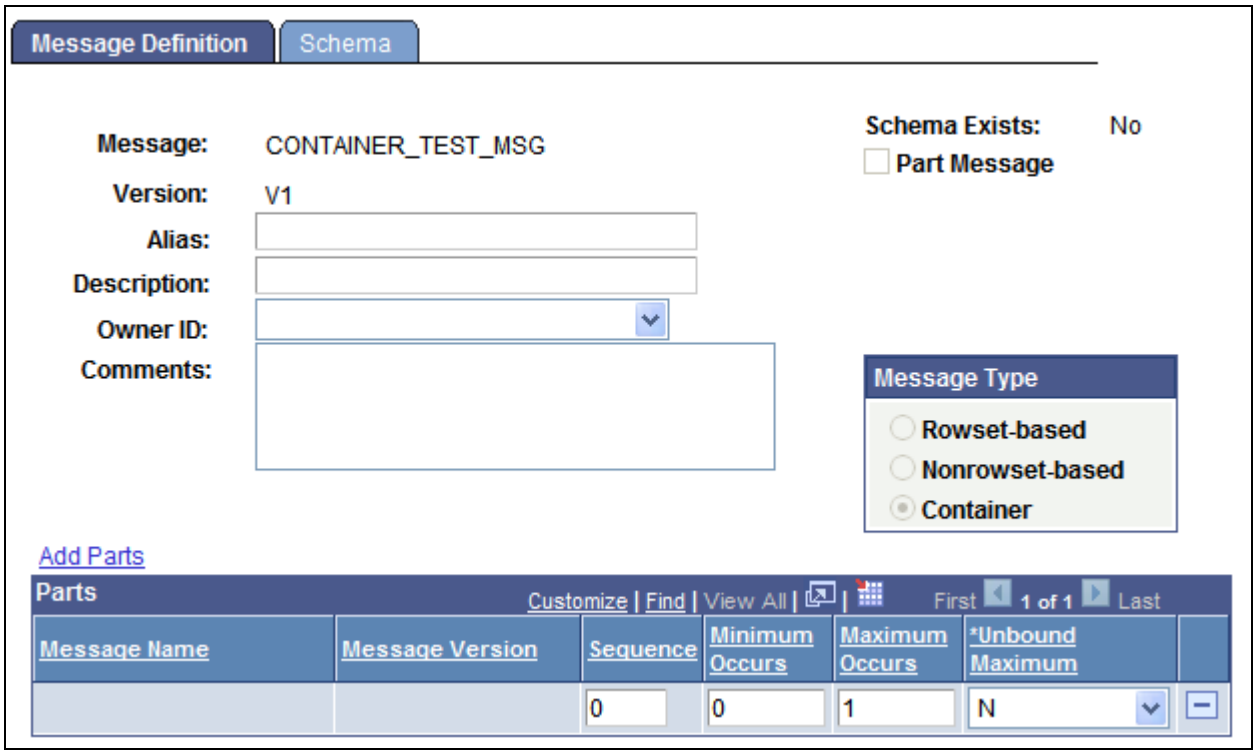


Message - Message Definition page when the message type is set to Rowset-based.

In the previous example, notice the additional options that display on the upper right portion of the page.

When you define a container message, it, too, has its own unique options that you define, as shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Messages - Message Definition page when the message type is set to Container.

Note. For asynchronous integrations, define a single message. For synchronous integrations, define two messages: one request message and one response message, unless the request and response have the same shape.

To add a rowset, nonrowset, or part message definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.

   The Messages – Search page appears.

2. Click the Add a New Valuelink.

3. From the Type drop-down list, select a message type to create. The options are:

   - *Container.* Select this value to add a container message to the system.
   - *Document.* Select this value to add a document message to the system.

     Adding a document message type to the system is described elsewhere in this section.

     See Chapter 6, "Managing Messages," Adding Document Message Definitions, page 84.
   - *Nonrowset.* Select this value to add a nonrowset-based message to the system.
   - *Part Nonrowset.* Select this value to add a nonrowset-based message part to the system.
   - *Part Rowset.* Select this value to add a rowset-based message part to the system.
   - *Rowset.* Select this value to add a rowset-based message to the system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. In the Message Name field, enter a name for the message.

   The message name cannot exceed 30 characters. Do not include any spaces in the message name.

5. In the Version field, enter a version for the message.

   The message version cannot exceed 30 characters. Do not include any spaces in the message version.

   Accepted formats for the message version include:

   - *Version_1*.
   - *V1*.

6. Click the Add button.

   The Messages - Message Definition page appears.

7. (Optional) In the Alias field, enter the name that the external system is expecting, if different from the value in the Message Name field.

   This field appears only when you are defining nonrowset-based or container messages.

8. (Optional) Select the Message Parts check box if the message will be used as a message part in a container message definition.

9. (Optional) In the Description field, enter a description for the definition.

10. (Optional) From the Owner ID drop-down list box, select an owner for the definition.

    The owner ID helps to determine the application team that last made a change to the definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.

11. (Optional) In the Comment field, enter any pertinent comments about the definition.

12. In the Root Element field, enter a value to appear in the root element in generated WADL documents when the message is used in a REST service operation.

    Note. You must provide a value in this field if you are using the message in a REST-based service operation. This field value is required for proper WADL document generation.

    This field appears only when you are defining nonrowset-based messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

13. The next steps to adding a message definition depend on the type of message definition that you are creating:

    - *Rowset-Based Message or Message Part.* You must add a root record to the definition before you can save it.

      See Chapter 6, "Managing Messages," Managing Rowset-Based Messages, page 87.

    - *Nonrowset-Based Message or Message Part.* The message definition is complete and you can click the Save button to save the changes. You can now add an XML message schema to the definition.

      See Chapter 6, "Managing Messages," Managing Nonrowset-Based Messages, page 100.

    - *Container Message.* You must add at least one message part to the definition before you can save the changes.

      See Chapter 6, "Managing Messages," Managing Container Messages, page 109.

Adding Document Message Definitions

This section discusses adding document message definitions.

Understanding Adding Document Message Definitions

When you create a document message definition in the system, you create a message definition that references a document. .

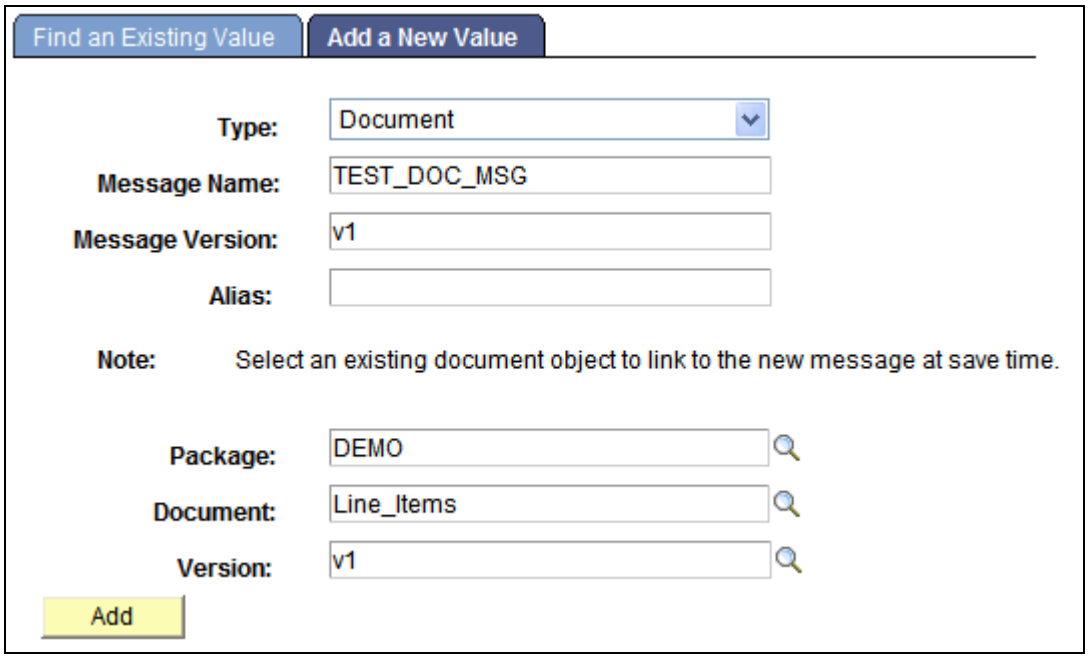
Prerequisites for Adding Document Message Definitions

Before you add a document message definition to the system, the document definition that the message will reference must exist in the system

Adding a Document Message Definition

When you add a document message definition, the following page appears:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

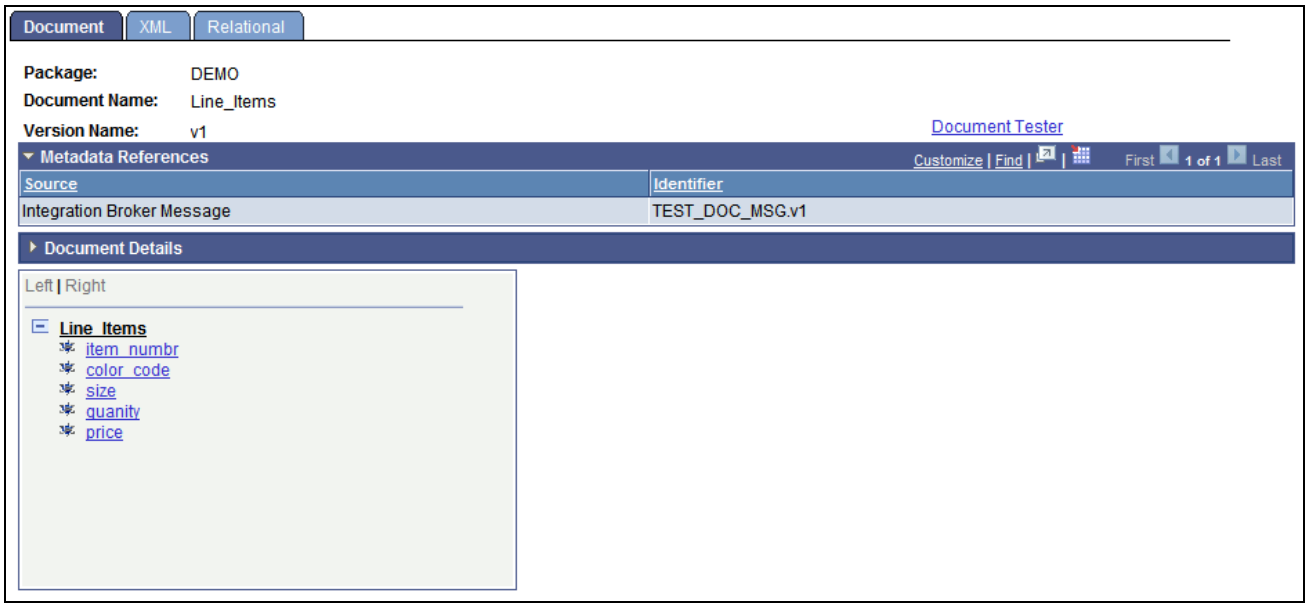


Adding a document message to the system

After you provide a message name, version, and optional alias, you specify the document package, document name, and version to which to link the message.

After you click the Add button, the document that you specified in the message definition appears in the Document Builder – Document page, as shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The Metadata References section shows that the Integration Broker message TEST_DOC_MSG.v1 references the document.

The Metadata References section in the definition shows that there is an Integration Broker message called *TEST_DOC_MSG.v1* that references the document.

Note that the message definition is not saved until you click the Save button in the Document Builder.

To add a document message definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select the Add New Value tab.
3. From the Type drop-down list, select *Document.*
4. In the Message Name field, enter a name for the message.

   The message name cannot exceed 30 characters. Do not include any spaces in the message name.
5. In the Version field, enter a version for the message.

   The message version cannot exceed 30 characters. Do not include any spaces in the message version.

   Accepted formats for the message version include:

   - *Version_1.*
   - *V1.*
6. (Optional) In the Alias field, enter the name that the external system is expecting, if different from the value in the Message Name field.
7. In the Package field, enter the document package or click the Lookup button to search for one.
8. In the Document field, enter the document name or click the Lookup button to search for one.
9. In the Version field, enter the document version or click the Lookup button to search for one.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

10. Click the Add button.

    The Document Builder–Document page appears, displaying the document definition for the document you specified. The Metadata References grid displays the name of the message definition you added.

11. Click the Save button.

Managing Rowset-Based Messages

This section provides an overview of managing rowset-based message definitions and discusses how to:

- View rowset-based message structures.
- Insert root records.
- Insert child and peer records.
- Specify record aliases.
- Delete records.
- Exclude fields from messages.
- Specify field name aliases.
- Managing XML message schemas for rowset-based messages.
- Enforce message record and field aliases in generated WSDL.

Understanding Managing Rowset-Based Messages

This section provides overview information about managing rowset-based message definitions.

Root Records

When you create a rowset-based message, you must at a minimum insert a root record (level 0) into the definition.

Records and Record Fields

You create and modify records and record fields in PeopleSoft Application Designer.

Note. Avoid using derived/work records in messages. Work records don't behave like regular records when used with PeopleCode rowset methods. A good alternative is to use dynamic views.

Record and Record Field Aliases

Record and field aliases are optional parameters that are used for schema and XML generation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When record and field aliases are used, the receiver of a message sees the alias names instead of the actual record and field names. The alias names are seen in the message definition, in message schemas, and on generated runtime XML that is sent to the receiver.

Note that the sender still codes to the actual record and field name.

XML Schema for Rowset-Based Messages

When you create or make changes to a rowset-based message definition, the system automatically generates message schema.

Viewing Rowset-Based Message Structures

This section discusses the three ways to view the structure of rowset-based message definitions. This section discusses how to:

- View the entire structure of rowset-based message definitions.
- View only the records in rowset-based message definitions.
- View only included records fields in rowset-based message definitions.

Viewing the Entire Structure of Rowset-Based Message Definitions

By default, when you open a rowset-based message definition PeopleSoft Integration Broker displays the complete message definition structure. The following graphic shows the complete message definition structure for the message *QE_FLIGHTPLAN*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Complete message structure for the message QE_FLIGHTPLAN

The system displays the definition in a tree structure. Use the plus (+) and minus (-) buttons to expand and collapse the tree to view all records, subrecords and fields (both included and excluded) in the definition.

Record fields that are included in the message definition have a check next to them. Record fields that are not included in the message definition have a box next to them. In the previous graphic, *QE_RANGE* is the only record field that is not included in the *QE_FLIGHTPLAN* message definition.

You can view the record or field properties by clicking the record or field name.

To view the entire structure of a rowset-based message:

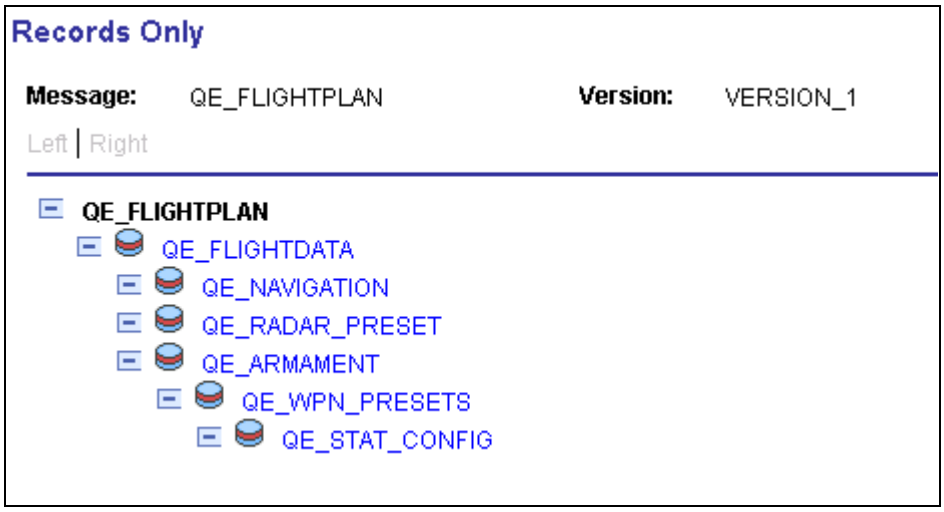
1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select a message to view.

   The Messages-Message Definitions page appears and the entire structure of the message appears in a tree view.
3. Expand and collapse the tree to view the message structure.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Only the Records in Rowset-Based Message Definitions

You can use the Records Only page (IB_MESSAGE_TR_SEC) to view the records and subrecords in a rowset-based message definition. The following graphic shows the Records Only page:



Records and subrecords for the QE_FLIGHTPLAN message displaying in the Records Only page

To view only the records in a rowset-based message:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select a message to view.

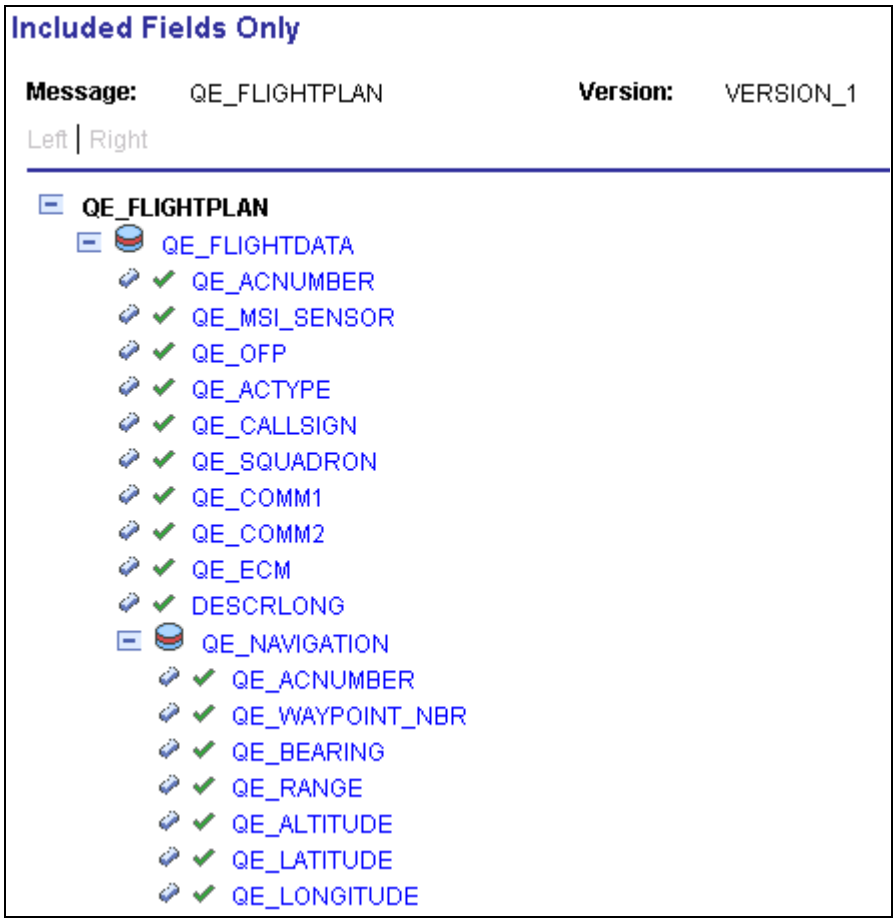
   The Messages-Message Definitions page appears.
3. Just above the tree structure view of the message structure, click the View Records Only link.

   The Records Only page appears and the records and subrecords in the message definition display in a hierarchical view.
4. Click the Return button to return to the Messages-Message Definitions page.

Viewing Only Included Record Fields in Rowset-Based Message Definitions

You can use the Included Fields Only page (IB_MESSAGE_TR_SEC) to view the included records fields for a rowset-based message definition. The following graphic shows a sample of the records and their included fields contained in the *QE_FLIGHTPLAN* message definition:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Records and their included fields for the QE_FLIGHTPLAN message displaying in the Included Fields Only page.

Included fields are denoted by a green icon in the shape of a check mark.

To view included record fields in a rowset-based message:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select a message to view.

   The Messages–Message Definitions page appears.
3. Just above the tree structure view of the message structure, click the View Included Fields Only link.

   The Included Fields Only page appears and included records fields contained in the message display.
4. Click the Return button to return to the Messages-Message Definitions page.

Inserting Root Records

You insert a root record into a rowset-based message definition using the Add New Record page (IB_MESSAGE_TOP_SEC) shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Add New Record page

Note. There can only be one root record defined for each rowset-based message.

To insert a root record into a definition:

1. Access the Add New Record page.

   Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears. Click the Add Record to Root link.

2. In the New Record Name field, enter the name of the record to add, or click the Lookup button to search for and select one.

3. Click the OK button.

The root record appears in the tree structure. Click the plus button to expand the tree and view fields that are associated with the record.

You can exclude fields from the record and specify field name aliases. Steps for performing these actions are described elsewhere in this chapter.

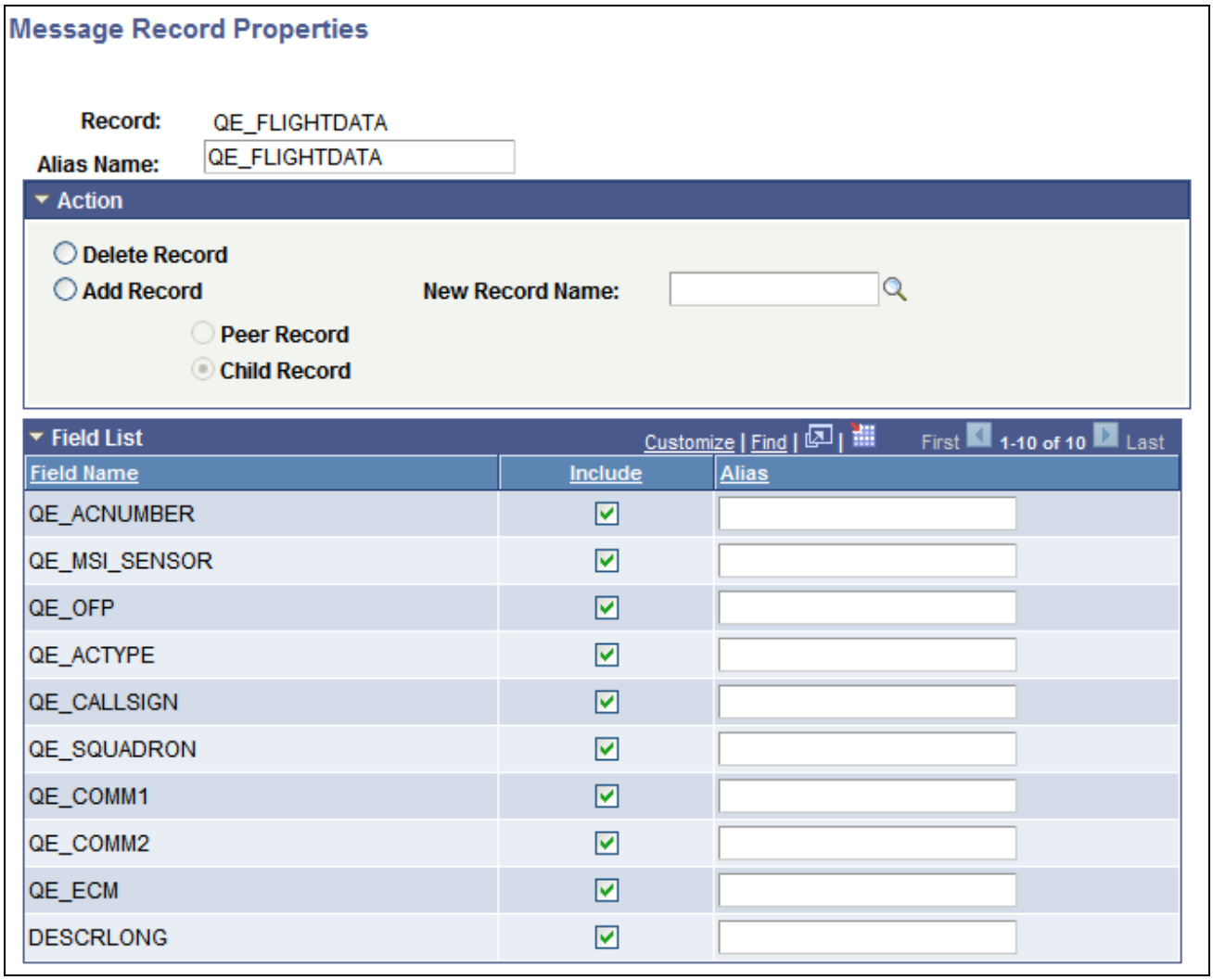
See Chapter 6, "Managing Messages," Excluding Fields from Messages, page 95.

See Chapter 6, "Managing Messages," Specifying Field Name Aliases, page 96.

Inserting Child and Peer Records

You insert child and peer records into a rowset-based message definition using the Message Record Properties page (IB_MESSAGE_REC_SEC) shown in the following example:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Message Record Properties page

To insert a child or peer record into a rowset-based message definition:

1. Access the Message Record Properties page.

   (Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears. Click the linked record name to which to add a peer or child record.)

2. In the Action group box, select Add Record.

3. In the New Record Name field, enter the name of the record to add, or click the Lookup button to search for and select a name.

4. Select whether to add the record as a peer record or a child record.

   - Select Peer Record to add the record as a peer.
   - Select Child Record to add the record as a child.

5. Click the OK button.

   The Messages-Message Definitions page appears.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

6. Click the Save button.

Specifying Record Aliases

You can specify aliases of the root, peer, and child records that you insert into rowset-based messages using the Message Record Properties page.

To specify a record alias:

1. Access the Message Record Properties page.

   (Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears. Click the linked record name to which to specify an alias.)

2. In the Alias Name field, enter an alias name.

3. Click the OK button.

   The Messages-Message Definitions page appears.

4. Click the Save button.

See Also

Chapter 6, "Managing Messages," Message Aliases and Message Versions, page 77

Deleting Records

This section describes how to delete records from a rowset-based message.

Note. Deleting the root record deletes all records and their associated fields that are inserted into the definition.

To delete a record:

1. Access the Message Record Properties page.

   (Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears. Click the name of the record to delete.)

2. In the Action group box, select Delete Record.

3. Click the OK button.

   The Messages-Message Definitions page appears.

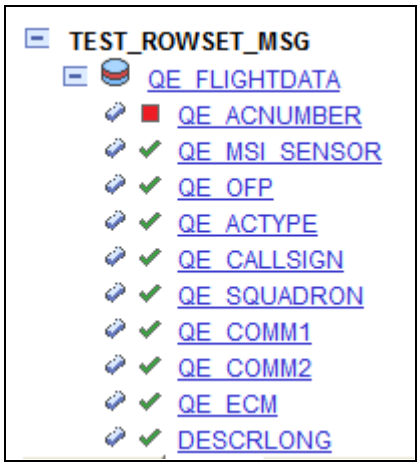
4. Click the Save button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Excluding Fields from Messages

You can exclude record fields from inclusion in a rowset-based message definition using the Message Field Properties page.

After you exclude fields from records, the tree view of the message definition on the Message Definitions page displays a red icon in the shape of box next to the excluded fields. The following example shows that the field *QE_ACNUMBER*, has been excluded from the *QE_FLIGHTDATA* record.



Fields excluded from the QE_FLIGHTDATA record

To exclude fields:

1. Access the Message Field Properties page.
   a. Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears.
   b. Click the plus button (+) to expand the record tree structure, and locate the field to exclude from the definition.
   c. Click the name of the field to exclude.

   The Message Field Properties page appears.
2. Click the name of the field to exclude.
3. Clear the Include check box.
4. Click the OK button.

   The Messages-Message Definitions page appears.
5. Click the Save button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Specifying Field Name Aliases

Use the Message Field Properties page to specify field name aliases.

To specify a field name alias:

1. Access the Message Field Properties page.
   a. Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears.
   b. Click the plus button (+) to expand the record tree structure, and locate the field to exclude from the definition.
   c. Click the name of the field for which to specify a field name alias.

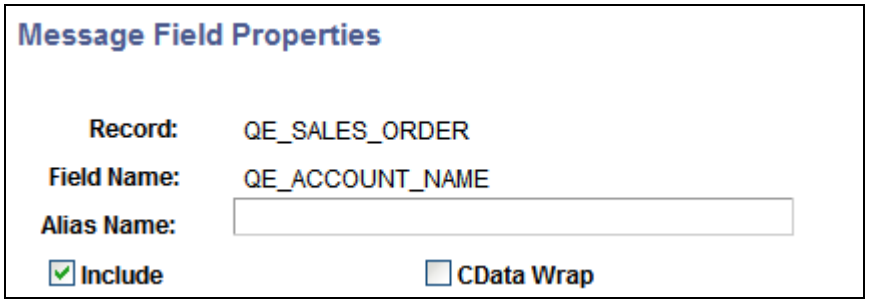
   The Message Field Properties page appears.
2. In the Alias Name field, enter an alias name.
3. Click the OK button.

   The Messages–Message Definitions page appears.
4. Click the Save button.

Including Fields in CData Sections in Generated XML

You can specify that fields be included in CData sections in generated XML. When you select the CData Wrap box on the Message Field Properties page, the field will be wrapped in a CData section in generated XML.

The following example shows the Message Field Properties page and the CData Wrap box:



Check the CData Wrap box to include the field in a CData section in generated XML

To include a field in a CData section in generated XML:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Message Field Properties page.

   a. Select PeopleTools, Integration Broker, Integration Setup, Messages. The Messages-Message Definitions page appears.

   b. Click the plus button (+) to expand the record tree structure, and locate the field to exclude from the definition.

   c. Click the name of the field for which to specify a field name alias.

   The Message Field Properties page appears.

2. Check the CData Wrap box.

3. Click the OK button.

   The Messages–Message Definitions page appears.

4. Click the Save button.

Managing XML Message Schemas for Rowset-Based Messages

This section discusses how to:

- View XML message schemas for rowset-based messages.
- Exclude descriptions in XML message schemas.
- Choose the number of level 0 rows to include in generated XML message schema.
- Include namespaces in generated XML message schemas.
- Suppress empty XML tags in message schema.

Viewing XML Message Schemas for Rowset-Based Messages

PeopleSoft Integration Broker automatically generates message schema for rowset-based messages when you save the message definition.

After you save a message definition on the Messages-Message Definitions page, click the Schema tab to view the generated XML message schema.

Excluding Descriptions in XML Message Schemas

Message data that is used to define services can have actual database record and field names in the generated XML message schema. PeopleSoft Integration Broker provides an option where you can exclude descriptions in generated message schemas so that sensitive information is not exposed.

The Messages–Message Definitions page features an Exclude Descriptions in Schema box that enables you to suppress descriptions in generated schema.

To exclude descriptions in XML message schemas:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Messages–Message Definition page (PeopleTools, Integration Broker, Integration Setup, Messages).
2. Select the Exclude Description in Schema box.
3. Save the changes.

   See Chapter 6, "Managing Messages," Managing XML Message Schemas for Rowset-Based Messages, page 97.

Choosing the Number of Level 0 Rows to Include in Generated XML Message Schema

You can choose to include a single level 0 row in the generated schema or all level 0 rows in the generated schema.

When you select the Single Level 0 Row check box, PeopleSoft Integration Broker includes a single level 0 row in the XML message schema when you Save the definition. If this box is not selected, then the system includes all level 0 rows in the message in the generated schema.

By default the Single Level 0 Row check box is not selected.

If you check the Single Level 0 Row check box to generate schema with one level 0 row, the level 0 row included in the schema is the first level 0 row the system encounters in the message.

Including Namespaces in Generated XML Message Schemas

PeopleSoft Integration Broker enables you to include a namespace in XML message schemas that you generate for rowset-based messages.

When working with a rowset-based message type, the Messages–Message Definition page displays an Include Namespace box. When the Include Namespace check box is selected, you can specify a namespace to include in the generated schema on the Messages-Schema page.

The following example shows the Namespace field as it appears on the Messages–Schema page:

The Messages–Schema page with the default namespace from the Service Configuration page populating the Namespace field.

By default the Namespace field is populated with the namespace defined on the Service Configuration page, however you can change the namespace to use in the message schema as required.

To include a namespace in generated schema:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Messages–Message Definition page (PeopleTools, Integration Broker, Integration Setup, Messages).
2. Check the Include Namespace box.
3. Click the Schema tab.

   The Messages–Schema page appears. By default the namespace as defined on the Service Configuration page populates the Namespace field.
4. In the Namespace field enter the namespace to use in the generated XML message schema.
5. Click the Message Definition tab.
6. Save your changes.

The system generates the message schema and includes the namespace specified.

Suppressing Empty XML Tags in Message Schema

PeopleSoft Integration Broker enables you to suppress empty XML tags in message schema of rowset-based messages.

The Messages-Message Definition page features a Suppress Empty XML Tags check box.

When you select this box, message schema generated for the message will not include any XML tags that contain empty or Null values.

Enforcing Message Record and Field Aliases in Generated WSDL

PeopleSoft Integration Broker enables you to enforce record and field aliases in generated WSDL.

The Service Configuration page features a WSDL Generation Alias Check drop-down list that enables you to set a system check level for aliases on message definition records and fields.

You can set the following check levels:

Check Level	*Description*
Error.	If the system encounters a message definition without proper record and field aliases, it displays an error and it will not generate a WSDL document.
None.	Default. The system creates a WSDL document regardless of whether message records and fields are aliased or not.
Warning.	As the system creates a WSDL document it displays a warning it encounters messages definitions that do not have complete aliasing for records and/or fields. If the system encounters records or fields that do not have aliases defined, you can continue to generate the WSDL document or terminate the generation of the WSDL document.

To enforce message record and field aliases in generated WSDL:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Service Configuration page (PeopleTools, Integration Broker, Configuration, Service Configuration).
2. From the WSDL Generation Alias Check drop-down list, select a value. The valid options are:
   - *Error.*
   - *None.*
   - *Warning.*

Managing Nonrowset-Based Messages

This section provides an overview of managing nonrowset-based messages and discusses how to:

- Add XML message schemas to nonrowset-based messages.
- Edit nonrowset-based XML message schemas.

Understanding Managing Nonrowset-Based Messages

After you create a nonrowset-based message definition, you do not need to complete any additional configuration steps for the definition, except to add an XML schema. The XML schema describes the data to be sent, and includes the field names, data types, field lengths and so on.

You may add or replace message schemas that are referenced by nonrowset-based messages in runtime tables. However, once you change a message schema for a nonrowset-based message, you must adjust the message for a successful integration.

See Also

Chapter 6, "Managing Messages," Adding Message Definitions, page 79

Adding XML Message Schemas to Nonrowset-Based Messages

To add an XML message schema to nonrowset-based messages:

Note. You cannot regenerate message schemas for messages that are defined as part of a restricted service.

---

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select the nonrowset-based definition to which you want to add an XML schema.

   The Messages - Message Definitions page appears.
3. Click the Schema tab.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Click the Add Schema button.

   The Schema page appears.

5. Add the XML schema to the message.

   You can add the schema to the message in two ways:

   - Click the Upload Schema From File button to browse for and upload a schema that you have already saved to a file.
   - Enter the XML schema in the Schema text box that is provided.

6. Click the Save button.

If you define the message as a message part, you must supply a schema to save the message. All message parts require a schema at save time.

Editing Nonrowset-Based XML Schemas

After an XML message schema is added to a nonrowset-based message, you can edit the schema using the Schema page.

Note. You cannot regenerate message schemas for messages that are defined as part of a restricted service.

To edit nonrowset-based XML message schemas:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select the nonrowset-based definition that contains the schema that you want to edit.

   The Messages - Message Definitions page appears.

3. Click the Schema tab.

   The Schema page appears and displays the existing XML message schema for the definition.

4. Click the Edit Schema button.
5. In the Schema text box, make your changes and additions to the XML schema.
6. Click the Save button.

Deleting Nonrowset-Based XML Message Schemas

This section discusses how to:

- Delete individual nonrowset-based XML message schemas.
- Delete nonrowset-based XML message schema in bulk.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Deleting Individual Nonrowset-Based XML Message Schemas

Use the Messages-Schema page (IB_MESSAGE_BUILD2) to delete individual nonrowset-based XML message schema.

To delete an individual nonrowset-based XML message schema:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.

   The Messages-Message Definitions page appears.

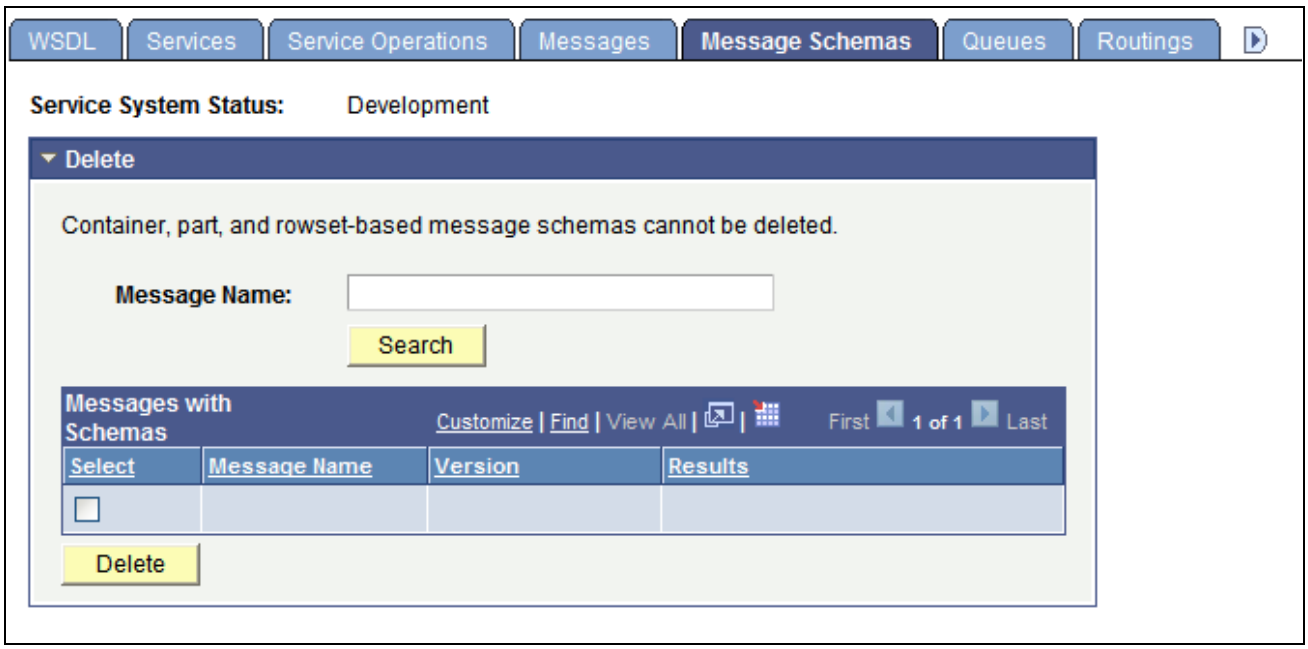
2. Click the Schema tab.

   The Messages-Schema page appears.

3. Click the Delete Schema button.

Deleting Nonrowset-Based XML Message Schemas in Bulk

To delete one or more nonrowset-based XML message schemas, use the Message Schemas page (IB_HOME_PAGE6) in the Service Administration component (IB_HOME_PAGE). The following example shows the Message Schemas page:



Service Administration–Message Schemas page

To delete nonrowset-based XML message schemas in bulk:

1. Select PeopleTools, Integration Broker, Service Utilities, Service Administration.
2. Click the Message Schemas tab.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Choose the schema or schemas to delete.

   To delete an individual schema, in the Message Name field enter the name of the message that contains the schema to delete.

   To delete more than one schema, click the Search button to display all nonrowset-based message in the system than contain schema.

   The message or messages appear in the Messages with Schema grid.

4. In the Select column, check the box next to each message name that contain schema you want to delete.

   If deleting multiple schemas, use the forward and backward arrows and/or the Last and First links to page through the results and select schemas to delete.

5. Click the Delete button.

Managing Message Parts

This section discusses how to create message parts.

Understanding Message Parts

Message parts are individual message definitions that get used in container messages.

While message parts can be rowset-based or nonrowset-based, the advantage of using message parts comes when working with rowset-based messages. By using nonrowset-based message parts, you cannot take advantage of PeopleSoft Integration Broker's framework for creating message definitions, use of PeopleCode, serialization, porting, and so on. The following table highlights some of the advantages of using rowset-based message parts:

Rowset-Based Message Parts	***Nonrowset-Based Message Parts***
You can use the PeopleSoft Pure Internet Architecture to build rowset-based message parts.	You cannot use the PeopleSoft Pure Internet Architecture to build nonrowset-based message parts.
Message schema is automatically generated for rowset-based messages.	You must generate message schema for nonrowset-based message parts.
The mapping from XML to rowset is managed by the framework.	You must use the XMLDoc class to manipulate nonrowset-based message content. In addition, you must manually map the XML into XMLDoc for the parts.

Container messages are always nonrowset-based. So, if you use a container message that contains rowset-based part messages, the container messages sends XML that contains none of the standard PeopleSoft message XML structures, such as PSCAMA, FieldTypes, and so on. However, you can use the rowset-based classes and methods to populate and read the structure of each part message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Creating Part Messages

To create a part message, create a standard rowset-based or nonrowset-based message and click the Part Message box on the Message Definition page.

When the service system status is set to *Production,* once a message is used in a container message, you cannot alter the message while it is associated with a container message.

You must generate schemas for all part messages before you can save them.

Schemas for rowset-based messages are automatically built when the message is saved. Schemas for nonrowset-based parts must be added in order to save the message.

See Also

Chapter 6, "Managing Messages," Adding Message Definitions, page 79

Chapter 6, "Managing Messages," Managing Container Messages, page 109

Distinguishing Blank from Zero in Rowset-Based Part Messages

The Message Definitions page features a Message Part Default Indicator field that appears when you select or define a rowset-based message part.

When you check the box, XML that has a value of *0* (zero) passed in an integer field, when serialized to a rowset, causes the IsChanged property flag on the field to set to *True.*

By default an integer field has a value of *0.* So if a *0* or <*blank*> is passed in a field, the end result is a 0 when accessing the field via the rowset. However, if you check the Message Part Default Indicator box the IsChanged property on such a field is set to *True,* meaning that a *0* (zero) was passed in the field.

Reusing Rowset-Based Message Parts

This section discusses how to:

- Reuse rowset-based message parts by reference.
- Reuse rowset-based message parts by copy.

Understanding Reusing Rowset-Based Message Parts

PeopleSoft Integration Broker enables you to reuse rowset-based message parts by referencing another message part or by copying another message part.

Note. You cannot reuse message parts at Level 0.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Referencing Message Parts

A reference to a message part is read-only in the message part where it is referenced. To make changes to a referenced message part, you must make the changes to the referenced message part directly. All changes are then propagated to every message in which the message part is referenced.

Copying Message Parts

If you copy a message part, the system copies all records and fields and displays them at the record level. The records and fields become permanent to the new message and you can edit all records and fields directly in the message where the copied part exists. Changes you make to a copied message part are not propagated to other copies of the message part that may exist. You must make changes to a copied message part, you do so manually to each message part that you want to change.

Reusing Rowset-Based Message Parts by Reference

This section discusses how to:

- Reuse a message part by reference.
- Check for recursion.
- View referenced message part information.
- View where message parts are referenced.
- Modify referenced message parts.
- Delete referenced message parts

Reusing a Message Part by Reference

To reuse a message part by reference:

1. Create a rowset-based message part.
2. Add records to the message part per your requirements. At a minimum, you must add a Level 0 record.
3. In the tree view of the message part definition, click the name of the record off of which to add the reused message part.

   The Message Record Properties page appears.
4. In the Action box, click Add Part Reference.
5. Identify if the message part is a peer part reference or a child part reference.

   If you are working off the Level 0 record, these fields are read only and Child Part Reference is selected by default.
6. In the Reference Message Version field, click the Lookup button to select the message that the system should reference.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

7. Click the OK button.

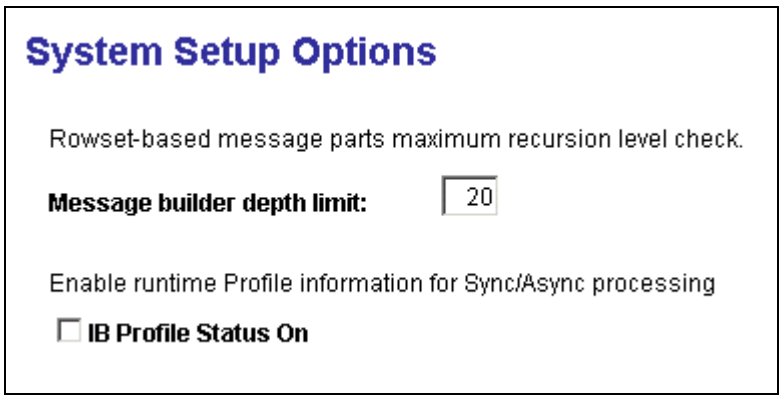
   The Messages-Message Definition page appears.

The reference part is identifiable in the tree view for the message part definition by the highlighted color on the root record of the referenced part. Since this is a reference, you can only view the reference part data structure. To make any modifications to the referenced part, you must open the message part directly and make your changes there. The system will propagate the changes to all messages that reference the message part.

Checking for Recursion

By default, the system checks up to 20 levels for recursion to ensure that no message part references itself. You can modify this setting to check for recursion in as few as three levels of records and as many as 50 levels.

This parameter is set on the System Setup Options page (IB_SYSTEMSETUP). The following example shows the page:



System Setup Options page used to set the level of recursion checking for referenced message parts

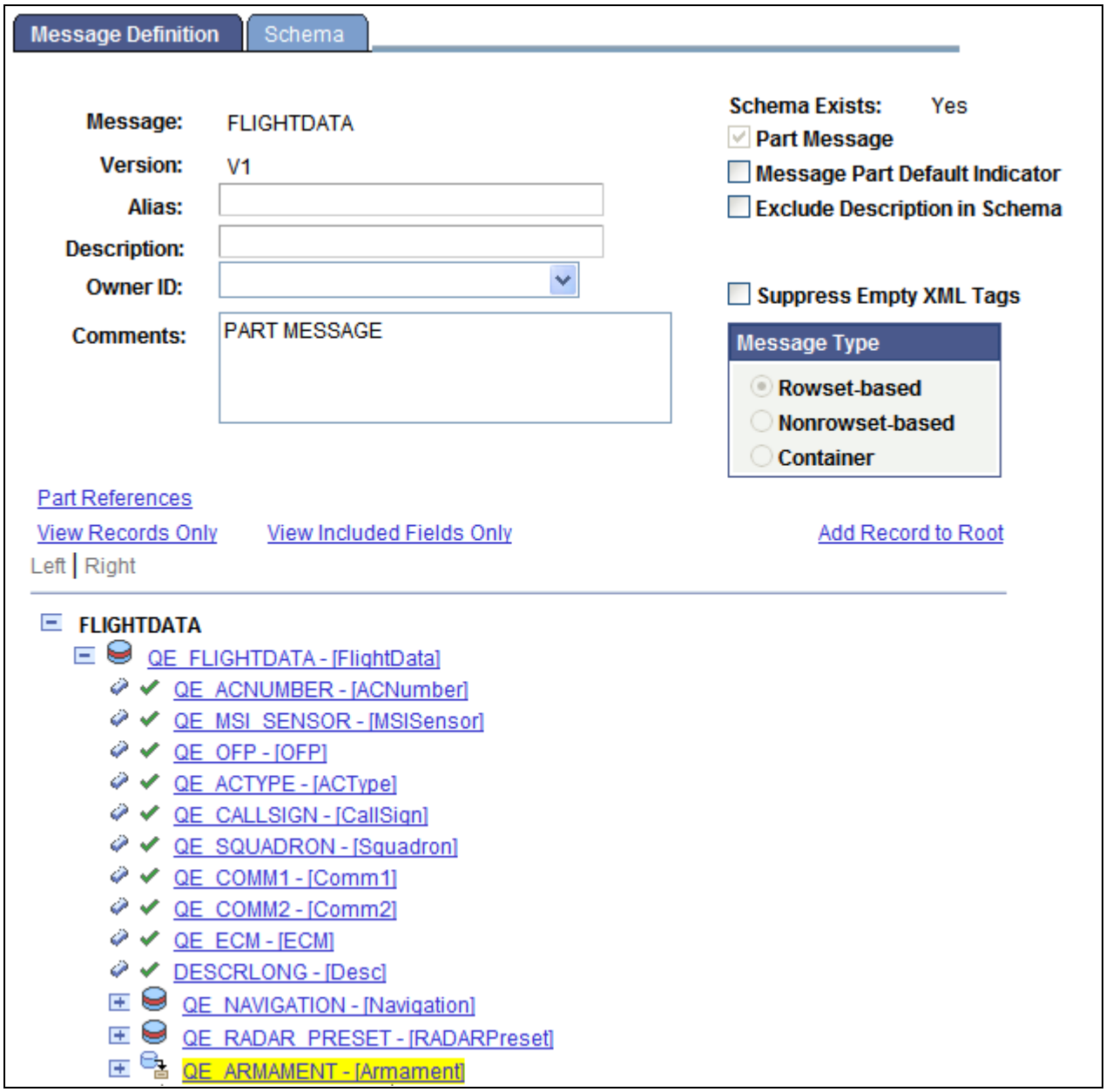
To modify the recursion checking level:

1. Access the System Setup Options page (select PeopleTools, Integration Broker, Configuration, System Setup Options)
2. In the Message Builder Depth Limit field, enter a value between *3* and *50*.
3. Click the Save button.

Viewing Referenced Message Part Information

A referenced message part appears highlighted in the tree structure for a message. The following example shows that the message record *QE_ARMAMENT* is a referenced message part in the message *FLIGHTDATA*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

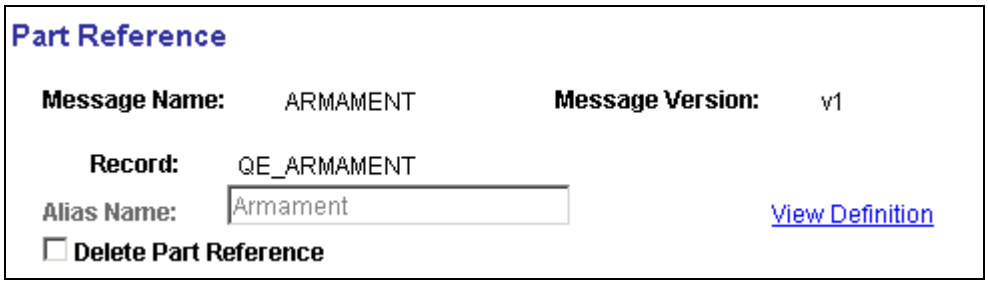


In this example, QE_ARMAMENT is highlighted and is therefore a referenced message part in the FLIGHTDATA message.

Note. You can make changes to a message part that is referenced in another part or subpart, as long as the message part is not in the runtime tables, has not been exported as WSDL, or is a restricted message.

If you click a referenced message part, the Part Reference page (IB_MESSAGE_PARTS2) appears, as shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



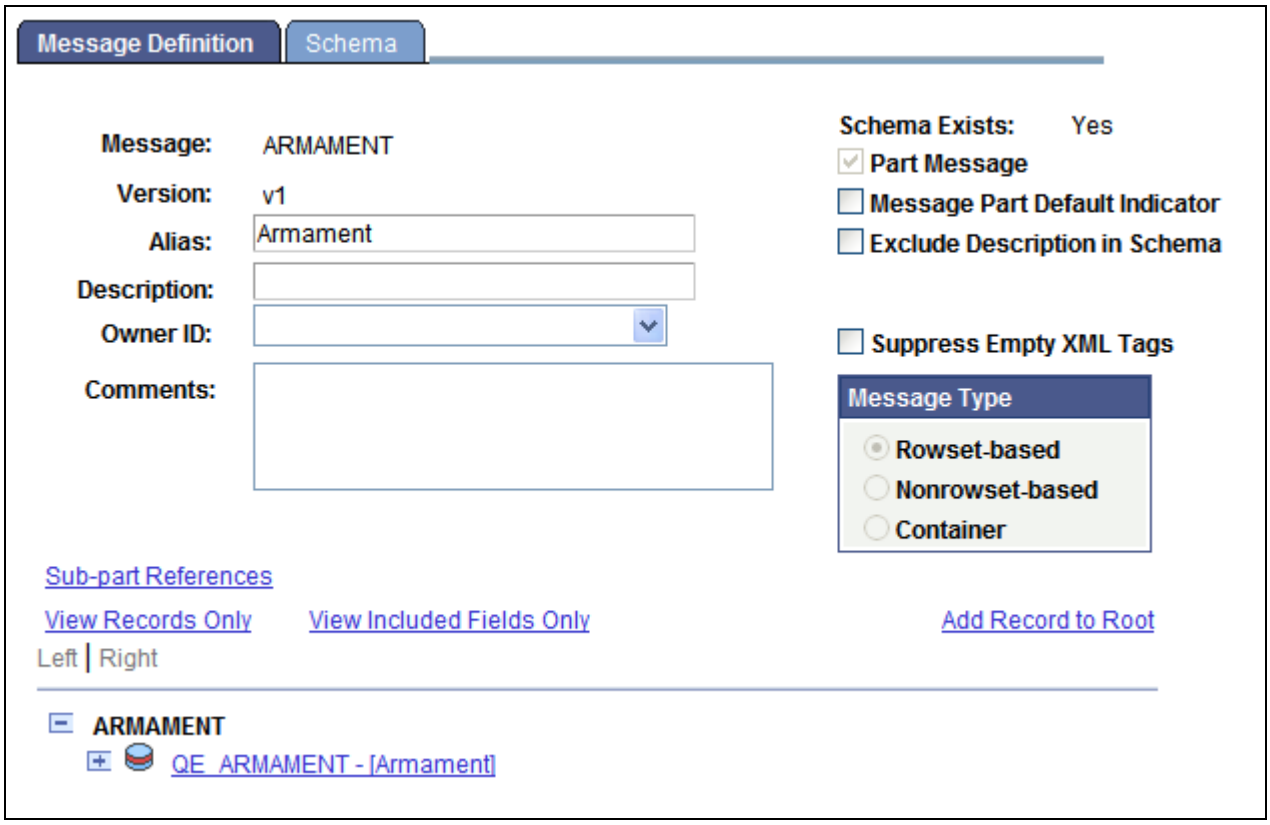
Part Reference page

You can use the Part Reference page to view general information about the referenced message part as well as view the complete definition for the message part.

You can also use this page to delete the reference to the message part. Deleting a part reference is discussed elsewhere in this section.

See Chapter 6, "Managing Messages," Deleting Referenced Message Parts, page 109.

To view the complete message definition for a referenced message part, on the Part References page click the View Definition link. The Messages-Message Definitions page for the referenced message part appears, like the one shown in the following example:



Message definition for the Armament message part.

You can use the page to view details about the record structure, view the generated message schema, and so on.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Modifying Referenced Message Parts

To make a modification to a referenced message part, you must make the modification in the message part definition itself. You cannot modify a referenced message part from a message in which it is referenced.

Deleting Referenced Message Parts

You delete a referenced message part in the message where the part is referenced.

To delete a referenced message part:

1. Open the message definition that contains the referenced message part to delete.
2. In the tree structure view of the message definition, click the name of the referenced message part to delete.

   The Part Reference page appears.
3. Check the Delete Part Reference box.
4. Click the OK button.

Managing Container Messages

This section provides an overview of managing container messages and discusses how to:

- Add message parts to container messages.
- Add and get container message attributes.
- Generate XML message schemas for container messages.

Understanding Managing Container Messages

Container messages are used for those situations where you want to produce XML that contains none of the standard PeopleSoft messaging XML structures, such as PSCAMA, FieldType, and so on, yet you want to use PeopleSoft rowset-based classes and methods to populate and read the message structure.

Container messages:

- Hold one or more message parts.
- Are always nonrowset-based messages.

The message parts you add to a container message must all be rowset-based message parts, or all nonrowset-based message parts.

When working with container messages that contain rowset-based message, PeopleSoft Integration Broker enables you to add attributes and attribute values to the container messages. Adding attributes to container messages enables you to provide integration partners with data and information, without the need to modify or provide the information in the container message definition or in any of the part message definitions.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Including Level 0 Rows for Message Parts in Container Messages

When you are working with a container message that holds rowset-based message parts, you can specify the minimum and the maximum number of level 0 rows for each message part.

When you are working with a container message, the Message Definition page, the Parts grid displays the following fields:

Minimum Occurs The value you enter determines the minimum number of level 0 rows in the message part to include in the container message.

Maximum Occurs The value you enter in this field determines the maximum number of level 0 rows in the message part to include in the container message.

By default the Maximum Occurs value is set to 1 to represent the single row of data on the level 0 record defined on the part (typical for component processing). However, for the case where more then one row of data is to be passed on the level 0 record, for example there is a single record defined on the message part and you want to send x number of rows of data, then increase the Maximum Occurs value to the value of x (the number of rows of data you are sending) or set the Unbounded Maximum field to Y.

Maximum Unbounded The value you select determines if the system includes unlimited level 0 rows from the message part in the container message. The valid values are:

- *Y.* The number of level 0 rows from the part message that the system includes in the container messages is unlimited, or unbound. When you select this option all rows from a part message are included in the container message.
- *N.* (Default) The number of level 0 rows from the part message that the system includes in the container message is limited. You must enter the maximum number of rows from the part message to include in the container message in the Maximum Occurs field.

Example: Message XML when Maximum Occurs is Set to a Non-Default Value

The section contains a example of a container message with three message parts: *QE_PART_1*,*QE_PART_2*, and *QE_PART_3*.

Each part contains only one record (level 0 record).

As described earlier in this section, the Maximum Occurs value is *1* by default.

In the following example *QE_PART_1* is defined on the container with a Maximum Occurs value of *2* and what is actually published in this case is two rows on the level 0 record for *QE_PART_1*, as shown in the example.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

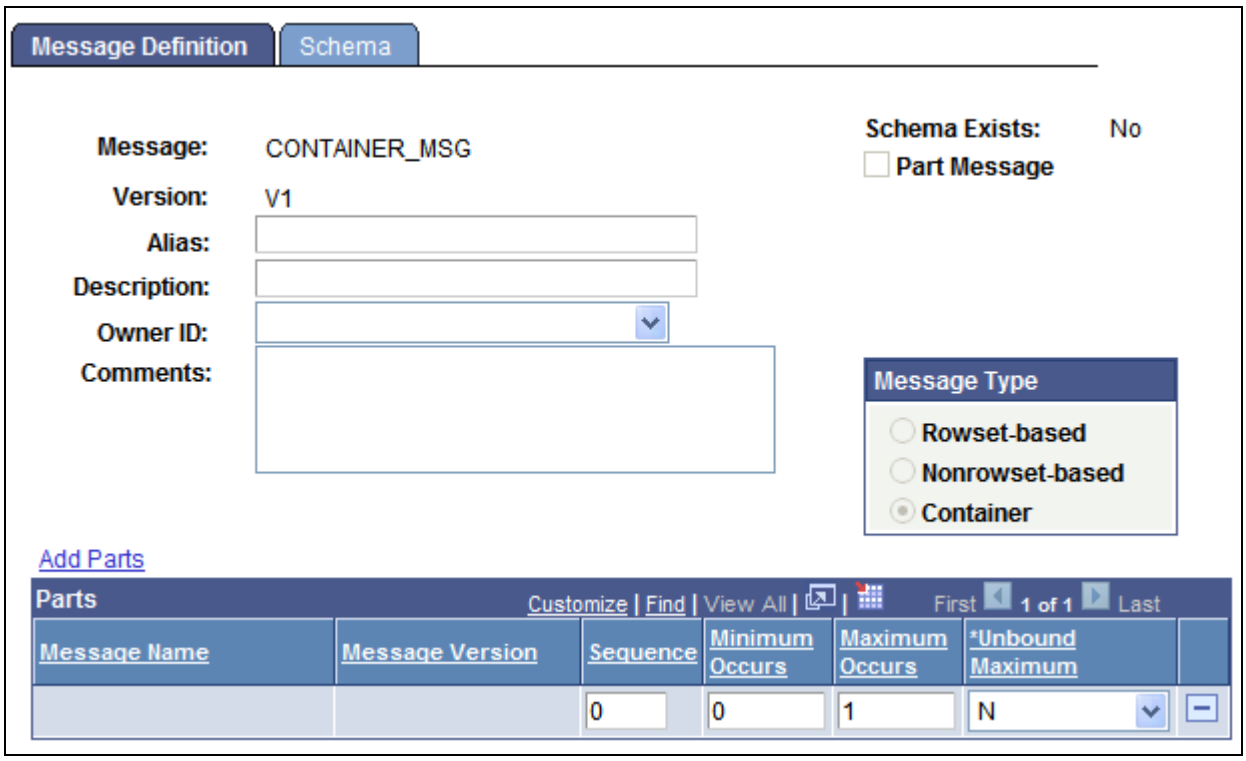
```
<?xml version="1.0"?>
<QE_PARTS xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
  QE_PARTS.VERSION_1">
  <QE_PART_1>
    <QE_NAVIGATION class="R" xmlns="http://xmlns.oracle.com/Enterprise/
      Tools/schemas/QE_PART_1.VERSION_1">
      <QE_ACNUMBER>100</QE_ACNUMBER>
      <QE_WAYPOINT_NBR>10</QE_WAYPOINT_NBR>
      <QE_BEARING/>
      <QE_RANGE/>
      <QE_ALTITUDE/>
      <QE_LATITUDE/>
      <QE_LONGITUDE/>
      <QE_HEADING/>
      <QE_VELOCITIES/>
      <QE_NAVDESC/>
    </QE_NAVIGATION>
  </QE_PART_1>
  <QE_PART_1>
    <QE_NAVIGATION class="R" xmlns="http://xmlns.oracle.com/Enterprise/
      Tools/schemas/QE_PART_1.VERSION_1">
      <QE_ACNUMBER>100</QE_ACNUMBER>
      <QE_WAYPOINT_NBR>20</QE_WAYPOINT_NBR>
      <QE_BEARING/>
      <QE_RANGE/>
      <QE_ALTITUDE/>
      <QE_LATITUDE/>
      <QE_LONGITUDE/>
      <QE_HEADING/>
      <QE_VELOCITIES/>
      <QE_NAVDESC/>
    </QE_NAVIGATION>
  </QE_PART_1>
  <QE_PART_2>
    <QE_RADAR_PRESET class="R" xmlns="http://xmlns.oracle.com/Enterprise/
      Tools/schemas/QE_PART_2.VERSION_1">
      <QE_ACNUMBER>2</QE_ACNUMBER>
      <QE_RADAR_SELECTION>1</QE_RADAR_SELECTION>
      <QE_RADARMODE>TWS</QE_RADARMODE>
      <QE_RADAR_OPERMODE>N</QE_RADAR_OPERMODE>
      <QE_BARSCAN>4B</QE_BARSCAN>
      <QE_RADARRANGE>40</QE_RADARRANGE>
      <QE_TGTAGE>8</QE_TGTAGE>
      <QE_CHANNELSET>B</QE_CHANNELSET>
      <QE_AZIMUTH>80</QE_AZIMUTH>
      <QE_PRF>H</QE_PRF>
    </QE_RADAR_PRESET>
  </QE_PART_2>
  <QE_PART_3>
    <QE_ARMAMENT class="R" xmlns="http://xmlns.oracle.com/Enterprise/Tools/
      schemas/QE_PART_3.VERSION_1">
      <QE_ACNUMBER>2</QE_ACNUMBER>
      <QE_STATION_NBR>1</QE_STATION_NBR>
      <QE_AGMODE>CCIP</QE_AGMODE>
      <QE_BIT>SBIT</QE_BIT>
      <QE_WEAPONSPECS/>
    </QE_ARMAMENT>
  </QE_PART_3>
</QE_PARTS>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Adding Message Parts to Container Messages

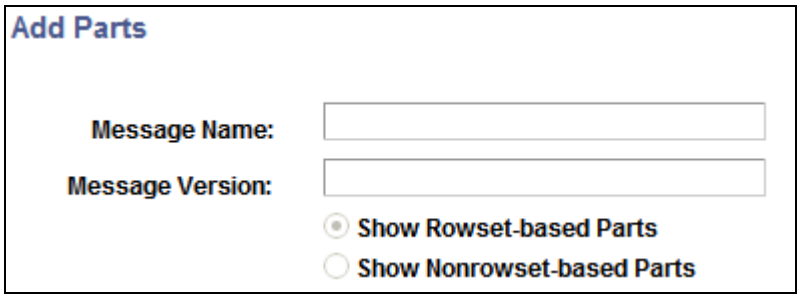
This section discusses how to add message parts to container messages.

Use the Messages - Message Definitions page to add message parts to a container message. To access the page, select PeopleTools, Integration Broker, Integration Setup, Messages. The following example shows this page:



Messages - Message Definitions page for a container message definition

When you click the Add Parts link to specify a message, version, and message type to add, the Add Parts page (IB_MESSAGE_PARTS) appears as shown in the following example:

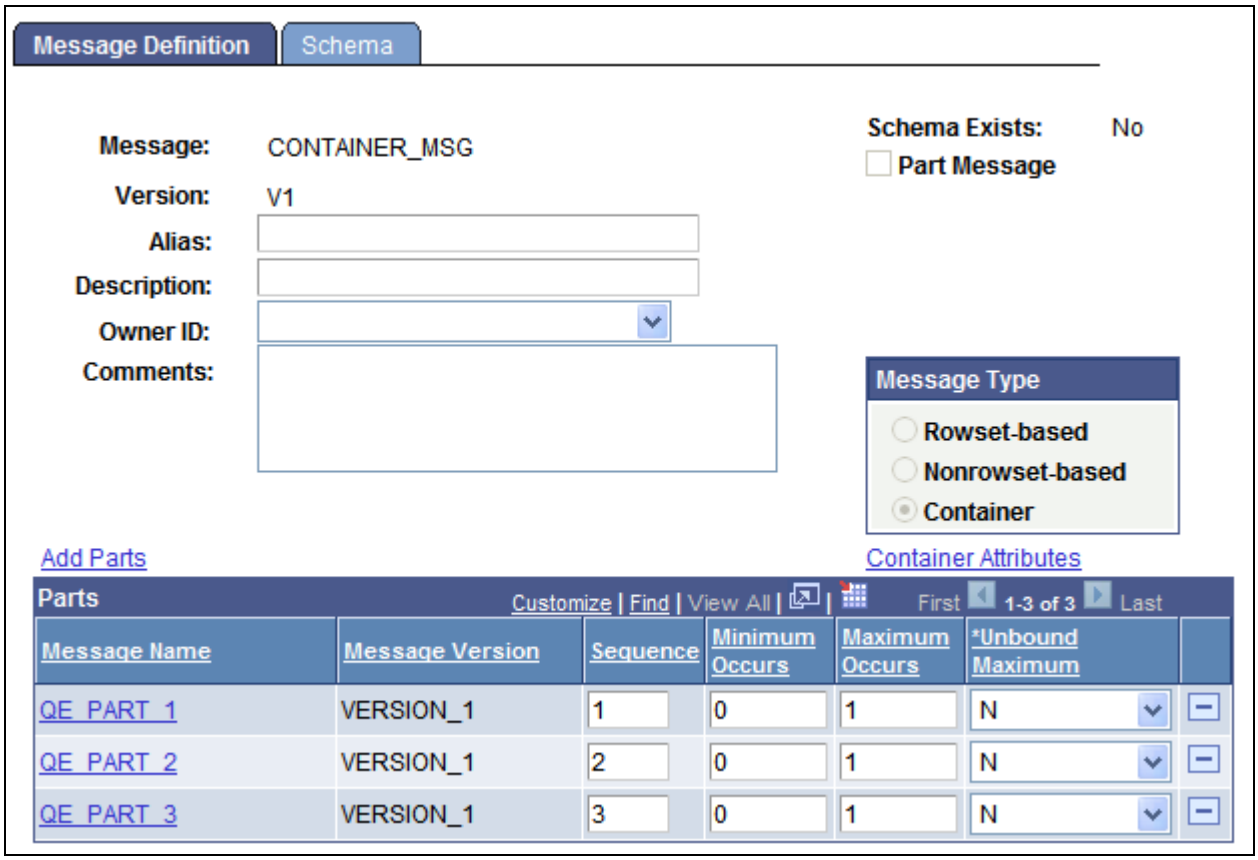


Add Parts page

For a message definition to be available for you to add to a container message, you must have selected the Message Parts check box when you created the message definition. In addition, container messages can contain only all rowset-based messages or all nonrowset-based messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

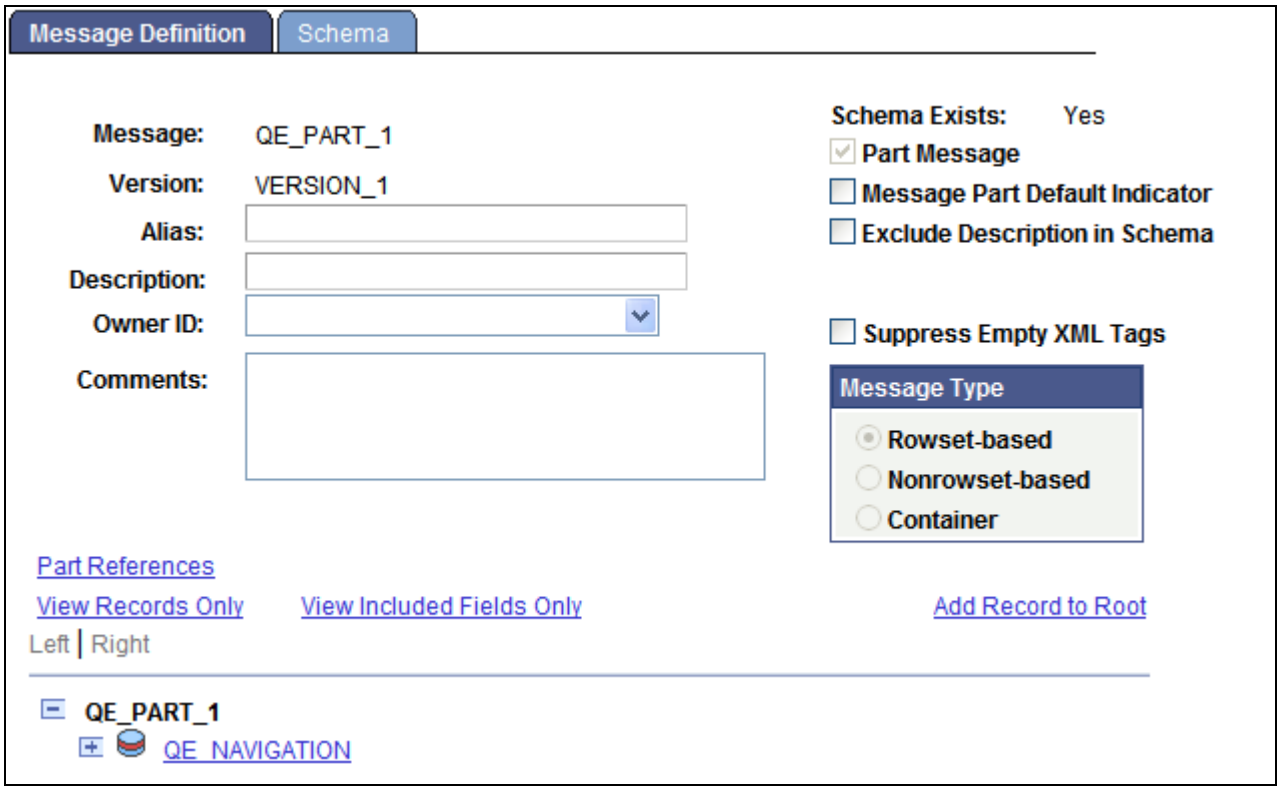
After you add message parts to a container message, the Messages - Message Definitions page displays and the message parts that you have added to the message are listed in the Parts grid. The following examples show of three message parts that are added to a container message:



A container message that contains three message parts

Click the name of any of the message parts that appear in the grid to open the individual message definition. If the service system status is set to *Production*, when assigned to a container message, you cannot modify a message definition. To modify the definition, you must delete it from the container message first. The following example shows how the *PART_1* message part displays if you click the message name in the Parts grid:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The definitions for the part message QE_PART_1

Clicking the Part References link displays all messages to which the message part is assigned.

Note. Before you add nonrowset-based message parts to a container message, you must upload XML message schemas to each message part that you intend to include in the container message. Nonrowset-based part messages cannot be saved without a schema.

To add a message part to a container message:

1. Select PeopleTools, Integration Broker, Integration Setup, Messages.
2. Select a container message to which to add message parts.

   The Messages - Message Definitions page appears.
3. Click the Add Parts link.

   The Add Parts page appears.
4. Select a message to add.

   You can use one of two methods to select a message to add:

   a. In the Message Name and Message Version fields, enter the message name and version to add.

   b. Select the Show Rowset-Based Parts option or the Show Nonrowset-Based Part option and click the Search button to display all rowset-based or nonrowset-based messages that are designated as part messages in the system.

      Select one or more messages to include in the container message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Click the OK button.

   The Messages - Message Definitions page appears, with the Parts grid populated with the message or messages that you selected.

6. (Optional.) In the Parts grid, enter numeric values in the Sequence column to order message part placement in the container message.

   If you do not enter any values, the system sequences the messages in the order in which you add them to the container message.

7. (Optional.) In the Minimum Occurs field, enter the number of minimum rows in the message part to include in the container message.

8. In the Maximum Occurs field, enter the maximum number of level 0 rows from the part message to include in the container message.

9. In the Unbound Maximum drop-down list, select whether to include all level 0 rows from the part in the container message.

   Note. If you select *Y*, note that the Maximum Occurs field no longer displays on the page, as all rows are included in the container message.

The Minimum Occurs, Maximum Occurs and Unbound Maximum fields are described elsewhere in this section.

See Chapter 6, "Managing Messages," Understanding Including Level 0 Rows for Message Parts in Container Messages, page 110.

Adding and Getting Container Messages Attributes

This section discusses how to:

- Add the language code of the message sender as an attribute to a container message.
- Add attribute names to a container message.
- Populate attribute values for container message attributes.
- Get attribute names and values from a container message.

This section also provides a summary of PeopleCode that you can use to populate attribute values and get attribute data from container messages.

Understanding Adding, Populating, and Getting Container Message Attributes

You can add attributes to container messages that contain rowset-based message parts to provide integration partners with data and information, without adding the information to the message definition.

To add attributes to a container message, you first define the attribute name, length, and required flag in the container message definition in the PeopleSoft Pure Internet Architecture. This information appears in generated container message schema. At runtime the attributes appear at the root level of the generated XML. Next you use PeopleCode to populate the attribute values using the IBInfo object. At runtime, PeopleSoft Integration Broker validates the attribute values against the lengths you defined in the container message definition.

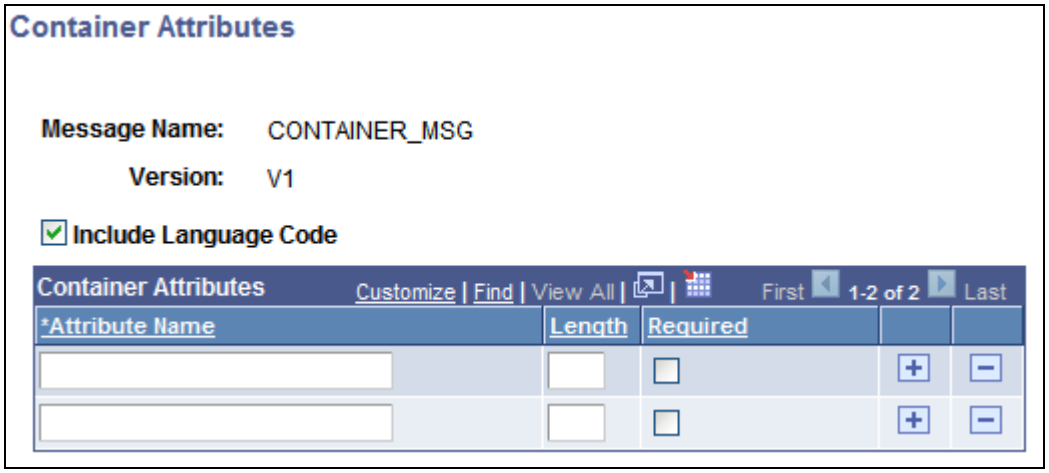
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleSoft provides a number of IBInfo object methods to get attributes from container messages.

Adding Language Codes of the Message Senders as Attributes to Container Messages

The language code of the user who executed the publish or sync request is a common attribute to add to a container message. As such, PeopleSoft provides an Include Language Code attribute box, that when selected automatically includes the information as an attribute name and value in the container message.

The Include Language Code attribute box appears on the Container Attributes page shown in the following example:



Container Attributes page with the Include Language Code box selected.

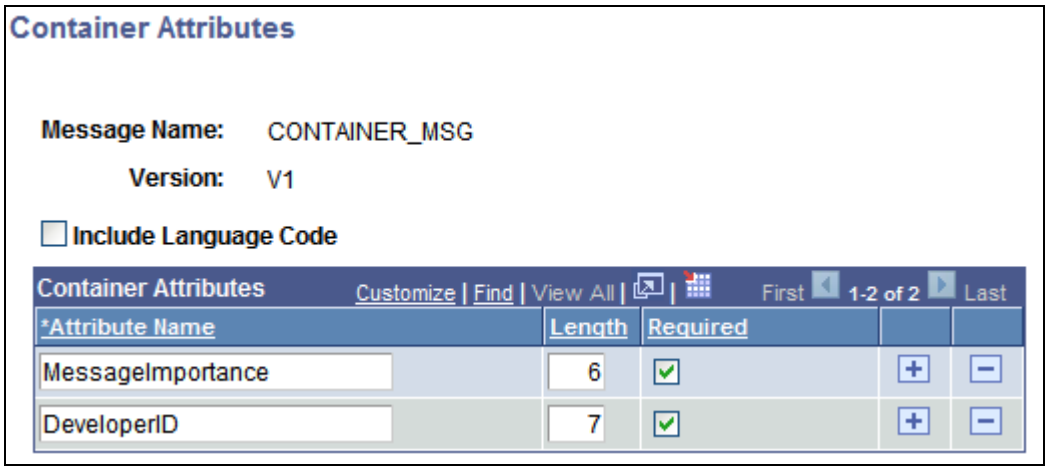
To add the language code of the message sender as an attribute:

1. Access the Container Attributes page (PeopleTools, Integration Broker, Integration Setup, Messages and click the Container Attributes link).
2. Select the Include Language Code box.
3. Click the OK button.
4. The Messages–Message Definitions page appears.

Adding Attribute Names to Container Messages

After you add one or more rowset-based message parts to a container message and save the message, a Container Attributes link appears on the Messages-Message Definition page under the Message Type group box. When you click the Container Attributes link, the Container Attributes page (IB_MESSAGE_ATT_SEC) shown in the following example appears:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Container message with two attribute names defined: MessageImportance, and DeveloperID.

To add an attribute name to a container message:

1. Access the Container Attributes page (PeopleTools, Integration Broker, Integration Setup, Messages and click the Container Attributes link).
2. In the Attribute Name field, enter a name for the attribute.
3. In the Length field, enter a numeric field length value.
4. (Optional.) Check the Required box if you want the attribute name to be required.
5. Click the OK button.

   The Messages–Message Definitions page appears.

Populating Attribute Values for Container Message Attributes

PeopleSoft provides several IBInfo object methods within the Message object to populate container message attributes.

Here is an example of how to populate attributes. The attribute values will be validated at runtime against the defined lengths.

```
&MSG = CreateMessage(Operation.MY_SVC_OPERATION);

&ret = &MSG.IBInfo.AddContainerAttribute("MessageImportance", "Medium");
&ret = &MSG.IBInfo.AddContainerAttribute("DeveloperID", "mdawson");
```

Additional IBInfo objects that you can use for working with container message attributes are described elsewhere in this section.

Getting Attribute Names and Values from Container Messages

PeopleSoft provides several IBInfo object methods within the Message object to Get attribute information from container messages.

Note that if you attempt to read attributes within an Integration Broker event, such as OnNotify, OnRequest, and so on, you must first Get a part rowset to load the attributes into the Message object from the XML.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following code snippet shows one example of how to read attributes from a container message:

```
RowSet = &MSG.GetPartRowset(1);
&index = &MSG.Ibinfo.GetNumberOfContainerAttributes();

For &i = 1 To &index

   &attrName = &MSG.Ibinfo.GetContainerAttributeName(&i);
   &attrValue = &MSG.Ibinfo.GetContainerAttributeValue(&i);

End-For;
```

Additional IBInfo objects that you can use for working with container message attributes are described elsewhere in this section.

Summary of PeopleCode Use for Working With Container Message Attributes

The following table summarizes the PeopleCode methods that you can use for working container message attributes.

Method	***Description***
GetNumberOfContainerAttributes Syntax: `&Integer = &MSG.IBInfo.GetNumberOf⇒ ContainerAttributes();`	Gets the number of container attributes
GetContainerAttributeName Syntax: `&String = &MSG.IBInfo.GetContainer⇒ AttributeName(Integer nIndex);`	Returns the name of the container attribute based on an index.
GetContainerAttributeValue Syntax: `&String = &MSG.IBInfo.GetContainer⇒ AttributeValue(Integer nIndex);`	Returns the value of the container attribute based on an index.
AddContainerAttribute Syntax: `&Bool = &MSG.IBInfo.AddContainer⇒ Attribute(string name, string⇒ value);`	Add container attributes by passing in attribute name and value.
DeleteContainerAttribute Syntax: `&Bool = &MSG.IBInfo.DeleteContainer⇒ Attribute(string name);`	Delete a container attribute based on the attribute name.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Method	*Description*
ClearContainerAttributes Syntax: `&MSG.IBInfo.ClearContainer⇒ Attributes();`	Deletes all container attributes in the IBInfo object.

Generating XML Message Schemas for Container Messages

XML message schemas for container messages re automatically generated when you save the definition. You can view the generated XML message schema on the Messages - Schema page. To access the page, select PeopleTools, Integration Broker, Integration Setup, Messages and click the Schema tab.

The following example shows this page:

System-generated XML message schema for container message with rowset-based message parts

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The namespace that is used in the XML message schema becomes by default the value that is defined on the Service Configuration page. To change the namespace, enter a the new namespace on the Schema page in the Namespace field, the Message Definition tab, and save the change. The XML message schema is generated again by means of the modified namespace value.

Managing Document Messages

After you add a document message to the system, you manage the document using the Document Builder and the document utilities.

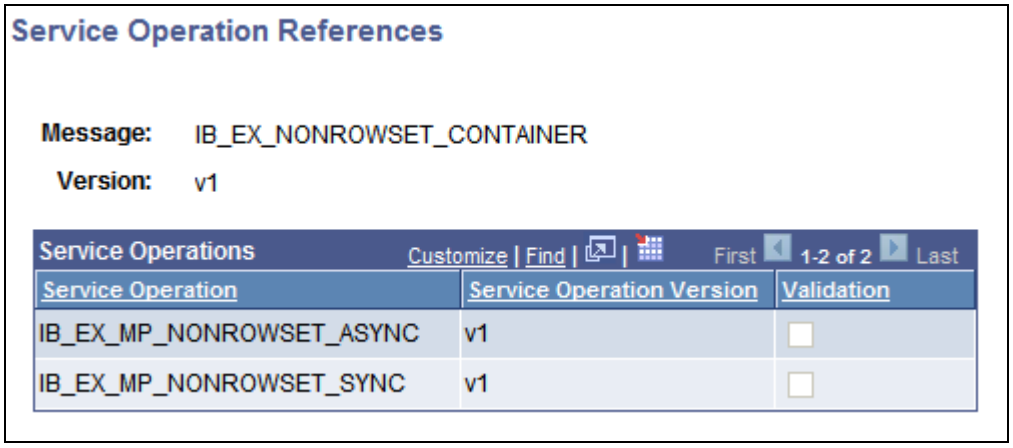
See Also

PeopleTools 8.52: PeopleSoft Documents Technology, "PeopleSoft Documents Technology Preface"

Viewing Service Operations that Reference Messages

Use the Service Operation References page (IB_MESSAGE_SO_SEC) to view a list of service operations that contain a message. The Messages-Message Definitions page provides a link to this page. To access the page, select PeopleTools, Integration Broker, Integration Setup, Messages, and click the Service Operation References link.

The following example shows the Service Operation References page:



Service Operation References page show a list of service operations that contain the message IB_EX_NONROWSET_CONTAINER

The following page elements appear on the Service Operation References page:

Message Name of the message that is referenced in one or more service operations.

Version Version of the message that is reference in one or more service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operation	Name of the service operation that contains the message.
Service Operation Version	Version of the service operation that contains the message.
Validation	When the box is selected message schema has been generated for the message in the service operation.

Resolving Inconsistencies in Exported WSDL and WADL Documents

This section discusses how to:

- View service operations with exported WSDL or WADL inconsistencies.
- Clear exported WSDL/WADL status flags.

Understanding Using Project Copy and Exported WSDL and WADL

When you generate WSDL or WADL for a service operation, the system sets an internal flag on the service operation that indicates that WSDL/WADL has been generated or exported for the specific service operation.

The system uses the same repository for WSDL and WADL documents. The WSDL metadata object is used for project copy of WSDL and WADL. There is no WADL metadata object only WSDL.

You may later decided to use Project Copy to copy the service operation to a new database. But you may decide not to or simply neglect to copy the exported WSDL or WADL to the new database.

Even though you have not copied the WSDL/WADL to the new database, the internal flag that says WSDL/WADL has been generated is still set on the service operation. As a result, the system expects WSDL/WADL to exist in the new database, when it does not. When this condition exists, the system displays a status message on the message definition(s) of messages referenced in the service operation.

When this condition exists, the options are:

- Clear the internal WSDL/WADL exported flag on the service operation.

  Information about how to perform this task is discussed in this section.

- Use Project Copy to copy the WSDL/WADL to the new database.

  See *PeopleTools 8.52: PeopleSoft Application Designer Lifecycle Management Guide*, "Copying Projects and Definitions."

- Regenerate the WSDL/WADL on the new database using the Provide Web Service wizard.

  See Chapter 20, "Providing Services," page 479.

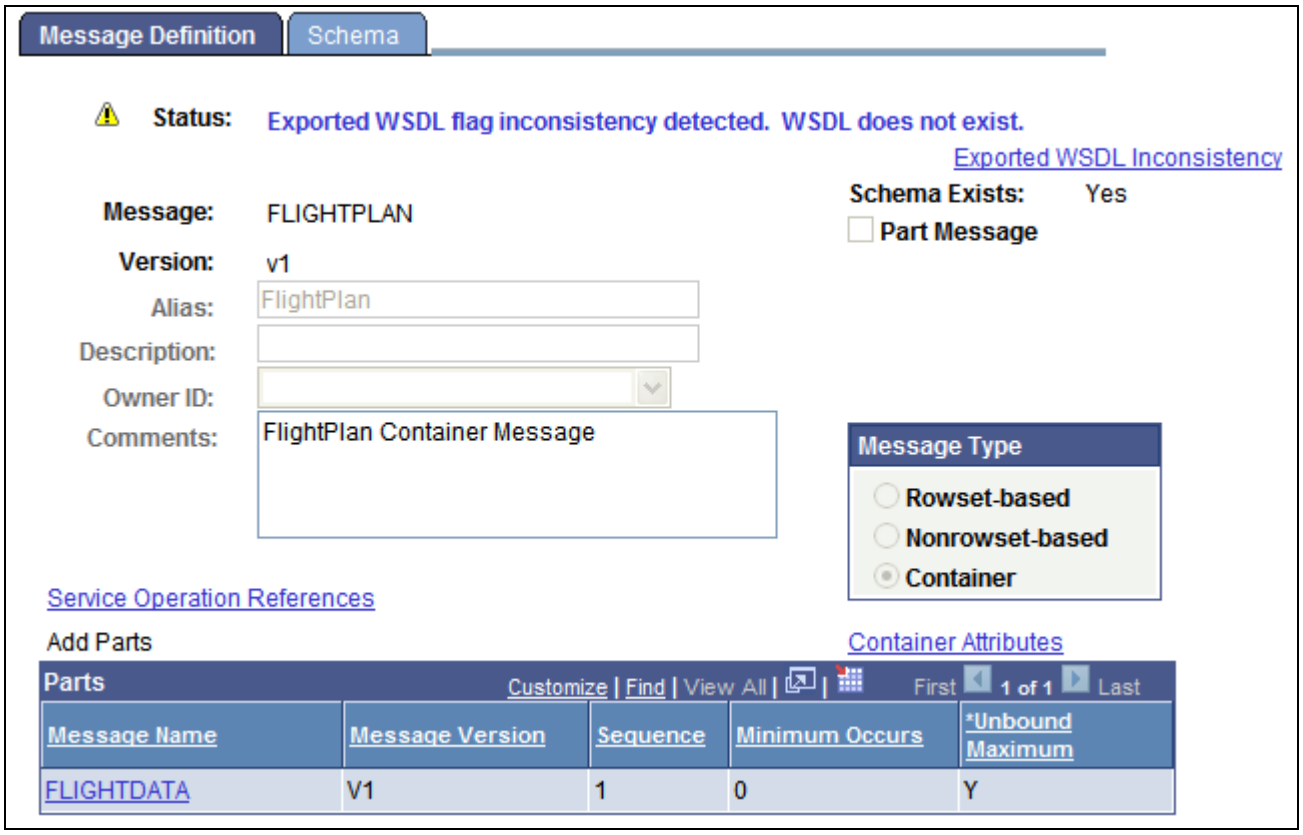
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Services Operations with Exported WSDL/WADL Inconsistencies

If the system detects a WSDL/WADL flag inconsistency, the following status message appears on the Messages-Message Definitions page for those message definitions referenced in the service operation for the WSDL/WADL in question:

```
Exported WSDL flag inconsistency detected. WSDL does not exist.
```

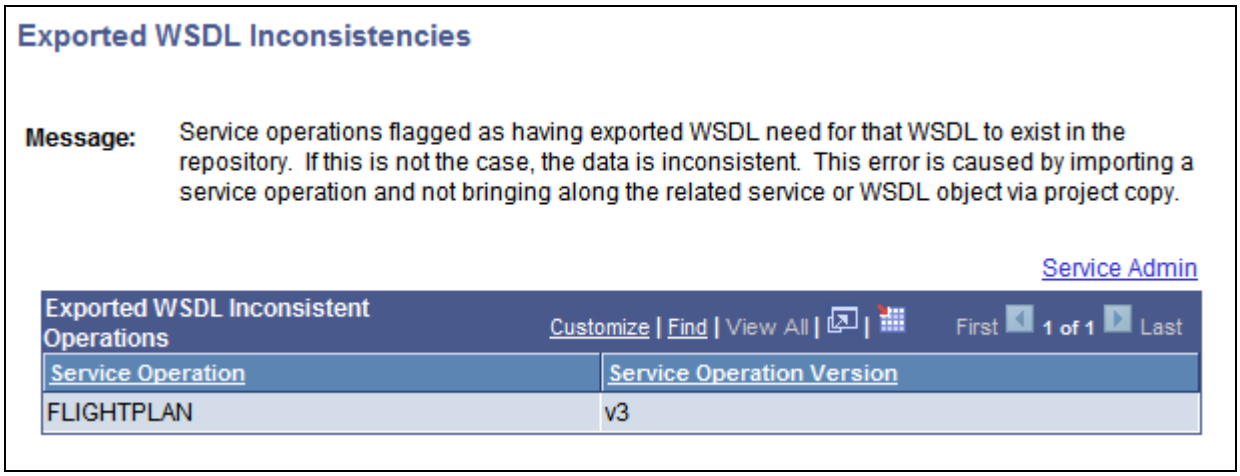
The following graphic shows an example of the Messages-Message Definitions page displaying the status message:



Messages-Message Definitions page showing the "Exported WSDL flag inconsistency detected" status message and the Exported WSDL Inconsistency link.

In addition, an Exported WSDL Inconsistency link appears on the Messages-Message Definitions page. Click this link to view the Exported WSDL Inconsistencies page (IB_HOME_PAGE7_SEC). The following graphic shows an example of the page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Exported WSDL Inconsistencies page

The page displays service operations that exist in the database that are flagged as having WSDL/WADL exported, yet no WSDL/WADL exists in the database for them. The Exported WSDL Inconsistencies page features a Service Admin link. Clicking the link opens the Service Administration-WSDL page (IB_HOME_PAGE7). The Service Administration-WSDL page provides a options to clear internal exported WSDL flag.

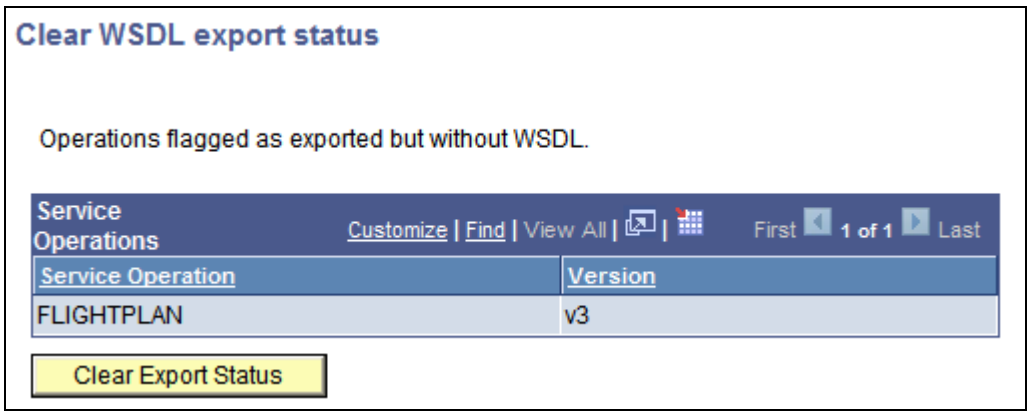
Clearing Exported WSDL/WADL Status Flags

The Clear WSDL Status page (IB_HOME_PAGE7_SEC) enables you to clear the internal exported WSDL/WADL status flag for service operations that contain specific messages, or for all service operations in the database.

Note. Clearing the internal exported WSDL/WADL status flag on a service operation is one way to resolve a WSDL/WADL flag inconsistency. Other options for resolving this condition are discussed elsewhere in this chapter.

See Chapter 6, "Managing Messages," Understanding Using Project Copy and Exported WSDL and WADL, page 121.

The following example shows the Clear WSDL Status page:



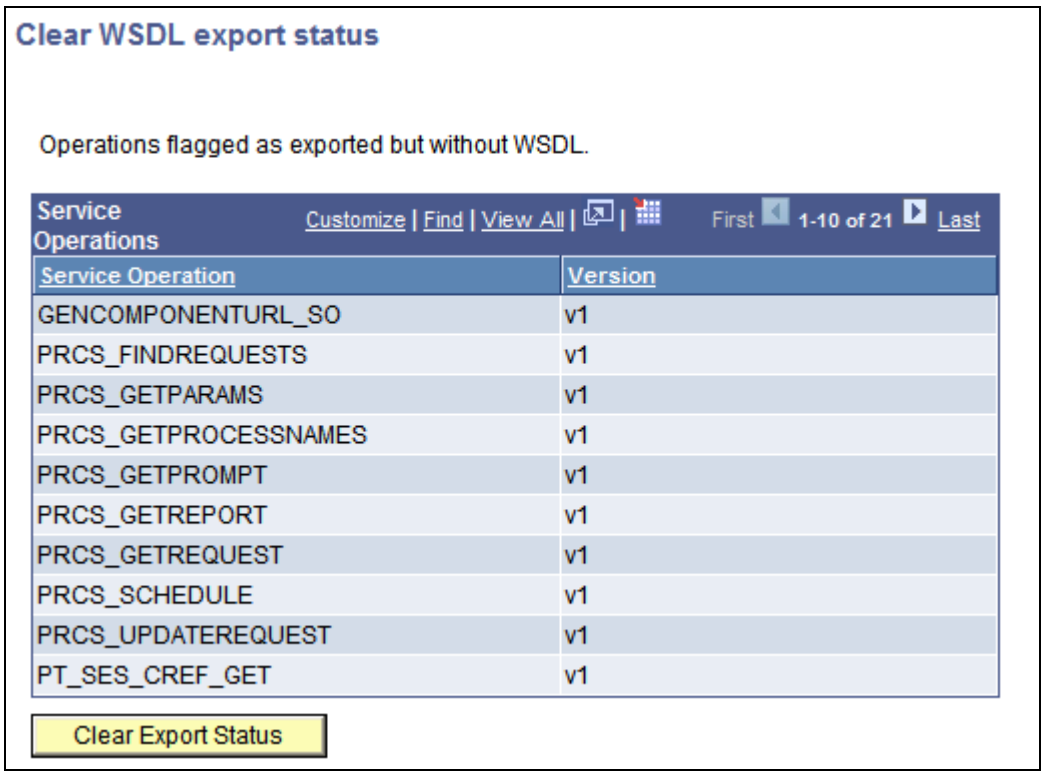
Clear WSDL Export Status page accessed from the Exported WSDL Inconsistencies page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Up to this point, this section has demonstrated accessing the Clear WSDL Export Status page starting from the Export WSDL Inconsistency link on a message definition, and then clicking on the Service Admin link from the Exported WSDL Inconsistencies page. When you access the page using this navigation, only the service operations that reference the message definition that you were originally viewing on the Messages-Message Definitions page appear. Further, those service operations that appear are those that are flagged has having WSDL/WADL exported, but for which there is none in the database.

You can also clear the WSDL/WADL export status flag for all service operations in the database that are in the inconsistent state of having been flagged as having WSDL/WADL generated, but no WSDL/WADL exists in the database for them. You can do so by accessing the Service Administration-WSDL page and clicking the Clear WSDL Export Status link. The Clear WSDL Export Status page appears populated with all service operations in the database that have inconsistent WSDL.

The following example shows the Clear WSDL Export Status page accessed from the Clear Export Status link on the Service Administration-WSDL page.



Clearing the WSDL export status flag for all service operations that are flagged as having WSDL exported but for which there is none in the database.

To clear the WSDL/WADL exported status flag:

1. Access the WSDL Export Status page using one of the following methods:

   - From a message definition that displays the "Exported WSDL flag inconsistency" status message: Click the Exported WSDL Inconsistency link. The Exported WSDL Inconsistencies page appears. Click the Service Admin link.

   - From the PeopleTools menu: Select PeopleTools, Integration Broker, Service Utilities, Service Administration. The Service Administration page appears. Click the WSDL tab. Click the Clear WSDL Export Status link.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Click the Clear Export Status button.

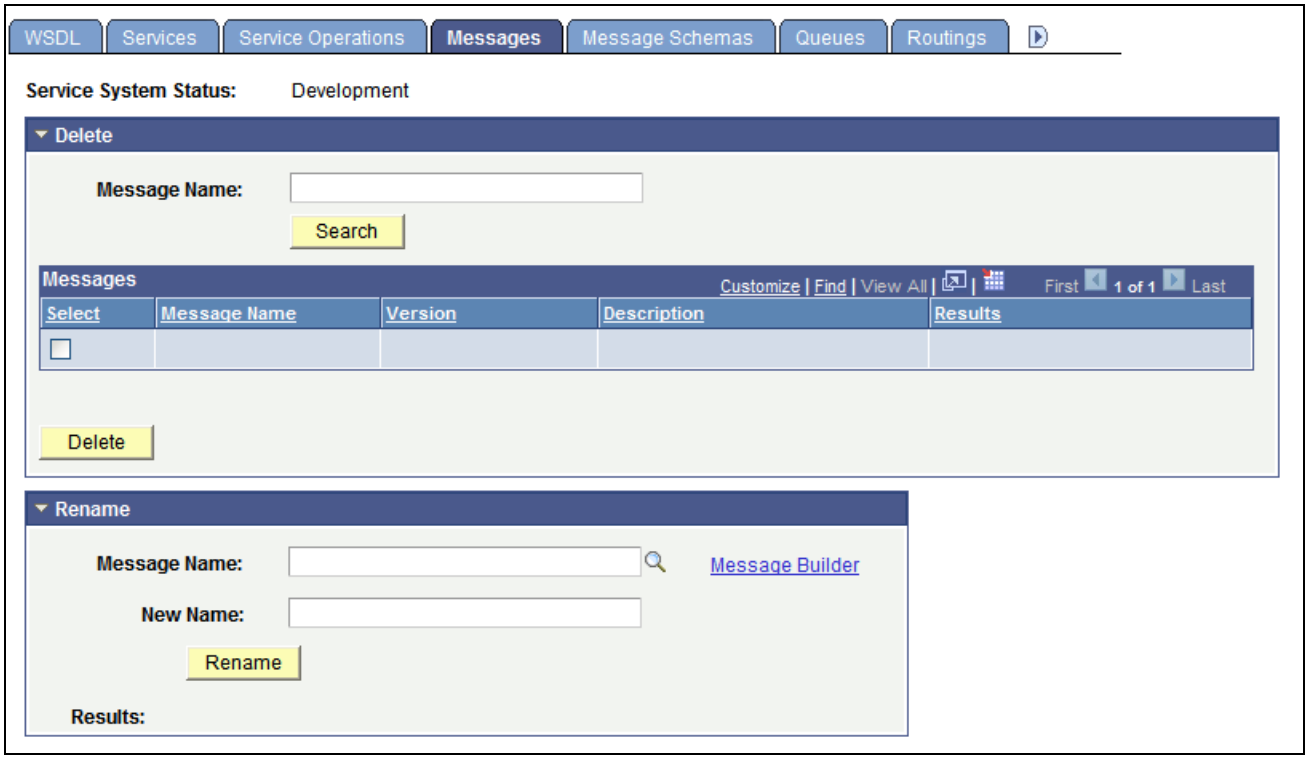
Renaming and Deleting Message Definitions

You can rename and delete messages using the Messages page (IB_HOME_PAGE5) in the Services Administration component (IB_HOME_PAGE). The Message page contains two sections: a Delete section that enables you to delete message definitions and a Rename section that enables you to rename message definitions.

To access the page, select PeopleTools, Integration Broker, Service Utilities, Service Administration, and click the Messages tab.

When you first access the Messages page, all sections are collapsed. Click the section header arrow buttons to expand and collapse each section.

The following example shows the Messages page with the Delete and Rename sections expanded:



Services Administration Messages page with the Delete and Rename sections expanded

At the top of the page, the Service System Status field displays the current setting. The service system status, set on the Service Configuration page, affects the ability to rename and delete messages.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services."

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Renaming Message Definitions

To rename a message definition:

Note. Renaming a message definition renames all versions.

1. Access the Services Administration - Messages page.

   Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Click the Messages tab.

2. Click the arrow next to the Rename section header to expand the section.
3. In the Message Name field, enter the message definition to rename, or click the Lookup button to search for and select the message to rename.
4. (Optional.) Click the Message Builder link to view details about the selected message in the Messages - Message Definitions page.

   When you are done viewing the message details, click the Return button to return to the Services Administration - Messages page.

5. In the New Name field, enter the new name for the message definition.
6. Click the Rename button.

Deleting Message Definitions

When you delete a message definition the system also deletes it's associated schema.

To delete a message definition:

1. Access the Services Administration - Messages page.

   Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Click the Messages tab.

2. Click the arrow next to the Delete section header to expand the section.
3. In the Message Name field, enter the name of the message to delete, and click the Search button.

   Search results appear in the results grid.

4. In the results grid, select the check box next to the message or messages to delete.
5. Click the Delete button.

Deleting Messages During Upgrade

To delete a message definition in an application upgrade project, you must first make sure that no live instances of the message exist. Archive or delete any such messages in both the source and the target database. Otherwise, you receive an error message during the copy process indicating that the object is in use.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 7

Sending and Receiving Messages

This chapter discusses how to:

- Generate and send messages.
- Receive and process messages.
- Process inbound errors.
- Use Message object functionality with nonrowset-based messages.
- Generate test messages.
- Work with message segments.
- Populate and retrieve document data.
- Send and receive binary data using MTOM encoding.
- Use PeopleCode to manage REST service operations.

Understanding Sending and Receiving Messages

To send and receive messages you use PeopleCode to:

- Send request messages from PeopleSoft Integration Broker to other systems.
- Receive response messages from other systems.
- Route messages.
- Manipulate message content.

You can also send messages directly to the integration gateway, thereby bypassing processing on the integration engine.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Prerequisites for Sending and Receiving Messages

Before you can define PeopleCode to generate, send, receive, and process messages, you must define the message in PeopleSoft Internet Architecture.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Once you create PeopleCode, you must also define nodes, services and service operations to implement a complete integration.

See Chapter 4, "Understanding PeopleSoft Integration Broker Metadata," page 35.

Messaging Process Flows

The integration engine uses asynchronous request processes and synchronous request processes to manage outbound and inbound messages. These processes examine the messaging elements that you create to determine how to treat each message.

Outbound Message Processing Flow

This section discusses message processing flow for outbound messages. In this section, the term process is used, and refers to either the integration engine's asynchronous request process or its synchronous request process, depending on the type of integration you are preforming.

Outbound messages you send go through the following steps.

1. The application triggers the sending PeopleCode that you developed.
2. The PeopleCode program populates and sends the message by using an asynchronous or synchronous method.
3. The method that the PeopleCode uses to send the message triggers a request process in the application's integration engine.
4. The process searches the outbound routings that are associated with that service operation to determine the valid target nodes for the message.

   The asynchronous process examines only asynchronous routings, and the synchronous process examines only synchronous routings. If for synchronous processing, a valid single outbound routing cannot be found, the sending method returns an error.

   Note. Only active routings are considered for processing.
5. For each outbound routing that it finds, the process submits the message to the local gateway, along with transaction information about the node and the target connector that should be used to send the message.
6. The local gateway transmits the message to the specified target node through the specified target connector.
7. If this is a synchronous message, the process waits for the target node to pass a response message back through the gateway, then returns it to the calling PeopleCode method.

Inbound Message Processing Flow

Each received message goes through the following steps:

1. The application's gateway receives a request message from a remote node or gateway, which specifies the application as its target node.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. The gateway submits the message to the application's integration engine, which searches for any inbound request routing parameter which has the same alias name as the external operation name passed in.

3. If a matching routing alias name isn't found, the integration engine returns an error message through the gateway to the sending node.

   If a routing alias name is found, the integration engine invokes either the asynchronous request process or the synchronous request process, as appropriate, to handle the message.

   Note. Any inbound routing alias that is found must have the proper permissions for that service operation for the process to proceed.

4. The process accesses the service operation that matches the routing alias name and passes the message to the service operation's handler associated with receiving PeopleCode.

   - The asynchronous request process invokes the service operation's handler OnNotify event PeopleCode.
   - The synchronous request process invokes the service operation's handler OnRequest event PeopleCode.

5. If this is a synchronous transaction, the process waits for the receiving PeopleCode to generate and return a response message, then passes it back to the sending node through the gateway.

Understanding Integration PeopleCode

This section discusses the PeopleCode used for integrations and describes:

- Sending and receiving PeopleCode.
- Integration application classes.
- Integration methods.
- Messaging methods.
- Error-handling methods.
- Messaging PeopleCode.
- Documents PeopleCode.

Sending and Receiving PeopleCode

This section discusses the PeopleCode you use for sending messages from PeopleSoft Integration Broker to other systems, and the PeopleCode you use for receiving messages from other systems.

Sending PeopleCode

PeopleCode for sending messages can be located in PeopleCode events associated with records, record fields, and components, and in application engine programs.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The PeopleCode method used to send messages is highlighted in the following table.

Transmission Type	*Sending PeopleCode*	*Comments*
Synchronous	SyncRequest method.	The SyncRequest method belongs to the IntBroker class.
Asynchronous	Publish method.	The Publish method belongs to the IntBroker class.

To work with rowset-based messages in SOAP format, transform the SOAP documents into XML documents and then use the IntBroker class SyncRequest or Publish methods. To work with nonrowset-based messages in SOAP format use the SOAPDoc class.

Receiving PeopleCode

The PeopleCode that you use to receive a message must be associated with the message definition. The transmission type of the message determines the location of the PeopleCode program.

Implement the OnRequest method for synchronous messages. Implement the OnNotify method for asynchronous messages. Both methods are located in the PS_PT application package, in the Integration sub-package, in the IRequestHandler and INotificationHandler classes, respectively.

Transmission Type	*Message Structure*	*Receiving PeopleCode*	*Comments*
Synchronous	Rowset-based	Message is passed into the method.	Implement the OnRequest method in the IRequestHandler application interface.
Synchronous	Nonrowset-based	Message is passed into the method.	Implement the OnRequest method in the IRequestHandler application interface.
Asynchronous	Rowset-based	Message is passed into the method.	Implement the OnNotify method in the INotificationHandler application interface.
Asynchronous	Nonrowset-based	Message is passed into the method.	Implement the OnNotify method in the INotificationHandler application interface.

To get content data out of a request message, use the following guidelines.

Message Structure	*PeopleCode*	*Comments*
Rowset-based	GetRowSet method.	None.
Nonrowset-based	GetXMLDoc method.	You can also use Message class functionality with nonrowset-based messages. See Chapter 7, "Sending and Receiving Messages," Using Message Object Functionality With Nonrowset-Based Messages, page 177.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Application Classes

Application classes house the processing logic for asynchronous and synchronous messages. By implementing the Integration Broker application classes, you can reuse code and access other benefits of application classes.

The following application classes exist for PeopleSoft Integration Broker. See the individual applicable application class interfaces for more information about the methods contained in an application class.

To access these application classes, in PeopleSoft Application Designer, open the PS_PT application package and open the Integration subpackage.

Note. All of the Integration Broker application classes are defined as interfaces. This means that there is no native implementation of them: you must import them to your program and implement them if you want to use them.

Application Class	*Methods Contained in Application Class*	*Comments*
INotificationHandler	• OnNotify • OnError	This interface is the equivalent of the Subscription Message event PeopleTools releases prior to PeopleTools 8.48.
IReceiver	• OnAckReceive • OnError	This interface is the equivalent of the OnAckReceive Message event in PeopleTools releases prior to PeopleTools 8.48.
IRequestHandler	• OnRequest • OnError	This interface is the equivalent of the OnRequest Message event in PeopleTools releases prior to PeopleTools 8.48.
IRouter	• OnRouteSend • OnRouteReceive • OnError	This interface is the equivalent of the OnRouteSend and OnRouteReceive Message events in PeopleTools releases prior to PeopleTools 8.48.
ISend	• OnRequestSend • OnError	This interface is the equivalent of the OnSend Message event in PeopleTools releases prior to PeopleTools 8.48.

Each of the methods contained in these application classes is described in this section.

Routing Methods

Routing methods determine how a message is routed to or from PeopleSoft Integration Broker.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnRouteSend Method

Implement the OnRouteSend method for outbound synchronous and asynchronous service operations to specify to what node PeopleSoft Integration Broker routes a message. The implementation of this method enables you to apply PeopleCode that filters the destination nodes to which PeopleSoft Integration Broker routes messages.

The OnRouteSend method is contained in the IRouter application class, which is contained in the PS_PT application package, in the Integration subpackage.

When the application PeopleCode is invoked to send a message, the routing definitions in the local database provide a list of target nodes to which PeopleSoft Integration Broker can route the message. The integration engine's request handler invokes the service operation's OnRouteSend event. You can implement the OnRouteSend method in the application package associated with the handler for this service operation, which enables you to apply additional PeopleCode that determines the final target nodes.

You can use OnRouteSend to validate the outbound service operation's target node list, prevent the message from transmitting, or redirect it to a completely different set of targets.

The following table lists the PeopleCode built-in constants that you can use with the OnRouteSend method:

Constant	*Description*
%IntBroker_ROUTE_NONE	Do not send this operation to any of the possible nodes.
%IntBroker_ROUTE_SOME	Send this operation to a selected list of nodes. The node list should be an array of strings in the property destinationNodes.
%IntBroker_ROUTE_ALL	Send this operation to all nodes that have a valid routing.

OnRouteSend enables you to account for multiple synchronous targets. Only one target node at a time can receive a request message sent with a synchronous transaction. Even though you can define the same outbound synchronous transaction for multiple nodes, you must make sure the transaction resolves to a single target node or the transaction fails.

The following code example shows an implementation of this class:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IRouter;

class RoutingHandler implements PS_PT:Integration:IRouter
   method RoutingHandler();
   property array of any destinationNodes;
   method OnRouteSend(&_MSG As Message) Returns integer;
end-class;

/* constructor */
method RoutingHandler
end-method;

method OnRouteSend
   /+ &_MSG as Message +/
   /+ Returns Integer +/
   /+ Extends/implements PS_PT:Integration:IRouter.OnRouteSend +/
   /* Variable Declaration */
   Local any &aNodeList;
   Local any &rootNode;
   Local any &xmlDoc;

   /* Check the message for the instructions on how to execute
     the OnRouteSend.*/

   &xmlDoc = &_MSG.GetXmlDoc();
   &rootNode = &xmlDoc.DocumentElement;
   &aNodeList = &rootNode.GetElementsByTagName("OnRouteSend");

   If (&aNodeList.Len <> 1) Then

      /* No Nodes are in the list, therefore exit. */
      Exit;
   Else

      /* check the value of the node to determine the action to
       take. */

      Evaluate &aNodeList [1].NodeValue
      When "True"
         Return (%IntBroker_ROUTE_ALL);
         Break;
      When "False"
         Return (%IntBroker_ROUTE_NONE);
         Break;
      When-Other

         /* assume that this is to be routed to the node given */
         Local array &nodeArray;
         &nodeArray = CreateArray();
         &nodeArray.Push(&aNodeList [1].NodeValue);

         Local string &sIBVariableTest = GetCurrentType(&nodeArray);
         Evaluate &sIBVariableTest
         When "Array"
            &destinationNodes = &nodeArray.Clone();
            Return %IntBroker_ROUTE_SOME;
         When "BooleanTrue"
            Return %IntBroker_ROUTE_ALL;
         When "BooleanFalse"
            Return %IntBroker_ROUTE_NONE;
         End-Evaluate;
         Break;

      End-Evaluate;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
   End-If;

end-method;
```

OnRouteReceive Method

Implement the OnRouteReceive method for inbound synchronous and asynchronous service operations to apply PeopleCode that determines whether the default local node accepts inbound messages.

The OnRouteReceive method is contained in the IRouter application class, which is contained in the PS_PT application package, in the Integration subpackage.

When the integration engine receives a message, the transaction definitions in the local database provide a list of source nodes from which the application can accept the message. The integration engine's request handler invokes the service operation's OnRouteReceive event. You can implement the OnRouteReceive method in the application package associated with the handler for this service operation, which enables you to apply PeopleCode that determines whether the default local node accepts the inbound message. You can employ this event regardless of the message transmission type.

The following is an example implementation of this method:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IRouter;

class RoutingHandler implements PS_PT:Integration:IRouter
   method RoutingHandler();
   property array of any destinationNodes;
   method OnRouteReceive(&_MSG As Message) Returns boolean;
end-class;

/* constructor */
method RoutingHandler
end-method;

method OnRouteReceive
   /+ &_MSG as Message +/
   /+ Returns Boolean +/
   /+ Extends/implements PS_PT:Integration:IRouter.OnRouteReceive +/
   /* Variable Declaration */
   Local any &aNodeList;
   Local any &rootNode;
   Local any &xmlDoc;

   /* Check the message for instructions on how to execute
    the OnRouteReceive.*/

   &xmlDoc = &_MSG.GetXmlDoc();
   &rootNode = &xmlDoc.DocumentElement;
   &aNodeList = &rootNode.GetElementsByTagName("OnRouteReceive");

   If (&aNodeList.Len <> 1) Then

      /* A single node must be present. */
      Exit;
   Else

      /* check the value of the node to determine the action to
        take. */

      Evaluate &aNodeList [1].NodeValue
      When "True"
         Return ( True);
         Break;
      When "False"
         Return ( False);
         Break;
      When-Other
         /* don't recognize the value. */
         Exit;
      End-Evaluate;

   End-If;

end-method;
```

Messaging Methods

This section describes methods used in messaging and the application classes in which they are contained.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Outbound Messaging Methods

This section describes methods used on outbound messages from PeopleSoft to other systems.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnRequestSend

Implement for outbound synchronous and asynchronous service operations to override connector properties before sending a message to the integration gateway.

This method is contained in the ISend application class.

The OnRequestSend method passes in a message to your derived application class method. The returned value needs to be a message.

The following is an example implementation of this method.

```
import PS_PT:Integration:ISend;

class SendHandler implements PS_PT:Integration:ISend
   method SendHandler();
   method OnRequestSend(&_MSG As Message)
    Returns Message;
end-class;

/* constructor */
method SendHandler
end-method;

method OnRequestSend
   /+ &_MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend. +/
   /+ OnRequest Send +/
   /* Variable Declaration */
   Local any &tempNode;
   Local any &rootNode;
   Local any &xmlDoc;
   Local any &msg;

   &msg = &_MSG;

   &xmlDoc = &msg.GetXmlDoc();

   /* Add a node to the doc to prove that we can
      edit it in this event. */

   &rootNode = &xmlDoc.DocumentElement;

   &tempNode = &rootNode.AddElement("OnSend");
   &tempNode.NodeValue = "If you see this, then
   the Sync OnSend PCode has altered the message";

   /* and write the data back into the message */

   &msg.SetXmlDoc(&xmlDoc);

   Return (&msg);

end-method;
```

See Chapter 7, "Sending and Receiving Messages," Setting and Overriding Target Connector Properties at Runtime, page 151.

When using the ISend handler with message parts, specifically with rowset-based message parts, the rowsets of the parts must be retrieved *in the order* that the content data will be sent.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following is an example that can be used for ISend events that use rowset-based parts (even for the cases where one is just overriding the connectors):

```
method OnRequestSend
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend. +/
   /+ OnRequestSend +/
   If (&MSG.IsPartsStructured) Then
      Local number &i;
      Local Rowset &rs;
      For &i = 1 To &MSG.PartCount
         &rs = &MSG.GetPartRowset(&i);
      End-For;
   End-If;

      Return &MSG;

end-method;
```

OnAckReceive

Implement for outbound asynchronous service operations to access the body of a message acknowledgement to check for SOAP faults.

This method is contained in the IReceiver application class.

The following is an example implementation of this method.

```
import PS_PT:Integration:IReceiver;

class AckReceiveHandler implements PS_PT:
  Integration:⇒
IReceiver
   method AckReceiveHandler();
   method OnAckReceive(&_MSG As Message) Returns
   integer;
end-class;

/* constructor */
method AckReceiveHandler
end-method;

method OnAckReceive
   /+ &_MSG as Message +/
   /+ Returns Integer +/
   /+ Extends/implements PS_PT:Integration:+/
   /+ IReceiver.OnAck Receive +/
   /* Variable Declaration */
   /*
/* We return a hardcoded value. In this case, a
   message error.*/

   Return (%Operation_Error);

end-method;
```

See Chapter 7, "Sending and Receiving Messages," Handling Inbound Asynchronous Transactions, page 156.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Inbound Messaging Methods

This section describes methods used on inbound messages to PeopleSoft from other systems.

OnRequest

Implement for inbound synchronous service operations.

This method is contained in the IRequestHandler application class.

The following is an example implementation of this method:

```
class RequestHandler implements PS_PT:Integration:
 IRequestHandler
   method RequestHandler();
   method OnRequest(&_MSG As Message) Returns
   Message;
end-class;

/* constructor */
method RequestHandler
end-method;

method OnRequest
   /+ &_MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:+/
   /+ IRequestHandler.OnRequest +/
   /* Variable Declaration */
   Local any &tempNode;
   Local any &descNode;
   Local any &rootNode;
   Local any &xmlDoc;
   Local any &xmldata;
   Local any &msg;

   &msg = CreateMessage(Operation.QE_IB_SYNC_RESP,%Int⇒
Broker_response);

   &xmldata = "<?xml version='1.0'?>
   <QE_IB_PeopleCode_Test⇒/>";

   &xmlDoc = CreateXmlDoc(&xmldata);

   &rootNode = &xmlDoc.documentelement;

   &descNode = &rootNode.AddElement("Description");
   &descNode.NodeValue = "Sync test of OnRouteSend.";

   &tempNode = &rootNode.addelement("OnRequest");
   &tempNode.NodeValue = "If you see this,
   then the On Request PCode created the response
   message";

   &msg.SetXmlDoc(&xmlDoc);

   Return &msg;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnNotify

Implement for inbound asynchronous service operations. This method can be used for code that does subscription processing, and for validating and loading message data.

This method is contained in the INotificationHandler application class.

The following is an example implementation of this method:

```
import PS_PT:Integration:INotificationHandler;

class NotificationHandler implements PS_PT:Integration:
  INotificationHandler
   method NotificationHandler();
   method OnNotify(&_MSG As Message);
end-class;

/* constructor */
method NotificationHandler
end-method;

method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration: +/
   /+ INotificationHandler.OnNotify +/
   /* Variable Declaration */

   Local Rowset &rs;

   &rs = &MSG.GetRowset();

   /* process data from rowset */
end-method;
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnResponse

Implement for inbound response asynchronous service operations.

This method can be used for code that does response subscription processing. This method is contained in the INotificationHandler application class.

The following is an example implementation of this method and shows how to get the request TransactionID.

```
import PS_PT:Integration:INotificationHandler;

class RESPONSE_NOTIFICATION implements PS_PT:
 Integration:INotificationHandler
   method RESPONSE_NOTIFICATION();
   method OnNotify(&MSG As Message);

end-class;

/* constructor */
method RESPONSE_NOTIFICATION
   %Super = create PS_PT:Integration:INotificationHandler⇒
();
end-method;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:+/
   /+ INotification Handler.OnNotify +/

   Local Rowset &rs;
   Local boolean &Ret;
   Local string &TransactionID;

   &rs = &MSG.GetRowset();

   If &MSG.IsSourceNodeExternal Then

      /* if the request came from an external non
        PeopleSoft System then you can get the
        original TransactionID from the WSA_MessageID
        from the request message. */

      &TransactionID = &MSG.IBInfo.WSA_MessageID;

   Else

      /* if the request came from a PeopleSoft
         System then get the original TransactionID
         from the  nReplyToID */

      &TransactionID = &MSG.IBInfo.InReplyToID;

   End-If;

end-method;
```

Error-Handling Methods

Each application class contained in the Integration application subpackage contains an OnError method that you can use for custom error handling.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Custom error handling can include sending an email notification or entering data in a log when an error occurs.

For the IRequestHandler application class, the OnError function returns a string. This enables you to send back custom error messages, for example SOAP faults, to non-PeopleSoft consumers. If the message consumed was a SOAP message and the custom error message is already wrapped in SOAP, it will not be modified and is sent as-is. However, if the OnError message is not SOAP, it is wrapped as a standard SOAP fault and returned to the sender.

If the message consumer is another PeopleSoft system the message set/message ID framework applies.

If an error occurs the OnError method, if implemented, is automatically invoked. The type of exception can be viewed by using the Message object to retrieve an Exception object populated with information about the error, using the message class IBException property.

The following is an example of the OnError method implementation:

```
/*On Error Implementation */
method OnError
   /+ &MSG as Message +/
   /+ Returns String +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnError +/
   Local integer &nMsgNumber, &nMsgSetNumber;
   Local string &sText;

   &nMsgNumber = &MSG.IBException.MessageNumber;
   &nMsgSetNumber = &MSG.IBException.MessageSetNumber;
   rem    &sText = &exception.DefaultText;
   &sText = &MSG.IBException.ToString();


   /* ADD SPECIFIC ERROR INFO HERE */
   Return &sText;

end-method;
```

See *PeopleTools 8.52: PeopleCode API Reference*, "Exception Class."

See *PeopleTools 8.52: PeopleCode API Reference*, "Message Classes," IBException.

Messaging PeopleCode

Messaging PeopleCode enables you to manipulate message content. The messaging PeopleCode classes you can use for this are:

Message classes	Use for rowset or nonrowset-based messages.
SOAPDoc class	Use for SOAP-compliant messages.
XMLDoc classes	Use for XML messages.
Document class	Use for Document type messages.

XML and SOAP-compliant messages are nonrowset-based messages. You can use their respective classes to manipulate message content, or use the Message classes.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 7, "Sending and Receiving Messages," Using Message Object Functionality With Nonrowset-Based Messages, page 177

PeopleTools 8.52: PeopleCode API Reference, "Message Classes"

PeopleTools 8.52: PeopleCode API Reference, "SOAPDoc Class"

PeopleTools 8.52: PeopleCode API Reference, "XmlDoc Classes"

Document PeopleCode

PeopleSoft provides a Document API for populating and retrieving document data that includes several built-in functions and classes.

The built-in functions are:

CreateDocument	Use this built-in function to instantiate a Document object.
CreateDocumentKey	Use this built-in function to instantiate a Document Key object

The classes are:

- Document class.

  Use the methods and properties in this class to populate and retrieve document data.

- DocumentKey class.

  Use the methods and properties in this class to create document keys.

  Document keys enable you to map the document package, document name, and document version, to one string. As a result, when you populate or retrieve data from a document, you can specify the one document key, instead of specifying the document package, name, and version.

  populating or retrieving data from a document, you can use the one document key

- Primitive class.

  Use the methods and properties in this class to populate and retrieve document data from primitive elements.

- Compound class.

  Use the methods and properties in this class to populate and retrieve document data from compound elements.

- Collection class.

  Use the methods and properties in this class to populate and retrieve document data from collection elements.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The Document API is discussed in the *PeopleTools 8.52: PeopleCode API ReferencePeopleBook.*

Examples of populating and retrieving document data are provided elsewhere in this chapter.

Generating and Sending Messages

This section provides an overview of outbound messaging and discusses how to:

- Handle outbound asynchronous message transmission.
- Handle outbound asynchronous request/response message transmission.
- Handle outbound synchronous message transmission.
- Read exceptions for outbound synchronous integrations.
- Handle cookies in messages.

Understanding Outbound Messaging

Successful outbound messaging relies on sending messages in the proper order and on testing the messages. Messages containing non-XML data have special considerations.

Message Order

PeopleSoft Integration Broker guarantees that messages are delivered in the order in which you send them and that they are single-threaded at the PeopleSoft receiving node. However, message order is not part of the queue definition. You must send messages in the proper order.

Note. You can modify this behavior by using queue partitioning.

See Chapter 13, "Managing Service Operation Queues," Applying Queue Partitioning, page 324.

Message Testing

Make sure that you adequately unit-test and system-test your messages.

Unit-test a message by triggering the PeopleCode that sends the message and then view the message details in Service Operations Monitor. From the Service Operations Monitor, you can view the contents of each field to verify that the message data is formatted correctly.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook*

You can also test handler code using the Handler Tester utility.

See *PeopleTools 8.52: Integration Testing Utilities and Tools PeopleBook*

Message Class Outbound PeopleCode

Use the record class SelectByKey method whenever possible to get data that isn't in the component buffer.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

If the record names are the same, use the standard record methods, such as SelectByKey, Insert, and Update, on the message records.

There are no automatic checks for message size. You must do it programmatically. Use the following code at level 0 to control message size when dealing with multiple transactions:

```
If &Msg.Size > %MaxMessageSize
```

Note. The OnRouteSend method enables you to apply PeopleCode that filters the destination nodes.

See *PeopleTools 8.52: PeopleCode API Reference*, "Record Class."

Non-XML Data

If you're generating a non-XML outbound message, it's up to you to insert the message content into a special XML section containing a CDATA tag:

```
<xml psnonxml="yes">
  <![CDATA[nonXML_message_data]]>
```

Outbound Messaging and Global Variables

When you invoke a SyncRequest method the system clears any declared global variables after OnRouteSend or OnSend PeopleCode events are fired. If a component attempts to access any of the global variables after the SyncRequest method, a context error occurs.

OnRouteSend and OnSend events are primarily used for asynchronous messaging, however they can be use for synchronous messages.

To avoid context errors when using OnRouteSend or OnSend events for synchronous messaging, following these guidelines:

1. Do not use global variables.
2. If you must use global variables, save them as temporary variables prior to executing a SyncRequest event, then after the event is fired re-assign them back to the globals.
3. Run OnRouteSend or OnSend logic prior to the SyncRequest and after the node is obtained use sender-specific routing. In sender-specific routing you pass the node as part of the SyncRequest call. This will send the request to the node as long as there is an active routing. Any connector overrides can be performed prior to the SyncRequest call and set on the message.

Handling Outbound Asynchronous Message Transmission

To send a message asynchronously, use the IntBroker class Publish method in:

- A record field PeopleCode event.
- A component PeopleCode event.

  When publishing from a component, publish messages only from the SavePostChange event, using the deferred mode property.
- An Application Engine program.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- An implementation of the OnNotify method.
- An implementation of the OnRequest method .

  The OnRequest service operation event is triggered only when an inbound transaction occurs. However, when the receiving PeopleCode runs, the program can also send messages.

Message Class Outbound Asynchronous Example

The following example uses the Publish method in the PeopleCode IntBroker class:

```
Local Message &MSG;
Local Rowset &SALES_ORDER, &RS;

 /*Get a pointer to the component buffer rowset */
&SALES_ORDER = GetLevel0();
/*Create an instance of the SALES_ORDER_ASYNC message object */
&MSG = CreateMessage(OPERATION.SALES_ORDER_ASYNC);

/*Copy the rows from the rowset to the message object */
&MSG.CopyRowset(&SALES_ORDER);
/*Send the message */
%IntBroker.Publish(&MSG);
```

XmlDoc Class Outbound Asynchronous Example

The following example uses the Publish method:

```
Local XmlDoc &xmlRequestDoc;
Local Message &MSG;

/*Create an XmlDoc Object */
&xmlRequestDoc = CreateXmlDoc();

/* Parse a URL or input XML file into an XmlDoc */

&bool = &xmlRequestDoc.ParseXmlFrom URL("C:\pt\appserv\files\
input.xml");

/* Populate message with XML data */
&MSG = CreateMessage(OPERATION.XmlRequest);

&MSG.SetXmlDoc(&xmlRequestDoc);

/* Sent request */

%IntBroker.Publish(&MSG);
```

Identifying SOAP Faults

You can implement the OnAckReceive method to access IBInfo data. This enables you to read the content of acknowledgements returned by recipient systems of asynchronous SOAP messages. The ability to access acknowledgement content is useful when sending SOAP messages, since although there may be no HTTP protocol errors while sending them, SOAP faults may occur.

If the message is nonrowset-based, use the message class GetXmlDoc method to get the response data. This returns an XmlDoc object.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

If the message is rowset-based, use the message class GenXMLString method to get the response data. This returns a string object which you can load into an XmlDoc object.

If SOAP faults are found, you can set the status equal to Error so that the errors appear in the Service Operations Monitor for the publication contract.

The following code example shows how to use GetXmlDoc and GenXMLString in an implementation of the OnAckReceive method. Valid status overrides are %Operation_Done, %Operation_Error, and %Operation_Retry:

```
import PS_PT:Integration:IReceiver;

class AckReceiveHandler implements PS_PT:Integration:IReceiver
   method AckReceiveHandler();
   method OnAckReceive(&_MSG As Message) Returns integer;
end-class;

/* constructor */
method AckReceiveHandler
end-method;

method OnAckReceive
   /+ &_MSG as Message +/
   /+ Returns Integer +/
   /+ Extends/implements PS_PT:Integration:IReceiver.OnAckReceive +/
   /* Variable Declaration */

If &MSG.IsStructure Then

/* if message is rowset-based */
&str = &MSG.GenXMLString();

Else
/* if message is nonrowset-based */
&xmldoc = &MSG.GetXmlDoc();

End-If;

 Return (%Operation_Done);

end-method;
```

You can also implement the OnAckReceive method to read response content data returned from third-party systems when using the HTTP target connector.

See Also

PeopleTools 8.52: PeopleCode API Reference, "XmlDoc Classes"

Handling Outbound Asynchronous Request/Response Message Transmission

To transmit an outbound asynchronous request/response message, send the message asynchronously using the Publish method.

See Chapter 7, "Sending and Receiving Messages," Handling Outbound Asynchronous Message Transmission, page 145.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Handling Outbound Synchronous Transactions

Use the IntBroker class SyncRequest method for handling outbound synchronous transfers. To send a message synchronously, you can employ SyncRequest in:

- The record field SavePreChange PeopleCode event.
- The record field SavePostChange PeopleCode event.
- The record field Workflow PeopleCode event.
- The record field FieldChange PeopleCode event.
- An implementation of the OnRequest method.
- An implementation of the OnNotify method.

Note. The OnRequest and OnNotify events are triggered only when an inbound transaction occurs, however, when the receiving PeopleCode runs, it can also send messages.

The response message that is returned from an outbound synchronous transaction is no different from an inbound request message. Once you have it in a Message, XmlDoc, or SoapDoc object, it has the same properties as any other object of that type and can, therefore, be treated exactly the same way.

See Chapter 7, "Sending and Receiving Messages," Receiving and Processing Messages, page 155.

Message Class Outbound Synchronous Example 1

The following example uses the IntBroker class SyncRequest method:

```
Local Message &MSG, &response;
Local Rowset &SALES_ORDER;

&SALES_ORDER = GetLevel0();
&MSG = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&MSG.CopyRowsetDelta(&SALES_ORDER);

/* send the synchronous request; the return value is the response
message object */
&response = %IntBroker.SyncRequest(&MSG);

/* check the response status; 0 means OK */
If (&response.ResponseStatus = 0) Then
  /* process the response */
  MY_SALES_ORDER_SYNC.ORDER_ID = &response.GetRowset().GetRow(1)
.GetRecord(Record.SO_RESPONSE).GetField(Field.ORDER_ID).Value);
else

  /* do error handling */

End-If;
```

Message Class Outbound Synchronous Example 2

The following example shows the use of CopyTo to get the data back from the response and into the component buffer, and therefore the page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &msgZipRequest, &msgZipResponse;
Local Rowset &RS, &rsMessageRowset;


&RS = GetLevel0();
&msgZipRequest = CreateMessage(OPERATION.ZIP_REQUEST);
&msgZipRequest.CopyRowset(&RS);
/* send the synchronous request; the return value is the response
message object */

&msgZipResponse = %IntBroker.SyncRequest(&msgZipRequest,
Node.ZIPTOCITYANDSTATE);

/* check the response status; 0 means OK */
If (&msgZipResponse.ResponseStatus = 0) Then
   /* process the response */
   &rsMessageRowset = &msgZipResponse.GetRowset();
   &rsMessageRowset.CopyTo(&RS);
else
   /* do error handling */
End-If;
```

XmlDoc Class Outbound Synchronous Example

The following example uses the IntBroker class SyncRequest method:

```
Local Message &MSG, &RESP_MSG;
Local XmlDoc &flightplan_xmldoc, &xmldocReturn;
Local XmlNode &ac_number, &msi_sensor, &ofp;

&flightplan_xmldoc = CreateXmlDoc("");

&ac_number = &flightplan_xmldoc.CreateDocumentElement("flightplan");

&msi_sensor = &ac_number.AddElement("msi_sensor");
&msi_sensor.NodeValue = "flir";
&ofp = &ac_number.AddElement("ofp");
&ofp.NodeValue = "8.44";

&MSG = CreateMessage(Message.SYNC_REQUEST_EXAMPLE);

&MSG.SetXmlDoc(&flightplan_xmldoc);

&RESP_MSG = &MSG.SyncRequest();

&xmldocReturn = &RESP_MSG.GetXmlDoc();

&return_data = &xmldocReturn.GenXmlString();
```

See Also

PeopleTools 8.52: PeopleCode API Reference, "XmlDoc Classes"

Reading Exceptions for Outbound Synchronous Integrations

The Routing – Routings Definition page features a User Exception check box that enables you to capture Integration Broker exceptions for outbound synchronous integrations using PeopleCode.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Do not use Try/Catch PeopleCode to attempt to read exceptions on outbound SyncRequest calls.

The following code example shows how to read captured exceptions:

```
&Return_MSG = %IntBroker.SyncRequest(&MSG);
If &Return_MSG.ResponseStatus = %IB_Status_Success Then

/* process the response message */
 &RS = &MSG.GetPartRowset();

Else

 /* evaluate the error and either throw a PeopleCode exception or continue⇒
    processing */
   &error_string = &Return_MSG.IBException.ToString());
   &nErrorMsgNumber = &Return_MSG.IBException.MessageNumber;
   &nErrorMsgSetNumber = &Return_MSG.IBException.MessageSetNumber;

End-If;
```

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382

Overriding Synchronous Timeout Intervals at Runtime

For long-running outbound synchronous transactions, you can override the default timeout period the transaction at runtime using the SyncServiceTimeout property. The default synchronous timeout period is five minutes.

The HTTP header file is modified to take this parameter. The value you set is sent to the integration gateway where it is used for the HTTP timeout.

The SyncServiceTimeout property takes a time (in seconds). The property is read-write.

The following code example shows how to use the property. To use this property, note that you must override and setup the target connector properties for the transaction. As the example demonstrates, there are helper methods that load properties based on node or transaction.

```
&MSG.SetXmlDoc(&xmlReq);
&MSG.IBInfo.LoadConnectorPropFromNode(Node.EAI);
&MSG.IBInfo.SyncServiceTimeout = 360000;
&MSG.IBInfo.ConnectorOverride = True;
&MSG_Resp = %IntBroker.SyncRequest(&MSG, Node.EAI);
&xmlResponseDoc = &MSG_Resp.GetXmlDoc();
```

See Also

Chapter 7, "Sending and Receiving Messages," Setting and Overriding Target Connector Properties at Runtime, page 151

PeopleTools 8.52: System and Server Administration, "PeopleSoft Timeout Settings"

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Handling Cookies

PeopleSoft Integration Broker provides basic cookie handling for exchanges that are initiated by your PeopleSoft application. You can accept a synchronous response message containing cookies, save those cookies in a global variable, and later return them to the remote node in an outbound synchronous or asynchronous request message. This is a typical application of cookies in a web interaction.

Cookies are implemented as an IBInfo class property, Cookies. You can access this property only in an inbound synchronous response message or an outbound request message.

Receiving Cookies Example

The following example retains the cookies from a response message to a global variable:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Send the synchronous request; the return value is the response
message object */
&SalesResponse = %IntBroker.SyncRequest(&SalesRequest);

/* Retrieve cookies from the response message */
&SalesCookies = &SalesResponse.IBInfo.IBConnectorInfo.Cookies;
```

Returning Cookies Example

The following example retrieves the previously retained cookies from the global variable and inserts them into a new request message:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Insert the cookies in the request message */
&SalesRequest.IBInfo.IBConnectorInfo.Cookies = &SalesCookies;

/* Send the asynchronous request */
%IntBroker.Publish(&SalesRequest);
```

Setting and Overriding Target Connector Properties at Runtime

PeopleSoft Integration Broker enables you to dynamically override target connector properties at runtime that have previously been set at the node, connector and transaction levels. To set or override target connectors at runtime, use the PeopleCode IBInfo object, the Connector Info object and implement the OnRequestSend method.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Properties set at the PeopleCode level take precedence over those set at the node, connector and routing level.

IBInfo object	An IBInfo object is instantiated from a message object. You can use this object in publishing or synchronous request PeopleCode. You can also use it in your implementation of the OnRequestSend method.
ConnectorInfo object	A ConnectorInfo object is instantiated from an IBInfo object. Use this object for reading and writing connector name/value pair information to and from the IBRequest. You can use this object in publishing or synchronous request PeopleCode. You can also use it in your implementation of the OnRequestSend method.
OnRequestSend Method	The OnRequestSend method is included in the ISend application class. Use your implementation of this method to override target connector properties at runtime for a subscribing node transaction. This event associated with the service operation executes before any transformations are processed. You can use this event for asynchronous and synchronous messages.

Since data is always carried with the message, you can use the IBInfo object, ConnectorInfo object and your implementation of the OnRequestSend method to populate connector information in the publishing PeopleCode and then override it for a specific node.

Setting and Overriding Target Connector Properties Using the OnRequestSend Method

You can use implement the OnRequestSend method to override IBInfo and connector properties at runtime for a subscribing node transaction.

Any content data that is changed on the message or XMLDoc is sent to the subscribing node or used within a transformation.

To override the properties of a target connector, you must set the following statement to *true*:

```
&MSG.IBInfo.ConnectorOverride=true
```

If a publication contract fails as a result of using an implementation of the OnRequestSend method to override connector properties at runtime, correct the PeopleCode in your implementation and resubmit the message.

Example: Setting Target Connector Properties Using the OnRequestSend Method

The following example shows loading all connectors that exist for the node and adding one additional property, FileName.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:ISend;

class SendHandler implements PS_PT:Integration:ISend
   method SendHandler();
   method OnRequestSend(&Msg As Message) Returns Message;
end-class;

/* constructor */
method SendHandler
end-method;

method OnRequestSend
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnRequestSend +/
   /* Variable Declaration */
   Local Any &Bo;
   Local Message &Msg;

&Bo = &MSG.IBInfo.LoadConnectorPropFromNode("nodename");

&Bo = &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties
      ("FileName", "temp", %Property);
&MSG.IBInfo.ConnectorOverride = True;

Return (&Msg);

end-method;
```

Example: Overriding Connector Properties Using the OnRequestSend Method

The following example demonstrates overriding target connector properties using an implementation of the OnRequestSend method for a rowset-based asynchronous message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:ISend;

class SendHandler implements PS_PT:Integration:ISend
   method SendHandler();
   method OnRequestSend(&Msg As Message) Returns Message;
end-class;

/* constructor */
method SendHandler
end-method;

method OnRequestSend
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnRequestSend +/
   /* Variable Declaration */
Local Boolean &bRet;


&bRet= &MSG.IBInfo.LoadConnectorProp("FILEOUTPUT");
&MSG.IBInfo.ConnectorOverride = True;
&bRet= &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties
      ("sendUncompressed", "Y", %Header);
&bRet= &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties
      ("FilePath", "c:\temp", %Property);

Return (&Msg);

End-Method;
```

The following example demonstrates overriding target connector properties using an implementation of the OnRequestSend method for a nonrowset-based asynchronous message.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:ISend;

class SendHandler implements PS_PT:Integration:ISend
   method SendHandler();
   method OnRequestSend(&Msg As Message) Returns Message;
end-class;

/* constructor */
method SendHandler
end-method;

method OnRequestSend
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnRequestSend +/
   /* Variable Declaration */

Local XmlDoc &xmldoc;
Local Boolean &bRet;


// if you have to access the content data for content based
// decisions, do this
&xmldoc = &MSG.GetXmlDoc();

&bRet= &MSG.IBInfo.LoadConnectorProp("FILEOUTPUT");
&MSG.IBInfo.ConnectorOverride = True;
&bRet= &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties
      ("sendUncompressed", "Y", %Header);
&bRet= &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties
      ("FilePath", "c:\temp", %Property);

Return (&MSG);

End-Method;
```

See Also

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," IBInfo Class

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," IBConnectorInfo Collection

PeopleTools 8.52: PeopleCode API Reference, "Message Classes"

Receiving and Processing Messages

This section discusses how to handle:

- Inbound asynchronous transactions.
- Inbound asynchronous request/response transactions.
- Inbound synchronous transactions.
- Simulating receiving messages from external nodes.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. The OnRouteReceive method can be implemented to apply PeopleCode that determines whether the default local node accepts the inbound message.

Handling Inbound Asynchronous Transactions

Implement the OnNotify method in the PS_PT application package, in the Integration sub-package, to handle inbound asynchronous transactions. All the examples in this section are assumed to be implementations of the OnNotify method.

Message Class Inbound Asynchronous Example 1

The following example implements the OnNotify method.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class FLIGHTPROFILE implements PS_PT:Integration:INotificationHandler
   method FLIGHTPROFILE();
   method OnNotify(&_MSG As Message);
end-class;

/* constructor */
method FLIGHTPROFILE
end-method;

method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.+/
   /+ OnNotify +/
   /* Variable Declaration */
   Local any &outstring;
   Local any &i;
   Local any &CRLF;

   Local Message &MSG;
   Local Rowset &rs, &rs1;
   Local Record &FLIGHTDATA, &REC;

   Local string &acnumber_value, &msi_sensor_value, &ofp_value,
   &actype_value, &callsign_value, &squadron_value, &comm1_value,
   &comm2_value, &ecm_value;

   Local XmlDoc &xmldoc;
   Local string &return_string_value;
   Local boolean &return_bool_value;

   &CRLF = Char(13) | Char(10);

   &MSG = &_MSG;

   &rs = &MSG.GetRowset();
   &REC = &rs(1).QE_FLIGHTDATA;

   &FLIGHTDATA = CreateRecord(Record.QE_FLIGHTDATA);
   &REC.CopyFieldsTo(&FLIGHTDATA);

   /* Parse out Message Data */
   &acnumber_value = &FLIGHTDATA.QE_ACNUMBER.Value;
   &msi_sensor_value = &FLIGHTDATA.QE_MSI_SENSOR.Value;
   &ofp_value = &FLIGHTDATA.QE_OFP.Value;
   &actype_value = &FLIGHTDATA.QE_ACTYPE.Value;
   &callsign_value = &FLIGHTDATA.QE_CALLSIGN.Value;
   &squadron_value = &FLIGHTDATA.QE_SQUADRON.Value;
   &comm1_value = &FLIGHTDATA.QE_COMM1.Value;
   &comm2_value = &FLIGHTDATA.QE_COMM2.Value;
   &ecm_value = &FLIGHTDATA.QE_ECM.Value;

   &outstring = "Send Async FLight test";

   /* Construct Output String */
   &outstring = &outstring | &acnumber_value | &CRLF |
   &msi_sensor_value |
   &CRLF | &ofp_value | &CRLF | &actype_value | &CRLF |
   &callsign_value |
   &CRLF | &squadron_value | &CRLF | &comm1_value | &CRLF |
   &comm2_value |
   &CRLF | &ecm_value;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
   /* Log Output String into page record */
   &FLIGHTDATA.GetField(Field.DESCRLONG).Value = &outstring;

   SQLExec("DELETE FROM PS_QE_FLIGHTDATA");
   &FLIGHTDATA.Insert();

end-method;
```

Message Class Inbound Asynchronous Example 2

The following example shows notification PeopleCode that checks the PSCAMA to determine the audit action code and that examines the language code to determine whether related language processing is needed:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
method OnNotify
   /+ &MSG as Message +/

   /* Simple PeopleCode Notifcation- - Check the PSCAMA*/
   Local Rowset &MSG_RS;
   Local Record &REC_NAME_PREFIX, &REC, &REC_RL;
   Local integer &I;
   Local string &ACTION, &LNG, &BASE_LNG, &RELLNG, &KEY1, &KEY2;

   &MSG_RS = &MSG.GetRowset();
   For &I = 1 To &MSG_RS.RowCount /* Loop through all transactions */
      &REC = &MSG_RS.GetRow(&I).GetRecord(Record.NAME_PREFIX_TBL);
      /* Get Audit Action for this transaction */
      &ACTION = &MSG_RS.GetRow(&I).PSCAMA.AUDIT_ACTN.Value;
      /* Get Language code for this transaction */
      &LNG = &MSG_RS.GetRow(&I).PSCAMA.LANGUAGE_CD.Value;
      &BASE_LNG = %Language;

      Evaluate &ACTION /*****************************/
         /******** Add a Record *******/
         /*****************************/

      When "A"
         /* Add the base language record */
         &REC_NAME_PREFIX = CreateRecord(Record.NAME_PREFIX_TBL);
         &REC.CopyFieldsTo(&REC_NAME_PREFIX);
         &REC_NAME_PREFIX.ExecuteEdits();
         If &REC_NAME_PREFIX.IsEditError Then
            /* error handling */

         Else
            &REC_NAME_PREFIX.Insert();
            /* Need error handling here if insert fails */
            If &LNG <> %Language Then
               /* add related language record */
               &RELLNG = &REC.RelLangRecName;
               &REC_RL = CreateRecord(Record.NAME_PREFIX_LNG);
               &REC.CopyFieldsTo(&REC_RL);
               &REC_RL.LANGUAGE_CD.Value = &LNG;
               &REC_RL.Insert();

               /* Need error handling here if insert fails */
            End-If;
         End-If;

         /*****************************/
         /***** Change a Record *******/
         /*****************************/
         /**** Using record objects ***/

      When "C"
         /* Get the Record - insert it */
         &KEY1 = &REC.GetField(Field.NAME_PREFIX).Value;
         &REC_NAME_PREFIX = CreateRecord(Record.NAME_PREFIX_TBL);

         &REC_NAME_PREFIX.NAME_PREFIX.Value = &REC.GetField(Field.
         NAME_PREFIX).Value;
         If &REC_NAME_PREFIX.SelectByKey() Then

            &REC.CopyFieldsTo(&REC_NAME_PREFIX);
            &REC_NAME_PREFIX.ExecuteEdits();
            If &REC_NAME_PREFIX.IsEditError Then
               /* error handling */
            Else
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
               &REC_NAME_PREFIX.Update();
            End-If;
         Else
            &REC.CopyFieldsTo(&REC_NAME_PREFIX);
            &REC_NAME_PREFIX.ExecuteEdits();
            If &REC_NAME_PREFIX.IsEditError Then
               /* error handling */
            Else
               &REC_NAME_PREFIX.Insert();
            End-If;
         End-If;

         /*****************************/
         /****** Delete a Record ******/
         /*****************************/
         /*** Using SQLExec ***********/

      When "D"
         /* Get the Record using SQLExec- error */
         &KEY1 = &REC.GetField(Field.NAME_PREFIX).Value;
         SQLExec("Select NAME_PREFIX from PS_NAME_PREFIX_TBL where
         NAME_PREFIX = :⇒1", &KEY1, &KEY2);
         If None(&KEY2) Then
            /* Send to error log */
         Else
            SQLExec("Delete from PS_NAME_PREFIX_TBL where
            NAME_PREFIX = :1", &KEY1);

            SQLExec("Delete from PS_NAME_PREFIX_LNG where
            NAME_PREFIX = :1", &KEY1);
         End-If;
      End-Evaluate;
   End-For;

end-method;
```

Message Class Inbound Asynchronous Example 3

There's a practical limit to how large a message can be, and this can be controlled by a system-wide variable called *%MaxMessageSize*. The system administrator can change this size in the PSOPTIONS settings. This size can be set only for all messages, not for individual definitions.

PeopleCode that populates a Message object should include code that is similar to the following example to check the message size before inserting a new Level 0 row.

Note. Always code the %MaxMessageSize checkpoint at the Level 0 record. A batch of transactions can be split across multiple messages, but *do not* split a single transaction (logical unit of work) into multiple messages.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local SQL &hdr_sql, &ln_sql;
Local Message &MSG;
Local Rowset &hdr_rs, &ln_rs;
Local Record &hdr_rec, &ln_rec, &hdr_rec_msg, &ln_rec_msg;

/* This PeopleCode will publish messages for a simple Header/
Line record combination. Multiple Header/Lines are copied to the
message until the %MaxMessageSize is exceeded at which point a
new message is built. This references MSR_HDR_INV (Header) and
DEMAND_INF_INV (Line) records */

/* Create an instance of the STOCK_REQUEST message */
&MSG = CreateMessage(OPERATION.STOCK_REQUEST);

/* Create an App. Message Rowset that includes the
MSR_HDR_INV (Header) and DEMAND_INF_INV (Line)*/
&hdr_rs = &MSG.GetRowset();

/* Create a SQL object to select the Header rows */
&hdr_sql = CreateSQL("Select * from PS_MSR_HDR_INV
       WHERE BUSINESS_UNIT='M04A1'
 AND ORDER_NO LIKE 'Z%' ORDER BY BUSINESS_UNIT,ORDER_NO");
&I = 1;

/* Create record objects for the Header and Lines */
&ln_rec = CreateRecord(Record.DEMAND_INF_INV);
&hdr_rec = CreateRecord(Record.MSR_HDR_INV);

/* Loop through each Header row that is fetched */
While &hdr_sql.Fetch(&hdr_rec)
  /* Publish the message if its size exceeds the MaxMessageSize
  /* specified in Utilities/Use/PeopleTools Options */
  If &MSG.Size > %MaxMessageSize Then
   %IntBroker.Publish(&MSG);
   &I = 1;
   /* Create a new instance of the message object */
   &MSG = CreateMessage(OPERATION.STOCK_REQUEST);
   &hdr_rs = &MSG.GetRowset();
  End-If;
  If &I > 1 Then
   &hdr_rs.InsertRow(&I - 1);
  End-If;
  /* Instantiate the row within the Header portion of the
   App Message rowset to which data will be copied */
  &hdr_rec_msg = &hdr_rs.GetRow(&I).GetRecord(Record.MSR_HDR_INV);
  /* Copy data into the level 0 (Header portion) of
  /* &MSG message structure */
  &hdr_rec.CopyFieldsTo(&hdr_rec_msg);

  /* Publish the last message if it has been changed*/
  If &hdr_rec_msg.IsChanged Then
   %IntBroker.Publish(&MSG);
  End-If;
End-While;
```

Message Class Inbound Asynchronous Example 4

The following code example shows how to get data out of the IBInfo object for a rowset-based message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Rowset &rs, &rs1;
Local Record &FLIGHTDATA, &REC;

Local string &acnumber_value, &msi_sensor_value, &ofp_value,
      &actype_value, &callsign_value, &squadron_value, &comm1_value,
      &comm2_value, &ecm_value, &datetime;
Local XmlDoc &xmldoc;
Local string &data;
Local boolean &return_bool_value;

&CRLF = Char(13) | Char(10);

/* this is how one would access data from IBinfo and
/* IBConnectorInfo objects*/
&return_bool_value = &MSG.IBInfo.ConnectorOverride;

For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfConnector
Properties()

   &data = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgName(&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgValue(&i);

End-For;

&MSG.IBInfo.IBConnectorInfo.ClearConnectorProperties();

&data = &MSG.IBInfo.IBConnectorInfo.ConnectorName;
&data = &MSG.IBInfo.IBConnectorInfo.ConnectorClassName;
&data = &MSG.IBInfo.IBConnectorInfo.RemoteFrameworkURL;
&data = &MSG.IBInfo.IBConnectorInfo.PathInfo;
&data = &MSG.IBInfo.IBConnectorInfo.Cookies;

For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfQueryStringArgs()

   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesName(&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesValue
     (&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesType(&i);

End-For;

&MSG.IBInfo.IBConnectorInfo.ClearQueryStringArgs();

&data = &MSG.IBInfo.MessageType;
&data = &MSG.IBInfo.RequestingNodeName;
&data = &MSG.IBInfo.OrigUser;
&data = &MSG.IBInfo.OrigNode;
&data = &MSG.IBInfo.AppServerDomain;
&data = &MSG.IBInfo.OrigProcess;
&data = &MSG.IBInfo.OrigTimeStamp;
&data = &MSG.IBInfo.DestinationNode;
&data = &MSG.IBInfo.FinalDestinationNode;
&data = &MSG.IBInfo.SourceNode;
&data = &MSG.IBInfo.MessageName;
&data = &MSG.IBInfo.MessageVersion;
&data = &MSG.IBInfo.VisitedNodes;

/* get the content data from the message rowset*/
&rs = &MSG.GetRowset();
&REC = &rs(1).QE_FLIGHTDATA;

&FLIGHTDATA = CreateRecord(Record.QE_FLIGHTDATA);
&REC.CopyFieldsTo(&FLIGHTDATA);
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
/* Parse out Message Data */
&acnumber_value = &FLIGHTDATA.QE_ACNUMBER.Value;
&msi_sensor_value = &FLIGHTDATA.QE_MSI_SENSOR.Value;
&ofp_value = &FLIGHTDATA.QE_OFP.Value;
&actype_value = &FLIGHTDATA.QE_ACTYPE.Value;
&callsign_value = &FLIGHTDATA.QE_CALLSIGN.Value;
&squadron_value = &FLIGHTDATA.QE_SQUADRON.Value;
&comm1_value = &FLIGHTDATA.QE_COMM1.Value;
&comm2_value = &FLIGHTDATA.QE_COMM2.Value;
&ecm_value = &FLIGHTDATA.QE_ECM.Value;
&datetime = &FLIGHTDATA.ACTIONDTTM.Value;

&outstring = "Send Async FLight test";

/* Construct Output String */
&outstring = &outstring | &acnumber_value | &CRLF | &msi_sensor_value
     | &CRLF | &ofp_value | &CRLF | &actype_value | &CRLF
     | &callsign_value | &CRLF | &squadron_value | &CRLF
     | &comm1_value | &CRLF | &comm2_value | &CRLF | &ecm_value |
        &datetime;

/* Log Output String into page record */
&FLIGHTDATA.GetField(Field.DESCRLONG).Value = &outstring;

SQLExec("DELETE FROM PS_QE_FLIGHTDATA");
&FLIGHTDATA.Insert();
```

Message Class Inbound Asynchronous Example 5

The following code example shows how to get data out of the IBInfo object for a nonrowset-based message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local XmlDoc &xmldoc;
Local XmlNode &node, &root, &acct_id_node, &acct_name_node,
     &address_node, &phone_node;
Local string &outstring, &CRLF;
Local Record &FLIGHT_DATA_INFO, &REC;

Local string &data;
Local boolean &return_bool_value;

/* this is how one wouild access data from IBinfo and
/* IBConnectorInfo objects*/

&return_bool_value = &MSG.IBInfo.ConnectorOverride;

For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfConnector
  Properties()

   &data = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgName(&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetQueryStringArgValue(&i);

End-For;

&MSG.IBInfo.IBConnectorInfo.ClearConnectorProperties();

&data = &MSG.IBInfo.IBConnectorInfo.ConnectorName;
&data = &MSG.IBInfo.IBConnectorInfo.ConnectorClassName;
&data = &MSG.IBInfo.IBConnectorInfo.RemoteFrameworkURL;
&data = &MSG.IBInfo.IBConnectorInfo.PathInfo;
&data = &MSG.IBInfo.IBConnectorInfo.Cookies;

For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfQueryStringArgs()

   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesName(&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesValue
     (&i);
   &data = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesType(&i);

End-For;

&MSG.IBInfo.IBConnectorInfo.ClearQueryStringArgs();

&data = &MSG.IBInfo.MessageType;
&data = &MSG.IBInfo.RequestingNodeName;
&data = &MSG.IBInfo.OrigUser;
&data = &MSG.IBInfo.OrigNode;
&data = &MSG.IBInfo.AppServerDomain;
&data = &MSG.IBInfo.OrigProcess;
&data = &MSG.IBInfo.OrigTimeStamp;
&data = &MSG.IBInfo.DestinationNode;
&data = &MSG.IBInfo.FinalDestinationNode;
&data = &MSG.IBInfo.SourceNode;
&data = &MSG.IBInfo.MessageName;
&data = &MSG.IBInfo.MessageVersion;
&data = &MSG.IBInfo.VisitedNodes;

&xmldoc = &MSG.GetXmlDoc();

&CRLF = Char(13) | Char(10);


&root = &xmldoc.DocumentElement;

/* Get values out of XMLDoc */
&node_array = &root.GetElementsByTagName("actype");
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
&ac_type_node = &node_array.Get(1);
&ac_type_value = &ac_type_node.NodeValue;

&node_array = &root.GetElementsByTagName("msi_sensor");
&msi_sensor_node = &node_array.Get(1);
&msi_sensor_value = &msi_sensor_node.NodeValue;

&node_array = &root.GetElementsByTagName("callsign");
&callsign_node = &node_array.Get(1);
&callsign_value = &callsign_node.NodeValue;

&node_array = &root.GetElementsByTagName("ofp");
&ofp_node = &node_array.Get(1);
&ofp_value = &ofp_node.NodeValue;

&outstring = "GetDataout of xmldoc Test";

&outstring = &outstring | &CRLF | &ac_type_value | &CRLF |
&msi_sensor_node
     | &CRLF | &callsign_value | &CRLF | &ofp_value;

/* Write out the result string */
SQLExec("DELETE FROM PS_QE_FLIGHT_DATA");
&FLIGHT_DATA_INFO = CreateRecord(Record.QE_FLIGHT_DATA);
&FLIGHT_DATA_INFO.GetField(Field.DESCRLONG).Value = &outstring;
&FLIGHT_DATA_INFO.Insert();
```

Message Class Inbound Asynchronous Example 6

The following example show a notification that could be an implementation of the OnNotify method, using a component interface in the notification. This example shows error trapping and has multilanguage support:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Component string &PUBNODENAME;
/* save pubnodename to prevent circular publishes */
Local Message &MSG;
Local Rowset &MSG_ROWSET, &cur_rowset;
Local ApiObject &oSession;
Local ApiObject &CONTACT_CI;
Local number &I;
Local string &KEY1;
Local Record &REC;
Local boolean &BC_CREATE, &ADD;
Local boolean &TRANSACTION_ERROR, &MSG_ERROR;
/** Transaction/Message Error Flags**/

Function errorHandler()
   Local ApiObject &oPSMessageColl;
   Local ApiObject &oPSMessage;
   Local string &strErrMsgSetNum, &strErrMsgNum, &strErrMsgText,
    &strErrType;
   &oPSMessageColl = &oSession.PSMessages;
   For &I = 1 To &oPSMessageColl.Count
      &oPSMessage = &oPSMessageColl.Item(&I);
      &strErrMsgSetNum = &oPSMessage.MessageSetNumber;
      &strErrMsgNum = &oPSMessage.MessageNumber;
      &strErrMsgText = &oPSMessage.Text;
      &LogFile.WriteLine(&strErrType | " (" | &strErrMsgSetNum | ","
          | &strErrMsgNum | ") - " | &strErrMsgText);
   End-For;
   rem ***** Delete the Messages from the collection *****;
   &oPSMessageColl.DeleteAll();
End-Function;

Function DO_CI_SUBSCRIBE()

   &oSession = %Session;

   &CONTACT_CI = &oSession.GETCOMPONENT(CompIntfc.CONTACT);
   If (&CONTACT_CI = Null) Then
      /* Replace this message with Tools message set when available */
      Error MsgGet(91, 58, " Unable to get the Component Interface.");
      Exit (1);
   End-If;

   /** Set Component Interface Properties **/
   &CONTACT_CI.GetHistoryItems = True;
   &CONTACT_CI.Interactivemode = False; /** set this to True while
    debugging **/
   rem Send messages to the PSMessage Collection;
   &oSession.PSMessagesMode = 1;

   &MSG_ERROR = False;

   For &I = 1 To &MSG_ROWSET.ActiveRowCount

      /** Set Session Language Code Property **/
      &REGIONALSETTINGS = &oSession.RegionalSettings;
      &REGIONALSETTINGS.LanguageCd = &MSG_ROWSET(&I).PSCAMA.
        LANGUAGE_CD.Value;

      &TRANSACTION_ERROR = False;
      &BC_CREATE = False;

      /** Instantiate Component Interface **/
      &KEY1 = &MSG_ROWSET(&I).CONTACT_TBL.PERSON_ID.Value;
      &CONTACT_CI.PERSON_ID = &KEY1;
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Evaluate &MSG_ROWSET(&I).PSCAMA.AUDIT_ACTN.Value
When = "A"
When = "N"
   &ADD = True;

   /* Check if Keys already exist. */
   &CONTACT_CIColl = &CONTACT_CI.Find();

   /*If None(&EXISTS) Then*/
   If &CONTACT_CIColl.Count = 0 Then
      If &CONTACT_CI.Create() Then
         &BC_CREATE = True;
      Else
         /* Replace this message with Tools message set
            when available */
         Warning MsgGet(18022, 56, "Error creating Component
         Interface for transaction %1", &I);
         &TRANSACTION_ERROR = True;
      End-If;
   Else
      If Not &CONTACT_CI.Get() Then
         /* Replace this message with Tools message set
            when available */
         Warning MsgGet(18022, 59, "Could not Get Component
         Interface for transaction %1", &I);
         &TRANSACTION_ERROR = True;
      End-If;
   End-If;
   Break;
When = "C"
   &ADD = False;
   If Not &CONTACT_CI.Get() Then
      /* Replace this message with Tools message set when
         available */
      Warning MsgGet(18022, 59, "Could not Get Component
      Interface for transaction %1", &I);
      &TRANSACTION_ERROR = True;
   End-If;
   Break;
When = "D"
When = "K"
When-Other
   /* delete and old key action codes not allowed at this
      time */
   &TRANSACTION_ERROR = True;
   Warning MsgGet(18022, 61, "Audit Action 'D' not allowed on
   transaction %1", &TRANSACTION);
   Break;
End-Evaluate;

&CONTACT_CI.CopyRowset(&MSG_ROWSET, &I);

If Not &TRANSACTION_ERROR Then
   If Not &CONTACT_CI.save() Then
      /* Replace this message with Tools message set when
         available */
      Warning MsgGet(18022, 57, "Error saving Component
      Interface for transaction %1", &TRANSACTION);
      &TRANSACTION_ERROR = True;
   End-If;
End-If;

/** close the last Component Interface in preparation for
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
            getting the next **/
         If Not &CONTACT_CI.Cancel() Then
            /* Replace this message with Tools message set when
               available */
            Warning MsgGet(18022, 58, "Error Canceling Component
            Interface for transaction %1", &TRANSACTION);
            Exit (1);
         End-If;

         /* Reset &TRANSACTION_ERROR to "False" for &BusComp.Save()
            to execute if no
         /*  Transaction Error found in the next Transaction.  */
         &TRANSACTION_ERROR = False;

      End-For;

      If &TRANSACTION_ERROR Then
         &MSG_ERROR = True;
      End-If;

End-Function;

/****  Main Process  ****/
&MSG.ExecuteEdits(%Edit_Required + %Edit_TranslateTable);
If &MSG.IsEditError Then
   &MSG_ERROR = True;
Else
   &PUBNODENAME = &MSG.PubNodeName;
   &MSG_ROWSET = &MSG.GetRowset();
   /* Do Component Interface subscribe */
   DO_CI_SUBSCRIBE();
End-If;

If &MSG_ERROR Then
   Exit (1);
End-If;
```

XmlDoc Class Inbound Asynchronous Example

The following example uses the GetXmlDoc method.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local XmlDoc &Document;
Local XmlNode &node, &root;
Local string &outstring;
Local Rowset &LEVEL0;
Local Record &SALES_ORDER_INFO, &REC;

&CRLF = Char(13) | Char(10);

& Document = &MSG.GetXmlDoc();

&root = & Document.DocumentElement;
&child_count = &root.ChildNodeCount;

/* Get values out of XmlDoc */
&node_array = &root.GetElementsByTagName("QE_ACCT_ID");
&acct_id_node = &node_array.Get(2);
&account_id_value = &acct_id_node.NodeValue;

&node_array = &root.GetElementsByTagName("QE_ACCOUNT_NAME");
&acct_name_node = &node_array.Get(2);
&account_name_value = &acct_name_node.NodeValue;

&node_array = &root.GetElementsByTagName("QE_ADDRESS");
&address_node = &node_array.Get(2);
&address_value = &address_node.NodeValue;

&node_array = &root.GetElementsByTagName("QE_PHONE");
&phone_node = &node_array.Get(2);
&phone_value = &phone_node.NodeValue;

&outstring = "GetMessageXmlDoc Test";
&outstring = &outstring | &CRLF | &account_id_value | &CRLF
     | &account_name_value | &CRLF | &address_value | &CRLF |
       &phone_value;

&SALES_ORDER_INFO = CreateRecord(Record.QE_SALES_ORDER);
&SALES_ORDER_INFO.GetField(Field.QE_ACCT_ID).Value =
     &account_id_value;
&SALES_ORDER_INFO.GetField(Field.DESCRLONG).Value = &outstring;
&SALES_ORDER_INFO.Update();
```

Handling Inbound Asynchronous Request/Response Transactions

Implement the OnNotify method in the PS_PT application package, located in the Integration subpackage, to handle inbound asynchronous request/response transactions.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class FLIGHTDATA_RETURN implements PS_PT:Integration:INotificationHandler
   method FLIGHTDATA_RETURN();
   method OnNotify(&MSG As Message);
end-class;

/* constructor */
method FLIGHTDATA_RETURN
end-method;

method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   /* Variable Declaration */


   Local string &str, &value;
   Local Rowset &rs;
   Local integer &num;

   Local Message &MSG, &MSG_resp;
   Local Record &FLIGHTDATA, &REC;

   &rs = &MSG.GetPartRowset(1);

  /* process request rowset */

   &MSG_resp = CreateMessage(Operation.FLIGHTPLAN_ARR, %IntBroker_Response);
   &rs = &MSG_resp.GetPartRowset(1);

   /* populate response rowset */


   &MSG_resp.IBInfo.WSA_MessageID = &MSG.IBInfo.WSA_MessageID;
   &MSG_resp.IBInfo.WSA_ReplyTo = &MSG.IBInfo.WSA_ReplyTo;
   &MSG_resp.IBInfo.WS_RequestAliasName = &MSG.IBInfo.WS_RequestAliasName;

   %IntBroker.Publish(&MSG_resp);
end-method;
```

Handling Inbound Synchronous Transactions

Implement the OnRequest method in the PS_PT application package, in the Integration subpackage, to handle inbound synchronous transactions. All the examples in this section are assumed to be implementations of the OnRequest method.

Message Class Inbound Synchronous Example

The following example implements both the OnRequest method and the OnError method

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IRequestHandler;

class RequestMan implements PS_PT:Integration:IRequestHandler
   method RequestMan();
   method OnRequest(&MSG As Message) Returns Message;
   method OnError(&MSG As Message) Returns string;
end-class;

/* constructor */
method RequestMan
   %Super = create PS_PT:Integration:IRequestHandler();
end-method;

method OnRequest
   /+ &MSG as Message +/
   /+ Returns Message +/
   Local Message &response;

   &response = CreateMessage(Operation.SYNC_REMOTE,
   %IntBroker_Response);

   &response.GetRowset().GetRow(1).GetRecord(Record.QE_FLIGHTDATA).
   GetField (Field.DESCRLONG).Value = &MSG.GenXMLString();

   Return &response;

end-method;

method OnError
   /+ &MSG as Message +/
   /+ Returns String +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnError +/
   Local integer &nMsgNumber, &nMsgSetNumber;
   Local string &sText;

   &nMsgNumber = &MSG.IBException.MessageNumber;
   &nMsgSetNumber = &MSG.IBException.MessageSetNumber;
   rem    &sText = &exception.DefaultText;
   &sText = &MSG.IBException.ToString();


   /* ADD SPECIFIC ERROR INFO HERE */
   Return &sText;

end-method;
```

XmlDoc Class Inbound Synchronous Example

The following example uses the GetXmlDoc method:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local XmlDoc &xmlRequestDoc;
Local XmlDoc &xmlResponseDoc;
Local XmlNode &select;
Local Message &Return_MSG;
Local array of XmlNode &whereClause;

Local string &recordName;
Local string &fieldName;
Local string &fieldValue;
Local Rowset &outputRowset;
Local boolean &return_bool_value;

&xmlRequestDoc = &MSG.GetXmlDoc();
&select = &xmlRequestDoc.DocumentElement;

&recordName = &select.GetAttributeValue("record");
&outputRowset = CreateRowset(@("Record." | &recordName));

&whereClause = &select.GetElementsByTagName("where");
If &whereClause <> Null And
   &whereClause.Len <> 0 Then
  &fieldName = &whereClause.Get(1).GetAttributeValue("field");
  &fieldValue = &whereClause.Get(1).GetAttributeValue("value");
  &outputRowset.Fill("WHERE " | &fieldName | "= :1", &fieldValue);
Else
  &outputRowset.Fill();
End-If;

&Return_MSG = CreateMessage(OPERATION.EXAMPLE, %Int⇒
Broker_Response);
&xmlResponseDoc  = &Return_MSG.GetXmlDoc();
&b = &xmlResponseDoc.CopyRowset(&outputRowset);
Return &Return_MSG;
```

SoapDoc Class Inbound Synchronous Example

The following example uses the GetXmlDoc method.

Note. Because GetXmlDoc returns an XmlDoc object, you must receive the inbound request message as an XmlDoc object, then convert it to a SoapDoc object to process it with SOAP methods.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local XmlDoc &request, &response;
Local string &strXml;
Local SoapDocs &soapReq, &soapRes;
Local Message &Response_Message;

&soapReq = CreateSoapDoc();

&request = &MSG.GetXmlDoc();
&soapReq.XmlDoc = &request;
&OK = &soapReq.ValidateSoapDoc();
&parmN = &soapReq.GetParmName(1);
&parmV = &soapReq.GetParmValue(1);

&Response_Message = CreateMessage(OPERATION.SoapExample,
%IntBroker_Response);

&response  = &Response_Message.GetXmlDoc();
&soapRes = CreateSoapDoc();
&soapRes.AddEnvelope(0);
&soapRes.AddBody();
&soapRes.AddMethod("StockPrice", 1);
&soapRes.AddParm(&parmN, &parmV);
&soapRes.AddParm("Price", "29");
&OK = &soapRes.ValidateSoapDoc();

&response = &soapRes.XmlDoc;
Return &Response_Message;
```

Simulating Receiving Messages from External Nodes

You can use PeopleCode to simulate receiving asynchronous messages from external nodes, including running transformations.

Use can use the IntBroker class InboundPublish method to work with rowset-based and nonrowset-based messages.

The following example shows an inbound publish as part of an OnNotify method implementation with a rowset-based message:

```
Local Message &MSG_REMOTE;
Local Rowset &rs;

&rs = &MSG.GetRowset();
/*create the message to be re-published from a simualted remote node*/

&MSG_REMOTE = CreateMessage(OPERATION.QE_FLIGHTPLAN);

&MSG_REMOTE.IBInfo.RequestingNodeName = "QE_IBTGT";

&MSG_REMOTE.IBInfo.ExternalOperationName = &MSG_REMOTE.OperationName | "." |

&MSG_REMOTE.OperationVersion;

&MSG_REMOTE.CopyRowset(&rs);

&Ret = %IntBroker.InBoundPublish(&MSG_REMOTE);
```

The following example shows an inbound publish as part of an OnNotify implementation with a nonrowset-based message:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &MSG_REMOTE;
Local XmlDoc &xmldoc;
Local Rowset &rs;

&xmldoc = &MSG.GetXmlDoc();
/*create the message to be re-published from a simualted remote node*/

&MSG_REMOTE = CreateMessage(OPERATION.QE_FLIGHTPLAN);

/* populate the Remote Message with data to be re-published*/

&MSG_REMOTE.SetXmlDoc(&xmldoc);

%IntBroker.InBoundPublish(&MSG_⇒
REMOTE, Node.REMOTE_NODE);
```

Processing Inbound Errors

This section discusses how to:

- Validate data.
- Use the Exit built-in function.
- Correct messaging errors.

Validating Data

You validate data differently depending on the PeopleCode class that you're using to receive the message.

XMLDoc Class Validation

You can validate incoming XML DOM-compliant messages by using the XmlDoc document type definition (DTD) that is delivered with your PeopleSoft application.

See *PeopleTools 8.52: PeopleCode API Reference*, "XmlDoc Classes."

SoapDoc Class Validation

You can validate SOAP-compliant messages by using the ValidateSoapDoc method in the PeopleCode SoapDoc class.

See *PeopleTools 8.52: PeopleCode API Reference*, "SOAPDoc Class."

Message Class Validation

Have a message receiving process validate incoming data by:

- Using the ExecuteEdits method in the code to invoke the definitional edits.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Calling PeopleCode validation built-in functions (if they already exist, for example in a FUNCLIB record, or if validation logic can be encapsulated within a small set of built-in functions) from within the receiving PeopleCode.
- Calling a component interface or Application Engine program from the receiving process (for complex validation logic).

  This enables you to reuse logic that is embedded in the component.

The ExecuteEdits method uses the definitional edits to validate the message. You can specify the following system variables alone or in combination. If you don't specify a variable, all of the edits are processed.

- %Edit_DateRange
- %Edit_OneZero
- %Edit_PromptTable
- %Edit_Required
- %Edit_TranslateTable
- %Edit_YesNo

The following example processes all edits for all levels of data in the message structure:

```
&MYMSG.ExecuteEdits();
```

The following example processes the Required Field and Prompt Table edits:

```
&RECPURCHASEORDER.ExecuteEdits(%Edit_Required +
 %Edit_PromptTable);
```

ExecuteEdits uses *set processing* to validate data. Validation by using a component interface or a PeopleCode built-in function is usually done with row-by-row processing. If a message contains a large number of rows per rowset, consider writing the message to a staging table and calling an Application Engine program to do set processing if you want additional error checking.

ExecuteEdits sets several properties on several objects if there are any errors:

- IsEditError is set on the Message, Rowset, Row, and Record objects if any fields contain errors.
- EditError, MessageNumber, and MessageSetNumber are set on the Field object that contains the error.

If you don't want to use ExecuteEdits, you can set your own errors by using the field properties. Setting the EditError property to True automatically sets the IsEditError message property to True. You can also specify your own message number, message set number, and so on, for the field. If you use the Exit(1) built-in function, the message status changes to Error when you finish setting the fields that are in error.

Using the Exit Built-in Function

Use the Exit built-in function to invoke a messaging error process when the application finds an error. This works only when you use the PeopleCode Message class to process inbound transactions. The same error processing is invoked automatically if PeopleTools finds an unexpected error, such as a Structured Query Language (SQL) error. The Exit built-in function has an optional parameter that affects how the error is handled.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

To handle error processing in application tables, use the Exit built-in function with no parameter, or just let the notification process finish normally. This marks the message receipt as successful and commits the data.

To handle the error tracking and correction with PeopleSoft Integration Broker, use the Exit built-in function with 1 as a parameter to log the errors, perform a rollback, and stop processing.

Using the Exit Built-in Function Without Parameters

In the `Exit( )` form (that is, Exit without any parameters specified), all data is committed and the message is marked as complete. Use this to indicate that everything processed correctly and to stop program processing. Note, though, that the message status is set to Complete; therefore, you can't detect or access errors in the Service Operations Monitor. If errors did occur, the subscription code should write them to a staging table, and then you must handle all of the error processing.

The Exit built-in function:

- Sets the message status in the application message queue for the subscription to Done.
- Commits the transaction.
- Stops processing.

Following is an example of using Exit without a parameter:

```
&MSG.ExecuteEdits();
If &MSG.IsEditError then
  App_Specific_Error_Processing();
  Exit();
Else
  Specific_Message_Processing();
End-if;
```

Using the Exit Built-in Function with Parameters

When you supply a 1 as a parameter for the Exit built-in function, the function:

- Processes a rollback.
- Sets the message status in the message queue for the subscription to Error.
- Writes the errors to the subscription contract error table.
- Stops processing.

You can view all errors that have occurred for this message in the Service Operations Monitor, even those errors that are detected by ExecuteEdits.

Following is an example of using the Exit function with 1 as a parameter:

```
&MSG.ExecuteEdits();
If &MSG.IsEditError then
  Exit(1);
Else
  Process_Message();
End-if;
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook

Using Message Object Functionality With Nonrowset-Based Messages

Prior to the PeopleTools 8.44 release, there were two distinct paths for writing and executing PeopleCode for PeopleSoft Integration Broker which were based on whether you were working with rowset-based XML messages or nonrowset-based XML messages.

For rowset-based XML messages, you could use a rowset and all the properties and methods associated with the Message class. For nonrowset-based XML messages, you could not use the Message class, but instead used built-in functions such as PublishXmlDoc and GetMessageXmlDoc. In addition, when working with nonrowset-based messages and these built-in functions, you could only access content data.

Effective with the PeopleTools 8.44 release, when working with nonrowset-based XML messages you can use *all* of the functionality of the Message object using two new methods, SetXMLDoc and GetXMLDoc.

SetXMLDoc	Use this method to load and pass nonrowset-based data into the Message object.
GetXMLDoc	Use this method to get nonrowset-based data out of the message object.

Using the SetXMLDoc Method

The following example shows how to use SetXMLDoc to use the Message object to publish a nonrowset-based message.

```
//&XmlDoc holds the nonrowset-based data as before.

// create an instance of the Message object
 &MSG = CreateMessage(OPERATION.QE_F18_ASYNC_XMLDOC);
// Load the Message object with the xmldoc data.
&MSG.SetXmlDoc(&XmlDoc);
// perform a publish for the nonrowset-based message
%IntBroker.Publish(&MSG);
```

Using the GetXMLDoc Method

The following code example shows how to use GetXMLDoc to get nonrowset-based XML out of the Message object.

```
Local XMLDOC &XmlDoc;

// get an xmldoc object loaded with the content data.
&XmlDoc = &MSG.GetXmlDoc();
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleCode API Reference, "Message Classes"

Generating Test Messages

Use the Handler Tester utility to generate test messages.

See *PeopleTools 8.52: Integration Testing Utilities and Tools PeopleBook*

Working With Message Segments

This section provides an overview of message segments and discusses how to:

- Configure nodes to handle segmented messages.
- Create message segments.
- Delete message segments.
- Send and receive segmented messages between PeopleSoft systems.
- Send and receive segmented messages to/from third-party systems.
- Send and receive large segmented messages using parallel processing.
- Access message segments.
- View message segment data.
- Use restartable processing for publishing large messages in batch.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Tuning Messaging System Performance," Sending and Receiving Large Segmented Messages Using Parallel Processing

Understanding Message Segments

When you create message segments, you can divide rowset-based and nonrowset-based messages into multiple data containers, or segments, for sending. Depending on the order in which you send a message that contains message segments, the receiving system can process the message as a whole, or process one segment at a time while the others are compressed in memory or held in the application database.

As a result creating message segments can enhance system performance and message exchange, especially when you are working with large messages that exceed one gigabyte (1 GB).

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

To create and manage message segments, you use several methods and properties of the PeopleCode Message class.

Understanding PeopleCode used to Work with Message Segments

This section discusses:

- Methods used with message segments.
- Properties used with message segments.

Methods Used with Message Segments

The following table lists the PeopleCode methods you can use when you work with message segments.

Method	*Class*	*Description*
CreateNextSegment	Message	Designates the end point of one segment and the beginning of a new segment.
DeleteOrphanedSegments	IntBroker	Used to delete segments that might have been orphaned if you were processing message segments using a PeopleSoft Application Engine program that had to be restarted.
DeleteSegment	Message	Deletes a segment.
GetSegment	Message	Gets the segment specified by the passed value. The passed value is the segment number.
UpdateSegment	Message	Use this method to update data within the current segment.

Note. Use the DeleteSegment and UpdateSegment methods only when storing segments data in memory. These methods do not function when segment data is stored in the database.

Properties Used with Message Segments

The following table lists PeopleCode properties that you can use when you work with message segments.

Property	*Class*	*Description*
CurrentSegment	Message	Returns a number, indicating which segment is the current segment.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Property	Class	Description
SegmentsUnOrder	IBInfo	Determines whether to process message segments in order or unordered. This property pertains to asynchronous messages only. The values are: • *True*: Process message segments unordered. • *False*: Process message segments in order. (Default.)
SegmentCount	Message	Returns the total number of segments in a message.
SegmentsByDatabase	Message	Enables you to override where message segment data is stored for a message. The values are: • *True*: Store message segments awaiting processing in the application database. • *False*: Store message segments awaiting processing in memory. (Default.)

See Also

PeopleTools 8.52: PeopleCode API Reference PeopleBook

Configuring Nodes to Handle Segmented Messages

This section describes how to configure nodes to handle segmented messages.

Understanding Configuring Nodes to Handle Segmented Messages

Before you can send segmented messages, you must configure the remote node defined on the local system to handle segmented messages by setting the Segment Aware option on the Node Definitions page in the PeopleSoft Pure Internet Architecture.

Warning! Do not set the Segment Aware option for remote PeopleSoft 8.45 or earlier nodes, or for third-party systems. If you do so, the receiving system will consume only the first segment of the messages and ignore any subsequent segments.

Configuring a Node to Handle Segmented Messages

To configure a node to handle segmented messages:

1. Select PeopleTools, Integration Broker, Integration Setup, Node Definitions.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Select a node with which to work and click OK.

   The Node Definitions page appears.

3. Select the Segment Aware box.

4. Click the Save button.

Creating Message Segments

This section provides an overview of creating message segments and message segment numbers and discusses how to:

- Create message segments.
- Count the number of segments in messages.
- Store message segments awaiting processing.
- Override where to store message segment awaiting processing.
- Specify the order in which to process message segments.
- Chunk asynchronous segmented messages.

Understanding Creating Message Segments

By default every message has one segment.

To create multiple message segments use the CreateNextSegment method in the location in the message where you want one segment to end and next segment to begin. Continue this process until you have created the desired number of segments for the message.

Segments can contain any number of rowsets of data (rowset-based messages) or rows of data (nonrowset-based messages).

Understanding Message Segment Numbers

When you create a message segment, PeopleSoft Integration Broker assigns a message segment number to the segment.

The first message segment has a message segment number or 1, and message segment numbers are increment by one sequentially thereafter. As an example, if you break a message into three segments, the first segment number is *1*, the second segment number is *2*, and the third segment number is *3*.

Creating Message Segments

The following example shows using the CreateNextSegment method to create three segments in the message QE_FLIGHTPLAN, populating each segment with data from the component buffer.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
&MSG = CreateMessage(OPERATION.QE_FLIGHTPLAN);

&rs=&MSG.GetRowset();
//Now populate rowset
// End of first segment. Beginning of second segment.
&MSG.CreateNextSegment();

&rs=&MSG.GetRowset();
//Now populate rowset
//End of second segment. Beginning of third segment.
&MSG.CreateNextSegment();

&rs=&MSG.GetRowset();
//Now populate rowset

%IntBroker.Publish(&MSG);
```

Counting the Number of Segments in Messages

You might have the need to determine the number of segments in a message. Use the SegmentCount property to determine this information.

Storing Message Segments Awaiting Processing

By default, message segments awaiting processing are stored in memory until all segments are processed. Once all segments are processed, PeopleSoft Integration Broker sends all data as one message.

Use the MessageSegmentFromDB parameter in PSAdmin to specify the number of segments to keep in memory before writing segmented messages to the database. The default value is *10.*

For synchronous messages, if the number of segments sent for processing exceeds the set for the MessageSegmentsFromDB parameter, an error occurs.

Overriding Where to Store Message Segments Awaiting Processing

You can override the number of segments to keep in memory before writing segmented messages to the database for a single message using the SegmentsByDatabase property of the Message class.

Storage Location	*Description*
Memory	When message segments are stored in memory, PeopleSoft Integration Broker writes all segments as one message to the database when you send the message. To store message segment data in memory, set the SegmentsByDatabase property to *False.* (Default.)
Application database	When message segments are stored in the database, PeopleSoft Integration Broker writes the segments to the database individually. When you store message segments in the database you can have an infinite number of segments in a message. To store message segment data in the application database, set the SegmentsByDatabase property to *True.*

When you store message segments in memory, the number of segments is limited by the value set in the MessageSegmentFromDB parameter in PSAdmin in the Setting for PUB/SUB servers section of the file.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When working with asynchronous messages, if you create more message segments then the value set, all segments are written to the database automatically and the SegmentsByDatabase property will automatically be set to True.

For synchronous messages, attempting to create more segments then the specified value will result in an error message.

Specifying the Order in Which to Process Message Segments

When you work with segmented asynchronous messages you can specify that PeopleSoft Integration Broker process the segments in order or unordered, using the SegmentsUnOrder property of the Message class.

Message Segment Processing	Description
In order	When Integration Broker processes message segments in order, it decompresses all message segments sequentially and then processes the message as a whole. In this situation, only one publication or subscription contract is created. To process message segment in order, set the SegmentsUnOrder property to *False.*
Unordered	When Integration Broker processes message segments unordered, it decompresses and processes all segments in parallel. In this situation, the system creates one publication or subscription contract for each message segment. To process message segment unordered, set the SegmentsUnOrder property to *True.*

If you attempt to send ordered segmented messages to a node that is not segment aware an error message will be created and can be viewed on the Message Errors tab on the Message Details page in Service Operations Monitor.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook*

Chunking Asynchronous Segmented Messages

Chunking asynchronous segmented messages sends message in blocks to the receiving node.

When using chunking, message instances display in *Hold* status in the Service Operations Monitor until all chunks are received. Once all chunks are received, the message status switches to *New.*

Note. Chunking applies to ordered asynchronous messages only.

The number of segments to chunk for an asynchronous message is determined by the value you set for the MessageSegmentByDatabase parameter in PSAdmin. The default value is *10.*

As an example, if a message has 20 segments and you set MessageSegmentByDatabase to *5,* PeopleSoft Integration Broker will send four groups (array of messages) of segments to the integration gateway, and each group will contain five segments.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Deleting Message Segments

You can delete message segments in a message only before you publish the message.

Use the DeleteSegment method of the Message class to perform the action.

You cannot delete the first segment in a message.

The following example demonstrates using the DeleteSegment method in an implementation of the OnRequestSend method.

```
import PS_PT:Integration:ISend;

class Send implements PS_PT:Integration:ISend
   method Send();
   method OnRequestSend(&message As Message) Returns Message;
   method OnError(&message As Message)
end-class;

/* constructor */
method Send
   %Super = create PS_PT:Integration:ISend();
end-method;

method OnRequestSend
   /+ &message as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnRequestSend +/
   Local integer &segment_number, &i;
   Local Rowset &rs;

   For &i = 1 To &message.SegmentCount
      &rs = Null;
      &message.GetSegment(&i);

      &rs = &message.GetRowset();

      /* determine that segment 3 needs to be deleted. */
      &segment_number = &i;

   End-For;

   &message.DeleteSegment(&segment_number);


   Return &message;
end-method;

method OnError
   /+ &message as Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnError +/

end-method;
```

Sending and Receiving Segmented Messages between PeopleSoft Systems

This section discusses how to:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Send segmented messages to PeopleSoft systems.
- Receive segmented messages from PeopleSoft systems.

Sending Segmented Messages to PeopleSoft Systems

To send a segmented message, use sending PeopleCode and events as you would with any other message.

Use the PeopleSoft target connector when the receiving node is a PeopleSoft system. The PeopleSoft target connector automatically handles message segments, and no additional configuration is required on the connector.

Before sending a transaction with message segments, on the sending PeopleSoft system, be sure that the Segment Aware box is selected for the remote node that represents the receiving system.

Receiving Segmented Messages from PeopleSoft Systems

To receive segmented message from PeopleSoft systems, use notification PeopleCode or implement the OnRequest method.

Use the PeopleSoft listening connector to receive transactions that contain message segments from other PeopleSoft systems. The PeopleSoft listening connector automatically handles message segments, and no additional configuration is required on the connector.

Sending and Receiving Segmented Messages to/from Third-Party Systems

This section discusses how to:

- Send segmented messages to third-party systems.
- Receive segmented messages from third-party systems.

Understanding DataChunkCount and DataChunk Properties

PeopleSoft Integration Broker uses two properties to communicate to sending and receiving systems the number of message segments that are contained in a transaction:

DataChunkCount	Indicates the total number of data chunks or message segments contained in the transaction.
DataChunk	Indicates the number of the data chunk or message segment that you are sending. For example, if there are a total of seven data chunks in the transaction, and the current segment is the third chunk, the DataChunk value for the current message is *3*.

Note that when you are sending and receiving message segments between PeopleSoft systems these properties are not used. The PeopleSoft target and listening connectors perform all necessary processing.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Sending Segmented Messages to Third-Party Systems

To send segmented messages from PeopleSoft systems to third-party system, use one of the following target connectors:

- AS2 target connector
- HTTP target connector
- JMS target connector
- SMTP target connector

No additional target connector configuration is required to send segmented messages. These connectors read the messaging PeopleCode on the integration gateway and determine the number of segments contained in the transaction. They then populate the DataChunkCount and DataChunk parameters and include this information with each outbound segment sent. All of these connectors except for the HTTP target connector send the DataChunkCount and DataChunk information in the message header of each outbound message segment. The HTTP target connector includes the DataChunkCount and DataChunk parameter information in the HTTP header of each outbound message segment.

Before sending a transaction with message segments, on the PeopleSoft system, be sure that the Segment Aware box is selected for the remote node that represents the third-party integration partner.

Receiving Segmented Messages from Third-Party Systems

At this time, only the HTTP listening connector can be used to receive message segments from third-party systems.

To receive segmented messages with third-party integration partners, the third-party must specify the following DataChunkCount and DataChunk parameters in the HTTP properties, query arguments, or SOAP header:

The receiving PeopleSoft system must use the HTTP listening connector as only this connector monitors transactions for these parameters.

After the third party sends in the first segment, the PeopleSoft system sends an acknowledgement to the third-party system. The acknowledgment contains a transaction ID that the third-party integration partner must include with all subsequent segments.

The following bullet points describe sample processing for a third-party integration partner sending a transaction to a PeopleSoft system that contains three segments:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. First segment processing:

    a. The third-party integration partner prepares the first message/segment of the transaction. In the HTTP properties, query string, or SOAP header, it sets the DataChunk equal to *1* indicating the first chunk, and sets the DataChunkCount equal to *3* indicating total number of chunks to be sent for the transaction.

    b. When the request is received by the PeopleSoft system the data chunk is saved in the database as a segment.

    c. In the Service Operations Monitor the transaction displays a status of *Hold.*

    d. The PeopleSoft system sends an acknowledgement to the third-party system, which includes a transaction ID.

    Note. The third-party integration partner must include the transaction ID as part of all subsequent requests for the transaction. The PeopleSoft system uses the transaction ID to identify the segments that belong to the transaction.

    ---

2. Second segment processing:

    a. The third-party integration partner prepares the second message/segment of the transaction. In the HTTP properties, query string, or SOAP header, it sets the DataChunk equal to *2* indicating that the message is the second chunk, and sets the DataChunkCount equal to *3* indicating total number of chunks to be sent for the transaction. It also specifies the transaction ID sent by the PeopleSoft system in the acknowledgement for the first segment.

    b. When the request is received by the PeopleSoft system the data chunk is saved in the database as a segment.

    c. In the Service Operations Monitor the transaction displays a status of *Hold.*

3. Third segment processing:

    a. The third-party integration partner prepares the third message/segment of the transaction. In the HTTP properties, query string, or SOAP header, it sets the DataChunk equal to *3* indicating that the message is the third chunk, and sets the DataChunkCount equal to *3* indicating total number of chunks to be sent for the transaction. It also specifies the transaction ID sent by the PeopleSoft system in the acknowledgement for the first segment.

    b. When the request is received by the PeopleSoft system the data chunk is saved in the database as a segment.

    c. Since the PeopleSoft system has received all of the segments in the transaction, in the Service Operations Monitor the transaction displays a status of *New.*

    d. The PeopleSoft system processing the transaction like any other transaction at this point.

The PeopleCode to read the data chunks/segments is the Message Segment API.

Sending, Receiving, and Correlating Multiple Segmented Messages

Previous sections in this chapter have discussed sending one message that contains multiple message segments. For very large messages this can have performance impact due to the large number of segments.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleSoft provides PeopleCode that allows you to send multiple messages with multiple segments and then correlate them into one transaction on the receiving system. So instead of sending one message with 50 message segments, you can send 10 messages with 5 message segments using parallel processing, and then correlate the 10 messages on the receiving system.

The InitializeConversationId property on the Message object, provides the correlation between messages. The FirstCorrelation method on the Message object ensures that the database table is truncated after receipt of the first message only.

On the first message to be published set the InitializeConversationId property to *True.* After the message is published retrieve the transaction ID from the message. For all subsequent messages, set the CorrelationID property to the value of the transaction ID returned from the first message. As a result, when the messages arrive at the receiving system they have different transaction IDs, but all have the same correlation ID.

On the receiving system when the first message is received the database table is truncated. To prevent a destructive load from occurring with the receipt and processing of each subsequent message, use the PreNotify event. You can use the PreNotify event to truncate the database table upon receipt of the first message. In subsequent messages use the FirstCorrelation method in the event, setting the method to *True*, to determine if a prior message with the same correlation ID has already run the event.

The following example shows an example of how the sending system uses the InitializeConversationId property:

```
/*First Message to Publish */

&MSG. InitializeConversationId = true;
%IntBroker.Publish(&MSG);
&strCorrelationID  =  &MSG.TransactionId;

/* all subsequent message to correlate*/

&MSG. IBInfo.ConversationID = &strCorrelationID;
%IntBroker.Publish(&MSG);
```

The following example shows an example of how the receiving system uses the FirstCorrelation method in the PreNotify event:

```
PreNotify Event:

If &MSG.FirstCorrelation() = true Then
/* process the event logic */

End-If;
```

Accessing message segments is described elsewhere in this section.

See Chapter 7, "Sending and Receiving Messages," Accessing Segments in Messages, page 189.

See Also

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," FirstCorrelation

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," InitializeConversationId

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Tuning Messaging System Performance," Sending and Receiving Large Segmented Messages Using Parallel Processing

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Accessing Segments in Messages

After you receive a segmented message, use the GetSegment method of the Message class to access message segment data.

After you access a message segment, use the Message class GetRowset or GetXmlDoc methods to work with the contents of the segment.

Warning! You can access only one segment in a message at a time. When you access a message segment, PeopleSoft Integration Broker removes the previously accessed message segment from memory.

When you access a message segment, set the existing rowset to null to eliminate storing multiple rowsets in the data cache.

The following example shows using the GetSegment method to access a message segment in the message QE_FLIGHTDATA.

```
For &i = 1 To &MSG.SegmentCount
   &rs = Null;  //Null the rowset to remove it from memory
   &MSG.GetSegment(&i);

   &rs = &MSG.GetRowset();
   &REC = &rs(1).QE_FLIGHTDATA;

   &FLIGHTDATA = CreateRecord(Record.QE_FLIGHTDATA);
   &REC.CopyFieldsTo(&FLIGHTDATA);

   /* Parse out Message Data */
   &acnumber_value = &FLIGHTDATA.QE_ACNUMBER.Value;
   &msi_sensor_value = &FLIGHTDATA.QE_MSI_SENSOR.Value;
   &ofp_value = &FLIGHTDATA.QE_OFP.Value;
   &actype_value = &FLIGHTDATA.QE_ACTYPE.Value;
   &callsign_value = &FLIGHTDATA.QE_CALLSIGN.Value;
   &squadron_value = &FLIGHTDATA.QE_SQUADRON.Value;
   &comm1_value = &FLIGHTDATA.QE_COMM1.Value;
   &comm2_value = &FLIGHTDATA.QE_COMM2.Value;
   &ecm_value = &FLIGHTDATA.QE_ECM.Value;

   &outstring = "Send Async Flight test";

   /* Construct Output String */
   &outstring = &outstring | &acnumber_value | &CRLF |
   &msi_sensor_value | &CRLF | &ofp_value | &CRLF | &actype_value |
   &CRLF | &callsign_value | &CRLF | &squadron_value | &CRLF |
   &comm1_value | &CRLF | &comm2_value | &CRLF | &ecm_value;

   /* Log Output String into page record */
   &FLIGHTDATA.GetField(Field.DESCRLONG).Value = &outstring;

   SQLExec("DELETE FROM PS_QE_FLIGHTDATA");
   &FLIGHTDATA.Insert();

End-For;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Message Segment Data

The Service Operations Monitor Message Details page provides information about messages that contain segments.

See Also

PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook

Using Restartable Processing for Publishing Large Messages in Batch

This section provides an overview, prerequisites and setup steps for using restartable processing for publishing large asynchronous segmented messages in batch.

Understanding Using Restartable Processing

PeopleSoft provides a PeopleSoft Application Engine library module, IB_SEGTEST, that you can use as a template to create a module to aid in processing large messages and messages in batch for outbound asynchronous PeopleSoft Integration Broker segment data with restart capability.

With restart capability, if there is an abnormal program termination, you can correct any data errors and continue processing from the point of the last commit without having to reload message segment data from the beginning.

Understanding the IB_SEGTEST Application Engine Library Module

This section provides overview information for using the IB_SEGTEST

The IB_SEGTEST library module consists of three sections:

- Section 1: Section1. The main processing section.
- Section 2: ABORT. Use to trigger a user abort of the running application engine program
- Section 3: CLEANSEG. An independent section you can call to clean up pending segment data that had been committed to the database but is no longer to be used.

Prerequisites

To use the information provided in this section, you should have a thorough understanding of PeopleSoft Application Engine.

Using the IB_SEGTEST Library Module

This section provides an overview of the high-level list of tasks to perform to set up a PeopleSoft Application Engine program to perform restartable message processing.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Make a copy of IB_SEGTEST, including all sections and PeopleCode.

   From here on, the copy of the application engine library module is referred to as IB_SEGTEST1, but you can use any name you choose.

2. In the State Records tab of IB_SEGTEST1, verify that PSIBSEGRSTR_AET is the default state record. Replace PT_EIP_ERR_AET with whatever state record is used in the main application engine program that will be calling the Library module.

   Note that IB_SEGTEST1 is flagged as not restartable. Since database commits will be performed in the middle of PeopleCode processing, the only way the commits can take effect is if the module is flagged as not restartable.

3. The application engine program used to call IB_SEGTEST1 should be restartable.

   Always issue a commit in the step prior to calling the library module IB_SEGTEST1.

4. In the application engine program that will be calling IB_SEGTEST1, insert a step to call IB_SEGTEST1, section Section1. Insert the step at the point in time when you want to do the message publish. You must issue a commit prior to calling this section, otherwise there will be a 'Unable to Process Commit' error issued from within IB_SEGTEST1.

5. Add PSIBSEGRSTR_AET as an additional state record to the calling application engine program.

6. Since both programs now share state records, when IB_SEGTEST1 is called, all state record values will be passed on to the called module. Presumably all application values needed to extract application data would be stored in the application state record.

7. Modify the PeopleCode in IB_SEGTEST1.Section1. Several comments have been added to the code to aid in the modifications. Note the following:

   - Change `&MSG = CreateMessage(OPERATION.QE_FLIGHTPLAN)` to create whatever message will be used.
   - SegmentsByDatabase should always be set to True.
   - The While loop is used to simulate application code processing large volumes of data. This can be changed to meet application needs. However, pay close attention as to when commits are issued, when state records are updated, when new segments are created, and finally, when the message publish is executed. The order of these events is crucial to proper workability. In the sample program, also note how to break out of the While loop.
   - Note the location where the application state record needs to be updated. A comment instructs in the PeopleCode provides instructions on where to perform this task.
   - Do not remove the `Exit(1)` from the end of the PeopleCode. This is necessary to bypass the Abort action that is coded into the same Step.
   - If in the middle of processing, the application code determines that an abort needs to be triggered, an `Exit(0)` can be coded. This triggers the Abort step to be called, which will terminate application engine processing. A restart could then be issued if processing needs to continue.

     If you determine that a message no longer needs to be published, the calling application engine program could then call the CLEANSEG step to get rid of all the pending data that has been saved in the database. Alternatively, the Abort step could be modified to call CLEANSEG if on any abort, no old data is to be kept.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleSoft Application Engine PeopleBook

Populating and Retrieving Document Data

This section discusses how to:

- Instantiate Documents objects.
- Populate document data.
- Retrieve document data.

Understanding Populating and Retrieving Document Data

This section provides guidelines for instantiating Document objects, and populating and retrieving document data from document message types.

Memory Management in Message Segments that Use Documents

Following these guidelines when populating or retrieving document data from message segments:

- After your code to populate a segment, set the following statement to *NULL*:

  ```
  &DOC = null;
  ```

  Including this statement in your code releases the segment data from memory. If you do not include this statement in your code, data from all segments accumulates in memory.

- Start each new segment processing section by setting the following statement to *TRUE:*

  ```
  &DOC = &MSG.GetDocument(true)
  ```

  Including this statement in your code guarantees that only one segment is in memory at a given time.

Instantiating Document Objects

This section discusses and provides examples for how to:

- Instantiate Document objects using package, name, and version.
- Instantiate Document objects using document keys.
- Instantiate Document objects for document message types.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Instantiating Document Objects Using Package, Name, and Version

You can instantiate a Document object using the CreateDocument built-in function. Using this approach, you use the Create Document built-in function, and then specify the document package name, the document name, and the document version, as shown in the following example:

```
&DOC = CreateDocument("Purchasing", "PurchaseOrder", "v1");
```

Instantiating Document Objects Using Document Keys

The following code example shows instantiating a Document object using the a document key. This approach uses the CreateDocumentKey and CreateDocument built-in functions.

First you instantiate a Document Key object, using the CreateDocumentKey built-in function and passing in the document package name, document name, and document version. You then instantiate a Document object using the CreateDocument built-in function, and pass in the document key.

```
&DOCKEY = CreateDocumentKey("Purchasing", "PurchaseOrder", "v1");
&DOC = CreateDocument(&DOCKEY);
```

Instantiating Document Objects for Document Message Types

The following example shows how to instantiate a document object when the document is being used as a message type.

```
&DOC = &MSG.GetDocument();
```

Populating Document Data

This section discusses and provides examples for how to:

- Populate documents from messages.
- Populate document using rowsets.
- Populate message segments with documents.

Populating Documents from Messages

The following example shows how to use the CreateMessage built-in function to populate a document from a message:

```
&MSG = CreateMessage(Operation.PURCHASE_ORDER);
&DOC = &MSG.GetDocument();
```

Populating Documents Using Rowsets

The following example shows using the CreateMessage built-in function and the Document class to populate a document using a rowset:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &MSG;
Local Document &DOC;

&MSG = CreateMessage(Operation.PURCHASE_ORDER);
&DOC = &MSG.GetDocument();

/* Get Rowset */
&Rowset = &DOC.GetRowset();

/* populate rowset like any other rowset

/* update document with popualted rowset * /
&nRet = &DOC.UpdateFromRowset(&Rowset);

If (&nRet) = True Then
   %IntBroker.Publish(&MSG);
End-If;
```

Populating Message Segments with Documents

The following example shows code for populating message segments with documents.

After you code to populate a segment, set the following statement to *NULL* to release memory:

```
&DOC = null;
```

Then, set the following statement to *TRUE* to manage memory and guarantee that only one segment is in memory at a given time:

```
&DOC = &MSG.GetDocument(true);
```

If you do not following these coding guidelines, memory will get filled with data from all segments in the message.

The previous statements are in emphasis in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Declare Function PopulateDocument PeopleCode QE_FLIGHTDATA.QE_ACNUMBER Field⇒
Formula;


Local Document &DOC;

&MSG = CreateMessage(Operation.FLIGHTPLAN_DOC);

/*pass in true to get ownership of the object*/
/
&DOC = &MSG.GetDocument(true);

/*popualte the docment with data */
PopulateDocument(&DOC, 1);

/*create a new segment */
&MSG.CreateNewSegment();

/* null out object to release memory */
&DOC = null;
&DOC = &MSG.GetDocument(true);
PopulateDocument(&DOC, 2);
&MSG.CreateNewSegment();

&DOC = null;
&DOC = &MSG.GetDocument(true);
PopulateDocument(&DOC, 3);

/* publish segmented Message (3 segments) */
%IntBroker.Publish(&MSG);
```

Retrieving Document Data

This section discusses and provides code examples for how to:

- Retrieve document data from the Message object.
- Retrieve document data from message segments.
- Retrieving document data within a Notification event using message segments

Retrieving Document Data from the Message Object

The following code provides an example of how to retrieve a document from the Message object:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &MSG;
Local Document &DOC;
Local Primitive &PRIM;
Local Compound &COM, &COM_ID, &COM_BILL, &COM_SHIP, &COM_ITEM;
Local Collection &COL_ITEM;

&MSG = CreateMessage(Operation.PURCHASE_ORDER);
&DOC = &MSG.GetDocument();
&COM = &DOC.DocumentElement;

&COM.GetPropertyByName("LanguageCode").Value = "ENG";

/* Populate TransactionID Compound */
&COM_ID = &COM.GetPropertyByName("TransactionId");
&COM2.GetPropertyByIndex(1).value = "KAC";
&COM2.GetPropertyByIndex(1).value = "12345678";

/* Populate BillTo Compound */
&COM_BILL = &COM.GetPropertyByName("BillTo");
&COM5.GetPropertyByName("name").Value = "RobbyNash";
&COM5.GetPropertyByName("number").Value = 713;
&COM5.GetPropertyByName("street").Value = "High Wind";
&COM5.GetPropertyByName("unit").Value = "";
&COM5.GetPropertyByName("city").Value = "Paia";
&COM5.GetPropertyByName("state").Value = "Maui HI";
&COM5.GetPropertyByName("zipcode").Value = "96779";

/* Populate item_collection Collection (2 rows) */
&COL_ITEM = &COM.GetPropertyByName("item_collection");

&COM_ITEM = &COL_ITEM.CreateItem();
&PRIM = &COM_ITEM.GetPropertyByName("item");
&PRIM.Value = "mast";
&PRIM = &COM_ITEM.GetPropertyByName("sku");
&PRIM.Value = "123322";
&PRIM = &COM_ITEM.GetPropertyByName("price");
&PRIM.Value = 300;
&PRIM = &COM_ITEM.GetPropertyByName("quantity");
&PRIM.Value = 12;
&nRet = &COL_ITEM.AppendItem(&COM_ITEM);

&COM_ITEM = &COL_ITEM.CreateItem();
&COM5.GetPropertyByName("item").Value = "boom";
&COM5.GetPropertyByName("sku").Value = "123334";
&COM5.GetPropertyByName("price").Value = 270;
&COM5.GetPropertyByName("quantity").Value = 10;
&nRet = &COL_ITEM.AppendItem(&COM_ITEM);

%IntBroker.Publish(&MSG);
```

Retrieving Document Data from Message Segments

The following code example shows an example of retrieving document data from message segments. As discussed elsewhere in this section, the example shows setting the following statement to *NULL* :

```
&DOC = Null;
```

Setting the statement to *NULL* clears memory between segment loops:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class DOCUMENT_TESTER implements PS_PT:Integration:INotificationHandler
   method FLIGHTDATA();
   method OnNotify(&MSG As Message);
end-class;

/* constructor */
method FLIGHTDATA
end-method;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   /* Variable Declaration */

   Local Rowset &rs;
   Local Document &DOC;
   Local Record &FLIGHTDATA, &REC;
   Local integer &i;

   /* get each segment of data via a Document and proces it */

   For &i = 1 To &MSG.SegmentCount

      /* null out object to release memory */
      &DOC = Null;
      &MSG.GetSegment(&i);

      /* pass true to GetDocument method to take ownership of object */
      &DOC = &MSG.GetDocument( True);

      /* process Document data for each segment */


   End-For;

 end-method;
```

Retrieving Document Data within a Notification Event Using Message Segments

The following code example demonstrates how to retrieve document data within an Notification event using message segments.

As discussed elsewhere in this section, the example shows setting the following statement to *NULL* :

```
&DOC = Null;
```

Setting the statement to *NULL* clears memory between segment loops:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
For &i = 1 To &MSG.SegmentCount
   &DOC = Null;
   &MSG.GetSegment(&i);
   &DOC = &MSG.GetDocument();

   &COM = &DOC.DocumentElement;

   &str = &COM.GetPropertyByName("LanguageCode").Value;

   &COM_ID = &COM.GetPropertyByName("TransactionId");
   &str = &COM_ID.GetPropertyByIndex(2).Value;

   &COM_BILL = &COM.GetPropertyByName("BillTo").Value;
   &str = &COM_BILL.GetPropertyByName("name").Value;

   &COL_ITEM = &COM.GetPropertyByName("item_collecion");

   For &j = 1 To &COL_ITEM.count

      &COM_ITEM = &COL_ITEM.GetItem(&j);
      &str = &COM_ITEM.GetPropertyByName("item").Value;

   End-For;
End-For;
```

Sending and Receiving Binary Data

This section discusses how to:

- Send binary data.
- Receive binary data.

Understanding Sending and Receiving Binary Data

PeopleSoft supports the MTOM protocol for sending and receiving binary data using service operations. While you can send and receive binary data using SOAP, doing so requires that you Base64-encode the data, which can increase message size by 33 percent or more. The MTOM protocol enables you to send and receive binary data in its original binary form, without any increase in size due to encoding.

Whether sending or receiving MTOM-encoded binary data, you use message segments to store the data. The SegmentContentType property of the Message object is used to set or read the content type of each message segment.

Sending MTOM-Encoded Binary Data

This section discusses how to send MTOM-encoded binary data and discusses how to:

- Set target connector properties to send MTOM-encoded binary data.
- Develop messages to send MTOM-encoded binary data.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Setting Target Connector Properties to Send MTOM-Encoded Binary Data

When sending MTOM-encoded binary data, you must use the HTTP target connector. The HTTP target connector features an MTOM property that you must set to *Y* for MTOM encoding to occur. When you set the MTOM property to *Y*, the HTTP target connector attempts to convert all outgoing message to the MTOM wire format.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using Listening Connectors and Target Connectors," Using the HTTP Target Connector.

Developing Messages to Send MTOM-Encoded Binary Data

An outgoing MTOM message is composed of a SOAP message and one or more sets of binary data. Segments are used to add the binary data to the outgoing request.

For each chunk of data, use a single segment as follows:

1. Create the segment.
2. Copy the data to the segment.
3. Add the MIME type that is to appear for the MIME part containing the binary in the outgoing MTOM wire message.
4. Set the content transfer encoding to *binary*.

In PeopleCode this appears as:

```
&theMessage.CreateNextSegment();

  If (&theMessage.SetContentString(&dataString)) Then
     &theMessage.SegmentContentType = "image/jpeg";
     &theMessage.SegmentContentTransfer = %ContentTransfer_Binary;
  End-If;
```

The SetContentString method requires character data, and is not capable of passing binary data. For MTOM, pass in a Base64–encoded string that contains the binary data. The File object method GetBase64StringFromBinary allows a binary file to be read in and captured as a string; use this string to set the data for the newly created segment.

See *PeopleTools 8.52: PeopleCode API Reference*, "Message Classes," SetContentString and *PeopleTools 8.52: PeopleCode API Reference*, "File Class," GetBase64StringFromBinary.

When seen on the wire, the SOAP XML in the MIME multipart message contains xop:Include references. These references point to the MIME parts that contain the binary data. In order to be able to construct these references, Integration Broker requires that a specific XML element, PsftXopInclude, is present in the outgoing message.

Each PsftXopInclude element corresponds directly to an xop:include in the outgoing wire message, and therefore each PsftXopInclude element logically corresponds to an instance of binary data. Placement of the PsftXopInclude element in the XML is application-specific; Integration Broker does not require any particular location.

In the following example the first instance of the PsftXopInclude element corresponds to the logical point to include an image and the second PsftXopInclude element corresponds to the logical point to include a binary document:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version='1.0'?>
<JobApplication>
  <Photo name='JohnSmith'>
    <PsftXopInclude SegmentNumber='1'/>
  </Photo>
  <Resume name='JohnSmithCV'>
    <PsftXopInclude SegmentNumber='2'/>
  </Resume>
</JobApplication>
```

The value of the SegmentNumber attribute is used by Integration Broker to link the PsftXopInclude entry to a specific segment used to add the data. For the purposes of MTOM, the first segment used to add binary data is considered to be number 1, the second segment, number 2, and so on. Care should be taken when setting these values as Integration Broker does not check to ensure that they are correct; they are used as-is to build the xop:include references in the wire message.

In the previous example, the message data is not SOAP-wrapped. You can choose to build your own SOAP wrapper or elect to have Integration Broker SOAP-wrap the message. If Integration Broker is to SOAP wrap the message, you must set the HTTP target connector property SOAPUpContent to *Y*.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using Listening Connectors and Target Connectors," Using the HTTP Target Connector.

An example of XML to use this feature is as follows:

```
<?xml version="1.0"?>
<flt:process xmlns:flt=" http://xmlns.oracle.com/Enterprise/
  Tools/schemas/flightdata.v1">
  <PsftXopInclude/>
  <flt:input1>515</flt:input1>
  <PsftXopInclude/>
  <flt:input2>664</flt:input2>
 </flt:process>
```

You would include the previous XML in the XmlDoc object and add it to the first segment of the message.

The additional segments include the binary data associated with each declaration. For example:

```
&MSG = CreateMessage(Message.FLIGHTDATA);

&MSG.SetXmlDoc(&xmldoc);

&MSG.CreateNextSegment();
&MSG.SetContentString("your encoded image data");
&MSG.SegmentContentType = "image/gif";
&MSG.SegmentContentTransfer = %ContentTransfer_Binary;

&MSG.CreateNextSegment();
&MSG.SetContentString("your encoded video here")
&MSG.SegmentContentType = "video/mp4";
&MSG.SegmentContentTransfer = %ContentTransfer_Binary;

%IntBroker.Publish(&MSG);
```

The following code example provides another XML example that demonstrates using this feature:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local File &theFile;
Local XmlDoc &theXmlDoc;
Local Message &theMessage;
Local string &theBase64encodedString;

/* note: this example does not have any error handling, in */
/* order to keep the code relatively short and concise.    */

/* create the message, and add the basic XML message data */
/* ------------------------------------------------------ */

&theMessage = CreateMessage(Operation.QE_FLIGHTPLAN_UNSTRUCT);
Local string &xml;

/* this example requires the SOAPUPContent HTTP Target */
/* connector property to be set to "Y", so that the    */
/* outbound XML will be SOAP wrapped.                  */

&xml = &xml | "<?xml version='1.0'?>";
&xml = &xml | "<JobApplication>";
&xml = &xml | "<Photo name='JohnSmith'>";
&xml = &xml | "<PsftXopInclude SegmentNumber='1'/>";
&xml = &xml | "</Photo>";
&xml = &xml | "<Resume name='JohnSmithCV'>";
&xml = &xml | "<PsftXopInclude SegmentNumber='2'/>";
&xml = &xml | "</Resume>";
&xml = &xml | "</JobApplication>";


&theXmlDoc = CreateXmlDoc(&xml);
&theMessage.SetXmlDoc(&theXmlDoc);

/* add an image to the outgoing message */
/* ------------------------------------ */

&theFile = GetFile("D:\output\smallPicture.jpg", "R", %FilePath_Absolute);
If &theFile.IsOpen Then
   &theBase64encodedString = &theFile.GetBase64StringFromBinary();
   &theFile.Close();
End-If;

&theMessage.CreateNextSegment();
If (&theMessage.SetContentString(&theBase64encodedString)) Then
   &theMessage.SegmentContentType = "image/jpeg";
   &theMessage.SegmentContentTransfer = %ContentTransfer_Binary;
End-If;


/* add a PDF file to the outgoing message */
/* ------------------------------------ */

&theFile = GetFile("D:\output\smallDocument.pdf", "R", %FilePath_Absolute);
If &theFile.IsOpen Then
   &theBase64encodedString = &theFile.GetBase64StringFromBinary();
   &theFile.Close();
End-If;

&theMessage.CreateNextSegment();
If (&theMessage.SetContentString(&theBase64encodedString)) Then
   &theMessage.SegmentContentType = "application/pdf";
   &theMessage.SegmentContentTransfer = %ContentTransfer_Binary;
End-If;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
/* send the message */
/* --------------- */

%IntBroker.Publish(&theMessage);
```

See Also

Chapter 7, "Sending and Receiving Messages," Working With Message Segments, page 178

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," SetContentString

PeopleTools 8.52: PeopleCode API Reference, "Message Classes," SegmentContentType

Receiving Binary Data

This section discusses receiving MTOM-encoded binary data requests and discusses how to:

- Enable listening connectors to receive MTOM-enoded binary data.
- Use PeopleCode to process inbound MTOM-encoded binary data.

Understanding Receiving MTOM-Encoded Binary Data

To receive MTOM-encoded binary data from integration partners, you must use the PeopleSoft services listening connector or the HTTP listening connector.

When a message is received by an MTOM-enabled connector, the gateway first determines if the message is using MTOM. If it is not, the message is processed normally. If MTOM is detected, the gateway extracts the SOAP message from the MIME and then encodes the binary data in the MIME parts. This is effectively a pre-processing step and is done first, before normal processing can occur. Once complete, the SOAP message is then treated no differently from any other SOAP message received. The binary data is Base64 encoded, and is attached to the message in the form of segments.

Enabling Listening Connectors to Receive MTOM-Encoded Binary Data

The following listening connectors can receive and process MTOM-encoded messages:

- PeopleSoft services listening connector.
- HTTP listening connector.

You must enable these connectors for receiving and processing MTOM messages for any MTOM processing to occur. You enable these properties in the integration gateway properties file. The properties appear the MTOM Enable ListeningConnectors section as shown in the following example:

```
# MTOM Enable ListeningConnectors
#
#ig.MTOM.enablePeopleSoftServiceListeningConnector=true
#ig.MTOM.enableHttpListeningConnector=true
```

By default the properties are not enabled and are commented out.

To enable a given connector, remove the comment (#) and ensure that the property is set to *true*.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. When these properties are enabled there is a slight performance degradation to all non-MTOM requests sent to the connectors. The degradation is a result of system process that takes place to determine if requests are MTOM-encoded messages.

Using PeopleCode to Process Inbound MTOM–Encoded Binary Data

MTOM messages are processed in the form of message segments. The system processes inbound MTOM requests in two general steps:

- Process the XML data contained in the first segment.
- For each subsequent segment, process the binary data.

The first segment contains the XML data. The xop:include references in the XML are replaced with PsftXopInclude elements, and each instance will point to the segment containing the associated binary data.

The structure of this XML is application-specific, and therefore processing of this XML cannot be easily generalized. You may be able to use the location of the PsftXopInclude elements in the XML to derive information about the binary data segments.

Consider the following inbound MTOM request example:

```
<?xml version='1.0'?>
<JobApplication>
  <Photo name='JohnSmith'>
    <PsftXopInclude SegmentNumber='2'/>
  </Photo>
  <Resume name='JohnSmithCV'>
    <PsftXopInclude SegmentNumber='3'/>
  </Resume>
</JobApplication>
```

In this example, the XML has been structured such that the parent element contains a *name* value for the associated binary content. A more complete XML might also contain information such as file type, size, or creation date. Again, the structure of this XML is not necessarily determined by Integration Broker, but rather by the design of the application itself.

All segments after the first contain the Base64–encoded binary data. This data is accessible as a string. Processing of this is also application-specific. Some applications may decide to store the encoded string for later use, while others may wish to decode it immediately.

To process the string immediately, use the PeopleCode File object method WriteBase64StringToBinary to decode the string and to write it out as a byte array to a file. Once the method has completed and the file closed, the file can be accessed as any other file on the file system.

See *PeopleTools 8.52: PeopleCode API Reference*, "File Class," WriteBase64StringToBinary.

The following code example shows how to use PeopleCode to process and inbound MTOM request:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class MTOM_CLASS implements PS_PT:Integration:INotificationHandler
   method MTOM_CLASS();
   method OnNotify(&MSG As Message);
   method getFileExtensionForContentType(&contentType As string)
   Returns string;
end-class;

/* constructor */
method MTOM_CLASS
end-method;

method getFileExtensionForContentType
   /+ &contentType as String +/
   /+ Returns String +/

   Evaluate &contentType
   When = "image/jpeg"
      Return "jpg"
   When = "application/pdf"
      Return "pdf"
   When-Other
      Return "unk"
   End-Evaluate;

end-method;


method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/


   /* note: this example does not have any error handling, in */
   /* order to keep the code relatively short and concise.    */


   /* Variable Declaration */
   Local integer &i, &j;
   Local string &contentSectionData, &contentSectionType;
   Local File &theFile;
   Local XmlDoc &theXml;


   /* the first section will be XML */
   /* ----------------------------- */

   &MSG.GetSegment(1);
   &contentSectionData = &MSG.GetContentString(1);

   &theXml = CreateXmlDoc(&contentSectionData);

   Local array of XmlNode &nodeList;

   /* get all PsftXopInclude nodes, ignore namespaces */
   &nodeList = &theXml.DocumentElement.FindNodes("//*[local-name()=
   'PsftXopInclude']");


   /* all subsequent sections will be binary data */
   /* ------------------------------------------- */

   For &i = 2 To &MSG.SegmentCount
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      &MSG.GetSegment(&i);
      &contentSectionData = &MSG.GetContentString(&i);

      /* get the type information directly from the segment */
      /* we'll use this to determine the file extension */
      &contentSectionType = &MSG.SegmentContentType;

      Local string &theFileName = "D:\output\";

      For &j = 1 To &nodeList.Len
         If (&nodeList [&j].GetAttributeValue("SegmentNumber") =
         String(&i)) Then

             rem we've found the entry that matches this content section;
             rem use the 'name' attribute from the parent XML element to
             rem get the file name
             rem NOTE - this assumes a particular XML format that may not
             rem be the same for most applications;

            &theFileName = &theFileName | &nodeList [&j].ParentNode.
            GetAttributeValue("name");

         End-If;
      End-For;

      rem build the complete filename, including the extension;
      &theFileName = &theFileName | "." | %This.getFileExtensionForContentType
      (&contentSectionType);


      &theFile = GetFile(&theFileName, "W", %FilePath_Absolute);
      If &theFile.IsOpen Then
         &theFile.WriteBase64StringToBinary(&contentSectionData);
         &theFile.Close();
      End-If;

   End-For;

end-method;
```

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Using the integrationGateway.properties File

Chapter 7, "Sending and Receiving Messages," Working With Message Segments, page 178

PeopleTools 8.52: PeopleCode API Reference, "File Class," WriteBase64StringToBinary

Using PeopleCode to Manage REST Service Operations

This section discusses how to:

- Use PeopleCode to manage provider REST service operations.
- Use PeopleCode to manage consumer REST service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Generate fully-qualified URLs for REST resources.

Using PeopleCode to Manage Provider REST Service Operations

This section discusses how to:

- Read document template data and populate response messages.
- Set HTTP response headers.
- Retrieve HTTP response header data.

Reading Document Template Data and Populating Response Messages

To read document template data and populate provider response messages, use the OnRequest method. You implement the OnRequest method using an application class, specifically using the IRequestHandler application interface.

When the OnRequest event is fired the document template is populated with the values based on the corresponding URI template. You can use the populated primitive values along with the URI Template index to determine the proper response message data to send back to the client. The code snippet below shows a simple example of reading the document template data and populating the response message. Note that one can override the HTTP return code that is sent as part of the response to the client.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IRequestHandler;

class WeatherData implements PS_PT:Integration:IRequestHandler
   method WeatherData();
   method OnRequest(&MSG As Message) Returns Message;
   method OnError(&request As Message) Returns string;
end-class;

/* constructor */
method WeatherData
end-method;

method OnRequest
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnRequest +/
   /* Variable Declaration */

   Local Document &Doc;
   Local Compound &COM;
   Local Message &response;
   Local XmlDoc &weather_xmldoc;
   Local XmlNode &info, &country, &state, &city, &day, &data, &flightdata;
   Local Rowset &RS;

   /* get populated Document Template */
   &Doc = &MSG.GetURIDocument();
   &COM = &Doc.DocumentElement;

   &weather_xmldoc = CreateXmlDoc("");

 /* populate xmldoc with data from the Document Template */
   &info = &weather_xmldoc.CreateDocumentElement("WeatherInformation");
   &country = &info.AddElement("Country");
   &country.NodeValue = &COM.GetPropertyByName("country").Value;
   &state = &info.AddElement("State");
   &state.NodeValue = &COM.GetPropertyByName("state").Value;
   &city = &info.AddElement("City");
   &city.NodeValue = &COM.GetPropertyByName("city").Value;
   &day = &info.AddElement("Day");
   &day.NodeValue = &COM.GetPropertyByName("day").Value;

/*  determine HTTP method that was invoked to determine proper response
/*  message */
   If &MSG.HTTPMethod = %IntBroker_HTTP_GET Then

      &data = &info.AddElement("Forecast");
      &data.NodeValue = "55 degrees and raining";
      &response = CreateMessage(Operation.WEATHERSTATION_GET,
        %IntBroker_Response);
      &response.SetXmlDoc(&weather_xmldoc);

   End-If;

   If &MSG.HTTPMethod = %IntBroker_HTTP_DELETE Then

      &data = &info.AddElement("Forecast");
      &data.NodeValue = "deleted";
      &response = CreateMessage(Operation.WEATHERSTATION_DELETE,
        %IntBroker_Response);
      &response.SetXmlDoc(&weather_xmldoc);

   End-If;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
   Return &response;
end-method;

method OnError
   /+ &request as Message +/
   /+ Returns String +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnError +/
   Local Message &Fault_Msg;
   Local Document &fault_doc;
   Local Compound &COM;

   If &request.HTTPMethod = %IntBroker_HTTP_GET Then

      &Fault_Msg = CreateMessage(Operation.WEATHERSTATION_GET,
        %IntBroker_Fault);
      &fault_doc = &Fault_Msg.GetDocument();

      &COM = &fault_doc.DocumentElement;
      &COM.GetPropertyByName("fault_data").Value = &request.IBException.
        ToString();

      Return &fault_doc.GenXmlString();
   End-If;

   Return "";

end-method;
```

Setting HTTP Reponse Headers

Use the LoadRESTHeaders method of the IBInfo class to load the response headers defined on the routing for a REST-based service operation. Once loaded, the headers can be modified without specifying the connector override property.

The code snippet below shows how to add HTTP response headers to the any REST based service operation response within the OnRequest event.

```
&response = CreateMessage(Operation.WEATHERSTATION_GET, %IntBroker_Response);

&bRet =   &response.IBInfo.LoadRESTHeaders();

/* any/ modify additional Headers not defined on Routing */
&bRet = &response.IBInfo.IBConnectorInfo.AddConnectorProperties
    ("Content-Language ", "eng ", %HttpHeader);

Return &response;
```

Retrieving Response HTTP Header Data

You can use the REST method type Head to retrieve meta-information written in response HTTP headers, without having to transport the entire content.

The REST-based service operation created with a method of HEAD does not have a request or response message.

You can assign the OnRequest handler used for the GET method to the service operation to check if the method type is HEAD and, if so, simply send back the HTTP response headers.

The following code snippet shows how to use the OnRequest method to retrieve HTTP response headers

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. HTTP response headers can be sent back to the client for all REST method types.

```
If &MSG.HTTPMethod = %IntBroker_HTTP_HEAD Then

&response = CreateMessage(Operation.WEATHERSTATION_HEAD,
  %IntBroker_Response);

&bRet =   &response.IBInfo.LoadRESTHeaders();

/* any additional Headers not defined on Routing */
&bRet = &response.IBInfo.IBConnectorInfo.AddConnectorProperties
 ("Content-Language ","eng ", %HttpHeader);

&bRet = &response.IBInfo.IBConnectorInfo.AddConnectorProperties
 ("WWW-Authenticate", "Basic",  %HttpHeader);

Return &response;
End-If;
```

Using PeopleCode for Consumer REST Service Operations

This section discusses how to:

- Invoke a consumer REST service operation.
- Add REST HTTP connector headers.

Invoking a Consumer REST Service Operations

To invoke a consumer REST service operation, the message is instantiated and then the document template is retrieved and populated. The URI index is selected and a SyncRequest method is invoked. The response message contains the HTTP return code. Processing of the data is the same as any other SyncRequest method. In the case of an error, if the User Exception option is selected on the routing, you can attempt to read the fault if defined on the service operation. If the HTTP response code is the same as that defined on the fault message, then the fault message is created and returned. You can read the fault in the document, if the fault message is a Document message type. The message property IsFault is read to determine if a fault message was created.

The code example shows a simple example of populating the document template data and entering the URI resource index to use and invoking the SyncRequest method.

The GetURIDocument method of the Message class is used to retrieve the URI for the REST based on the specified index. The URIResourceIndex property of the Message class is used to set or return the index for the URI as an integer. This index corresponds to the row number in the URI grid of the REST Resource Definition section of the service operation definition.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Declare Function out_BI_results PeopleCode QE_FLIGHTDATA.QE_ACNUMBER
  FieldFormula;

&MSG = CreateMessage(Operation.MAPS_GET);

/* Get URI Document and populate with data */
&DOC = &MSG.GetURIDocument();

&COM = &DOC.DocumentElement;

&COM.GetPropertyByName("MapType").Value = "topographic";
&COM.GetPropertyByName("Scale").Value = "1:63";
&COM.GetPropertyByName("Planet").Value = "Earth";
&COM.GetPropertyByName("Country").Value = "USA";
&COM.GetPropertyByName("City").Value = "WhiteSalmon";
&COM.GetPropertyByName("Name").Value = "MainSteet";

/* Set URI Resource Index to be used */
&MSG.URIResourceIndex = 1;
&return_message = %IntBroker.SyncRequest(&MSG);

/* Get return data and display */

If &return_message.ResponseStatus = %IB_Status_Success Then

   &xmldocReturn = &return_mesage.GetXmlDoc();
   out_BI_results(&xmldocReturn.GenXmlString());

/* Read Response Headers if set */
   For &i = 1 To &MSG.IBInfo.IBConnectorInfo.GetNumberOfConnectorProperties()

     If &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesType(&i) =
       %HttpHeader

     Then

       &name = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesName(&i);
       &value = &MSG.IBInfo.IBConnectorInfo.GetConnectorPropertiesValue(&i);
     End-If;

   End-For;

Else

   If &return_message.IsFault = True Then

      &Fault_Doc = &return_message.GetDocument();
      &COM = &Fault_Doc.DocumentElement;
      out_BI_results(&COM.GetPropertyByName("fault_data").Value);

   Else
      out_BI_results(&return_message.IBException.ToString());
   End-If;

End-If;
```

Adding REST HTTP Connector Headers

Use the LoadRESTHeader method of the Message class to add HTTP header properties not defined on the routing for the service operation.

The code snippet below shows how to modify HTTP headers using PeopleCode

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. The connector override flag in PeopleCode does not need to be set in this case.

No HTTP properties are currently applicable for REST and will be removed by the Integration Broker framework.

```
&request = CreateMessage(Operation.MAPS_GET);

&bRet = &request.LoadRESTHeaders();

/* add any additional Headers not defined on Routing */
&bRet = &request.IBInfo.IBConnectorInfo.AddConnectorProperties
  ("Content-Language ", "eng ", %HttpHeader);
```

Generating Fully-Qualified URLs for REST Resources

In most REST-based services, representations are hypermedia documents that contain not just data, but links to other resources.

Use the GetUrl method contained in the %IntBroker class to generate fully-qualified URLs for REST service operation resources. You can use the URLs with defined HTML definitions to dynamically add REST-based web service URL links.

Note. A provider or a consumer REST based Service Operation representations can be used to generate the fully-qualified link(s).

The syntax of the GetUrl method is:

```
string &str = %IntBroker.GetURL( string <Service Operation>, integer
<Resource Index of Service Operation>,  document <Document object
defined for document Template> , <optional> bool <secure/ unsecure
REST tgt location>, <optional> bool <add encoding for unsafe characters >
```

The following example shows within an implementation (OnRequest event) of a REST-based provider service, HTML is generated using links defined from other REST-based service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
method OnRequest

 Local Document &Doc_Tmpl, &DOC;
 Local Compound &COM_Tmpl, &COM;
 Local Message &response;
 Local string &STR, &STR1, &STR2, &STR3, &STR4, &strHTML;
 Local boolean &bRet;

 &response = CreateMessage(Operation.WEATHERSTATION_GET, %IntBroker_Response);

 /* read URI Document to get parms out from the request*/
 &Doc_Tmpl = &MSG.GetURIDocument();
 &COM_Tmpl = &Doc_Tmpl.DocumentElement;

 /* Instantiate a Document object based on the REST based Service */
 /* Operations Document Template for which to create a link.      */

 &DOC = CreateDocument("Weather", "WeatherTemplate", "v1");
 &COM = &DOC.DocumentElement;

 /* based off the data from the request populate the Document object */

   If &COM_Tmpl.GetPropertyByName("state").Value = "Washington" Then

      &COM.GetPropertyByName("state").Value = "Washington";

      /* call new method to create fully qualified URL(s) */

      &COM.GetPropertyByName("city").Value = "WhiteSalmon";
      &STR = %IntBroker.GetURL("WEATHERSTATION_GET", 2, &DOC);

      &COM.GetPropertyByName("city").Value = "Troutlake";
      &STR1 = %IntBroker.GetURL("WEATHERSTATION_GET", 2, &DOC);

      &COM.GetPropertyByName("city").Value = "Yakima";
      &STR2 = %IntBroker.GetURL("WEATHERSTATION_GET", 2, &DOC);

      &COM.GetPropertyByName("city").Value = "Lyle";
      &STR3 = %IntBroker.GetURL("WEATHERSTATION_GET", 2, &DOC);

      /* use these URLs as bind variables for the HTML definition */
      &strHTML = GetHTMLText(HTML.WEATHER_CITIES, &STR, &STR1, &STR2, &STR3);

      /* set the data in the response message */
      &bRet = &response.SetContentString(&strHTML);

   End-If;

Return &response;
end-method;
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 8

Building Message Schemas

This chapter provides an overview of the message Schema Builder and describes how to:

- Select and view data in the message Schema Builder.
- Build message schemas for rowset-based messages.
- Import message schemas for nonrowset-based messages.
- Modify message schemas.
- Delete message schemas.

Understanding the Message Schema Builder

The message Schema Builder enables you to build, import, modify and delete XML message schemas.

Note. The terms *message schema*,*XML message schema*, and *schema* are used interchangeably in this chapter.

To test message schemas during development, use the Schema Tester utility.

Use the Service Operations - General page to enable runtime validation for a service operation, or use the Service Schema Validation page to enable validation for several service operations at a time.

See Also

PeopleTools 8.52: Integration Broker Testing Utilities and Tools, "Using the Schema Tester Utility"

Chapter 14, "Enabling Runtime Message Schema Validation," page 329

Message Schemas

An *XML message schema* describes a model for the arrangement of tags and text in a valid XML document. A schema provides a common vocabulary for a particular application that exchanges documents.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Building, Importing, Modifying and Deleting Message Schemas

You can use the Message Schema Builder to manage message schemas for rowset-based messages in the application database.

Note. You can also use the pages of the Message Builder component to manage rowset-based and nonrowset-based schemas. However, the Message Builder enables you to work with only one message schema at a time, whereas , the Message Schema Builder enables you to perform actions, such as building and deleting message schemas, on multiple messages at a time.

Note. You cannot use the Message Schema Builder to build schemas for message parts or container messages. You must use the Message Builder component to build schemas for these message types.

Rowset-Based Message Schemas

Use the Message Schema Builder to generate, regenerate, view or delete rowset-based message schemas.

You cannot regenerate or delete a rowset-based message schema that is a message part. Part and container schemas are automatically generated at save time so there's no need to explicitly regenerate or delete them.

Nonrowset-Based Message Schemas

Use the Message Schema Builder to import new nonrowset-based schemas into the database, modify existing nonrowset-based message schemas, or delete them.

Schemas for nonrowset-based message parts can be deleted or modified, but message parts should never be without a schema. After deleting a nonrowset-based message part, you should always import or enter a new schema for the message.

Selecting and Viewing Data in the Message Schema Builder

This section discusses how to:

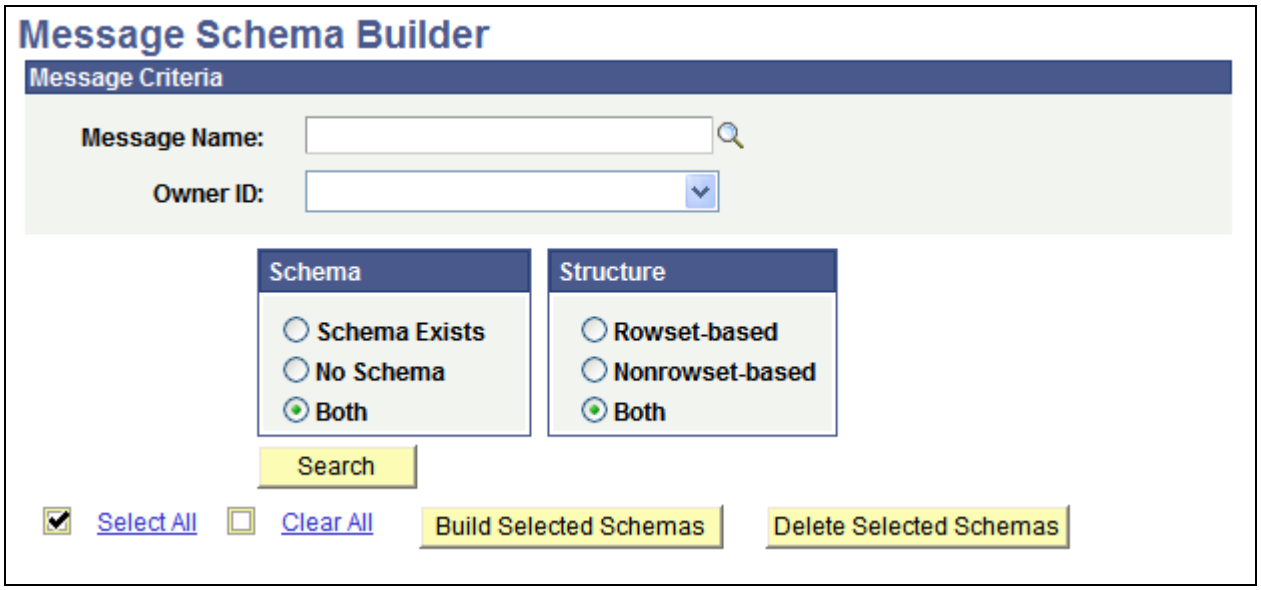
- Select data in the Message Schema Builder.
- View message schema data details.
- View XML message schema code.

Selecting Data in the Message Schema Builder

When you access the Message Schema Builder component (IB_SCHEMABUILD) the Schema Builder page (IB_SCHEMABUILD) displays a search engine to use to search for messages and message schema data with which to work and view.

To access the Schema Builder page, select PeopleTools, Integration Broker, Service Utilities, Message Schema Builder.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Schema Builder page

The Schema Builder page provides the following options for searching for data with which to work and view in the application database.

Message Name

(Optional.) Click the Lookup button to locate a message definition with which to work.

If you do not select a message name, the search will be based on all message definitions in the application database.

Owner ID

(Optional.) From the Owner ID drop-down list, select the owner ID for the message definition.

The owner ID helps to determine the application team that last made a change to a message definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.

Schema

Select from the following options in the Schema group box:

- Schema Exists. Select this option to search message versions for which schemas have been built.
- No Schema. Select this option to search message versions for which no schemas have been built.
- Both. (Default.) Select this option to search all message versions.

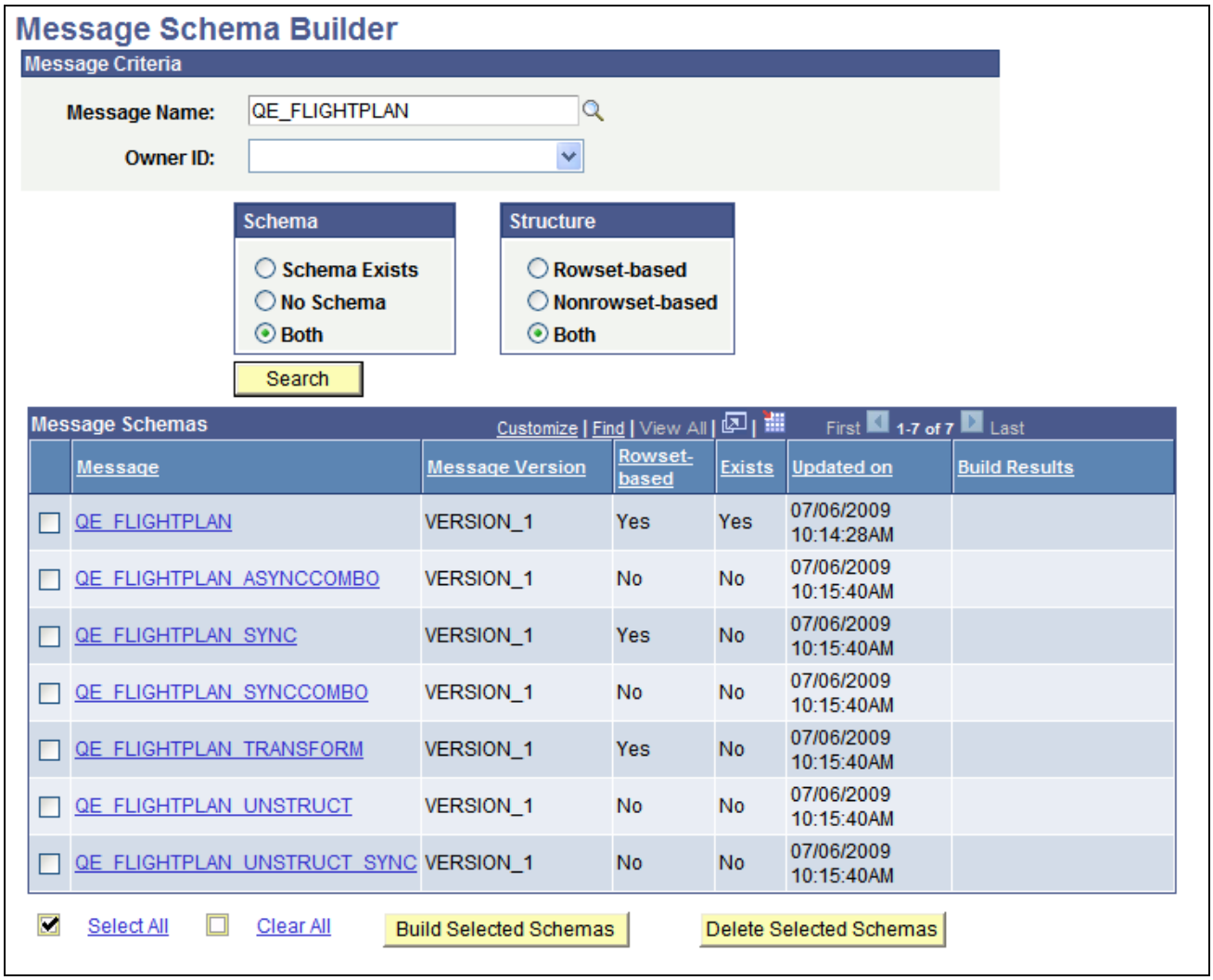
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Structure	Select from the following options in the Structure group box: • Rowset-based. Select this option to search for rowset-based message versions. • Nonrowset-based. Select this option to search for nonrowset-based message versions. • Both. (Default.) Select this option to search for rowset-based and nonrowset-based message versions.
Search	Click the button to search the database based on the criteria selected.

Viewing Message Schema Details

When you search for data in the Schema Builder, message detail results appear in the Message Schemas grid.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Message schemas grid

Message Message name returned from the search of the application database.

Message Version Version of the message returned from the search of the application database.

Rowset-based Indicates the structure of the message. The valid values are:

- *Yes.* Indicates that the message is a rowset-based message.
- *No.* Indicates that the message is a nonrowset-based message.

Exists Indicates whether a schema has been built for the message. The valid values are:

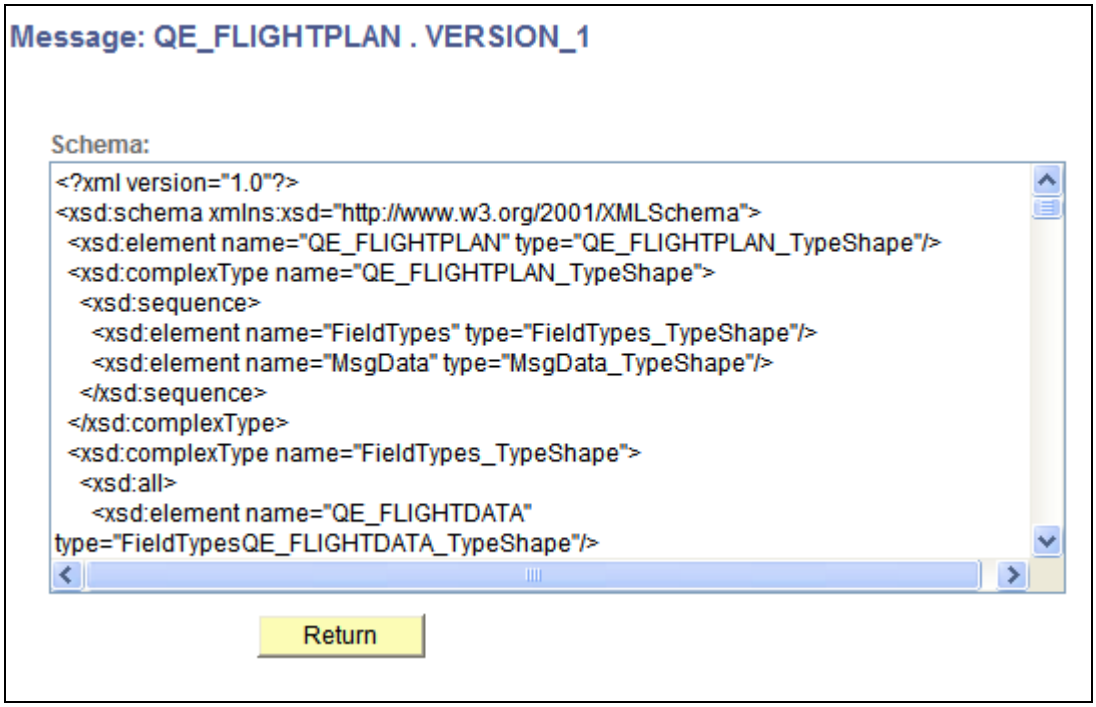
- *Yes.* A schema has been built for the message.
- *No.* A schema has not been built for the message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Updated On	Timestamp of the last update of the record. A new timestamp displays when a schema is generated or deleted for a message.
Build Results	Displays the results of actions performed on a schema.
Build Selected Schemas	Click the button to build schemas for the selected messages.
Delete Selected Schemas	Click the button to delete schemas that exist for the selected messages.

Viewing XML Message Schema

If a message schema exists for a message, click the message name in the Message Schema grid to view the schema details in the Schema Viewer page (IB_SCHEMABUILD_SEC).



Schema details for version 1 of the QE_FLIGHTPLAN message definition

Note. For easier viewing, highlight the data with your cursor.

Message schemas for rowset-based messages are read-only. You can edit message schemas for nonrowset-based messages.

Building Message Schemas for Rowset-Based Messages

This section discusses how to build message schemas for rowset-based messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Building a Message Schema for a Rowset-Based Message

To build a message schema for a rowset-based message:

1. Access the Schema Builder page (PeopleTools, Integration Broker, Service Utilities, Message Schema Builder).
2. Search the application database for the message or messages for which to build schemas.

   See Chapter 8, "Building Message Schemas," Selecting Data in the Message Schema Builder, page 214.
3. Check the box next to the message or messages for which to build schemas.
4. Click the Build Selected Schemas button.

When the schema is built successfully, a timestamp appears in the Updated On field and the Build Results field displays *Successful Schema Insert.*

Importing Message Schemas for Nonrowset-Based Messages

This section discusses how to import message schemas for nonrowset-based messages.

Importing a Message Schema for a Nonrowset-Based Message

To import schemas for nonrowset-based messages:

1. Access the Schema Builder page (PeopleTools, Integration Broker, Service Utilities, Message Schema Builder).
2. Use the Message Schema Builder search engine to locate the message for which you want to import a schema.

   See Chapter 8, "Building Message Schemas," Selecting Data in the Message Schema Builder, page 214.
3. In the Message Schema grid, click the message name link for the message for which you want to import a schema.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Import the schema.

   - Import a schema from a file.

     You can import a schema from a file by using the Upload Schema from File button and selecting the file to import. After you import the file, the contents displays in the Schema text box.

     Note. If you receive the error, "Error retrieving the file from database," verify that one of the variables PS_FILEDIR or PS_SERVDIR is defined in the system variables on your machine.

     See Appendix D, "Setting PS_FILEDIR and PS_SERVDIR Environment Variables," page 655.

   - Direct data entry.

     You can also enter the schema directly in the Schema text box.

5. Click the Save button.

   The Schema Builder page appears.

A timestamp appears in the Updated On field and the Build Results field displays *Successful Schema Insert.*

Modifying Message Schemas

This section discusses how to modify message schemas.

Note. You can modify the content of message schemas built for nonrowset-based messages only.

To modify a schema, you can edit it directly in the Message Schema Builder, or you can export to make changes.

Modifying a Message Schema

To modify a message schema:

1. Select PeopleTools, Integration Broker, Service Utilities, Message Schema Builder.

   The Schema Builder search page appears.

2. Locate the message with which you want to work.

   See Chapter 8, "Building Message Schemas," Selecting Data in the Message Schema Builder, page 214.

3. In the Message Schema grid, click the message name link.

   A new page displays with the message schema populated in a text box.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Modify the schema as needed.

    - Modify the schema directly in the text box, or
    - Modify the schema in the editor of your choice.

      Use your cursor to highlight the contents of the text box and use the keyboard command CTRL + C to copy the contents of the text box. Paste the contents into your editor using the keyboard command CTRL + V. Modify the content as needed. Import the content back into the Message Schema Builder using the instructions described previously in this chapter for importing message schemas for nonrowset-based messages.

      See Chapter 8, "Building Message Schemas," Importing Message Schemas for Nonrowset-Based Messages, page 219.

5. Click the Save button.

The Schema Builder page displays and the Updated On field displays the date and time of the modification, and the Build Results field displays the results of the new schema build.

Deleting Message Schemas

This section discusses how to delete message schemas.

Understanding Deleting Message Schemas

You can delete message schemas using the Message Schema Builder page in the Message Schema Builder component (IB_SCHEMABUILD) or using the Message Schemas page in the Service Administration component (IB_HOME_PAGE).

Note. The Message Schema Builder page provides more comprehensive capabilities for searching for message schema.

You cannot delete a message schema when the message on which the schema is based is:

- Referenced in a service operation.
- Referenced as a message part in a container message.
- A rowset-based message part.
- A container message.
- Referenced in a provided WSDL document.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Using the Message Schema Builder Page to Delete Message Schemas

When deleting a schema using the Message Schema Builder page use only the Delete Selected Schemas button. Do not attempt to delete message schemas by deleting content in the Schema text box in the schema details view; if you save the changes, PeopleSoft Integration Broker will attempt to validate the blank schema at runtime and the validation will fail.

You cannot delete message schemas when the service system status is set to *Production.*

The service system status that is set on the Service Configuration page.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration,* "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services.

To delete a message schema:

1. Access the Message Schema Builder page (PeopleTools, Integration Broker, Service Utilities, Message Schema Builder).

   The Schema Builder search page appears.

2. Locate the message with which you want to work.

   See Chapter 8, "Building Message Schemas," Selecting Data in the Message Schema Builder, page 214.

   The Schema Builder page appears.

3. In the Message Schema section, check the boxes next to the message names that contain schemas you want to delete.

4. Click the Delete Selected Schemas button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 9

Managing Services

This chapter provides an overview of managing services and discusses how to:

- Access and view service definitions.
- Add service definitions.
- Add service operations to service definitions.
- Create and manage integration groups.
- Restrict write access to service definitions.
- Rename and delete service definitions.
- Activate and deactivate services in bulk.

Note. Before you can provide or consume services in a PeopleSoft system, you must configure the system for handling services.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

Understanding Managing Services

Services are used to logically group a set of service operations.

For example, if you have a number of service operations that are related to customers, such as those pertaining to customer information, adding customers, updating customers, deleting customers, and so on, you can create a customer web service and then associate the related service operations with that service.

Warning! PeopleSoft delivers two services with PeopleSoft Integration Broker: IB_GENERIC and IB_UTILITY. These services are used internally by the system. Do not delete or modify these services.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Common Elements Used in This Chapter

Comments (Optional.) Enter comments about the service or service definition.

Description Description of the service.

Generate SOAP Template Click to open the Generate SOAP Template utility. The utility enables you to generate SOAP documents for each service operation in a service for testing purposes.

The link appears when working with a SOAP-based service and only if WSDL has been generated for the service. Note that you can use the Provide Web Service link on the Services page to generate WSDL for the service.

Link Existing Operations Click to add service operations already defined in the system to a service.

Object Owner ID (Optional.) Indicates the owner of the service.

The owner ID helps to determine the application team that last made a change to a service definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.

Operation Type Specifies how the service is transmitted.

On the Service page this field defines the operation type of the service operation added.

Provide Web Service Click to launch the Provide Web Services component and export PeopleSoft services as WSDL documents.

Schema Namespace Provides qualification for attributes and elements within an XML schema document (XSD).

The default is *http://xmlns.oracle.com/Enterprise/Tools/schemas.*

The namespace on the message definition defaults to the schema namespace you set as the default on the Service Configuration page.

Note. If you change the namespace, all future messages will have the new namespace.

Service The name of the service.

Service Alias (Optional.) Overrides the service name and will be the name of the service when the WSDL is provided or exported. The alias enables you to use mixed case in the name.

Service Operation The name of the service operation to associate with the service.

On the Services page, use this field to add new service operations for the current service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Namespace and **Namespace**	The namespace field on the Service pages provides qualification for attributes and elements within a WSDL document. The value defined in the Service Namespace field in the Service Configuration page is used as the default service namespace on the Services page. The default value is *http://xmln.oracle.com/enterprise/tools/service.*
Service System Status	The status that is selected restricts rename, delete, and other administrative actions that users can perform on integration metadata in the Services Administration component. Values are: • *Production.* • *Development.* See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services.
Target Location	Specifies the URL to be used for service requests. You must define this location before creating services and service schemas.
View WSDL	Click to view WSDL documents in the WSDL repository that have been generated for the service. This link appears only when the service you are viewing is a SOAP-based service.

Accessing and Viewing Service Definitions

This section discusses how to.

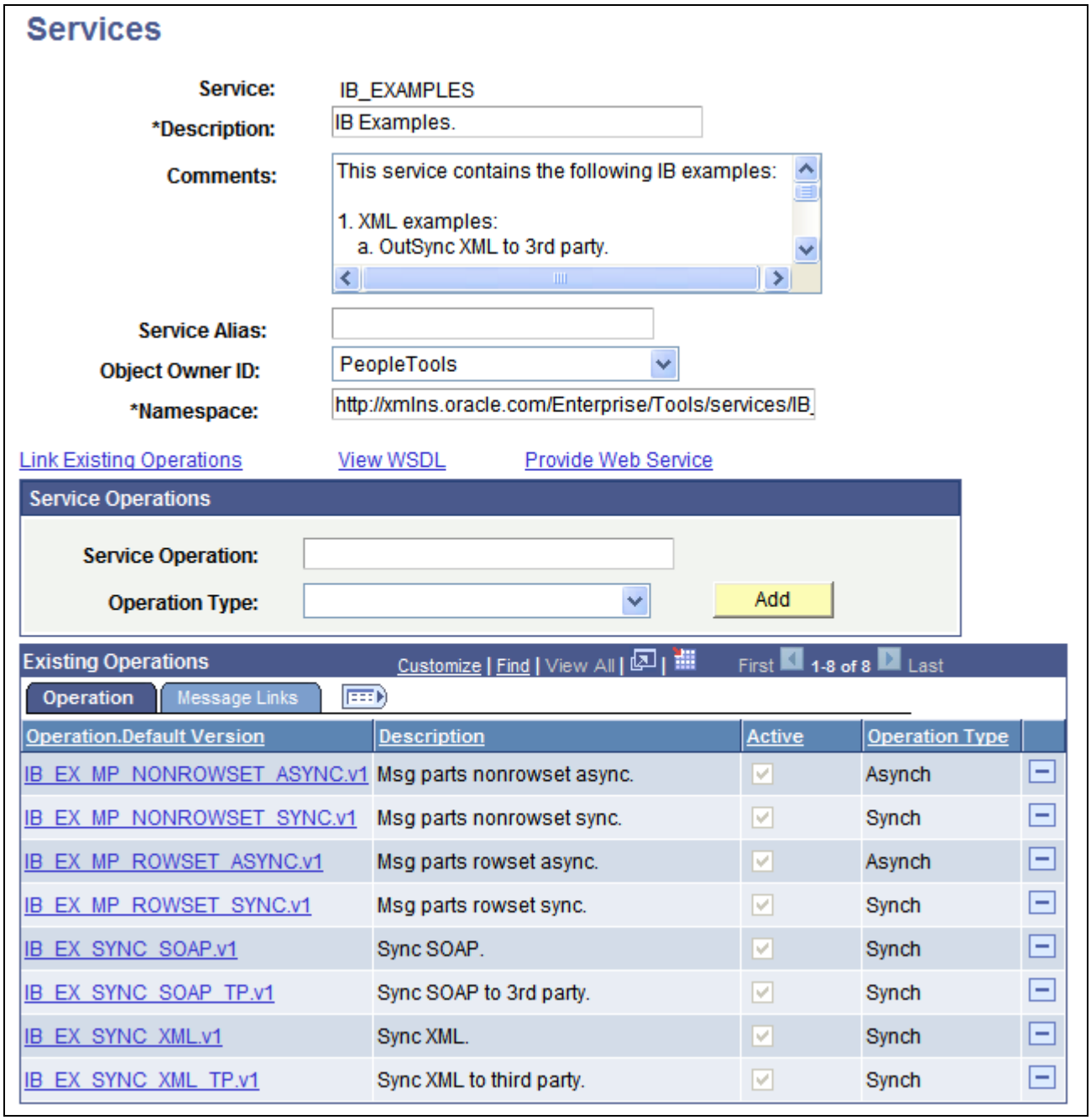
- Access service definitions.
- View WSDL documents generated for services.
- View service operation information.
- View messages defined for services.

Accessing Service Definitions

Service definitions appear on the Services page (IB_SERVICEDEFN) in the Service component (IB_SERVICEDEFN).

To access the Services page, select PeopleTools, Integration Broker, Integration Setup, Services. The following example shows the Services page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Services definition for the IB_EXAMPLES service

The top of the Service page displays general information about the service, including the name of the service, its description, its alias name, and so on.

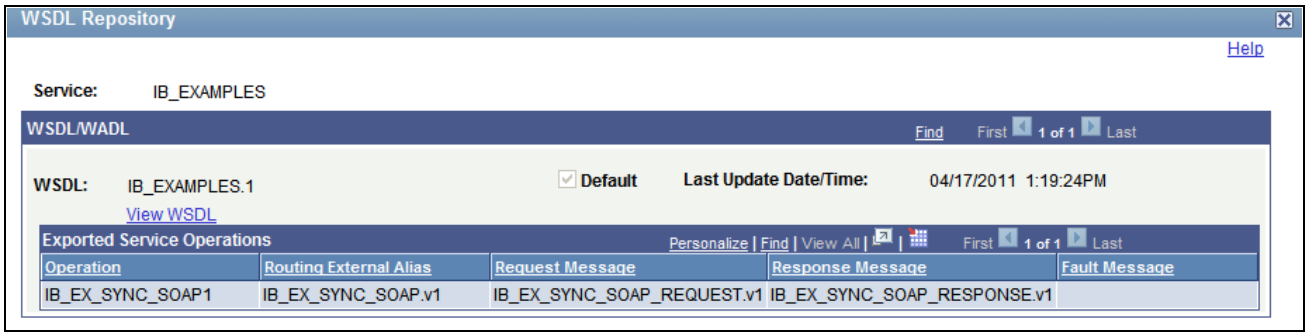
Viewing WSDL Documents Generated for Services

Click the View WSDL link to display the WSDL Repository page (IB_SERVICEDEFN_SEC). This page provides a summary of all the WSDL documents that are generated for the service, as well as the service operations, request messages, response messages, and fault messages that are contained in each.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Service operations must exist for a service to view WSDL documents for the service.

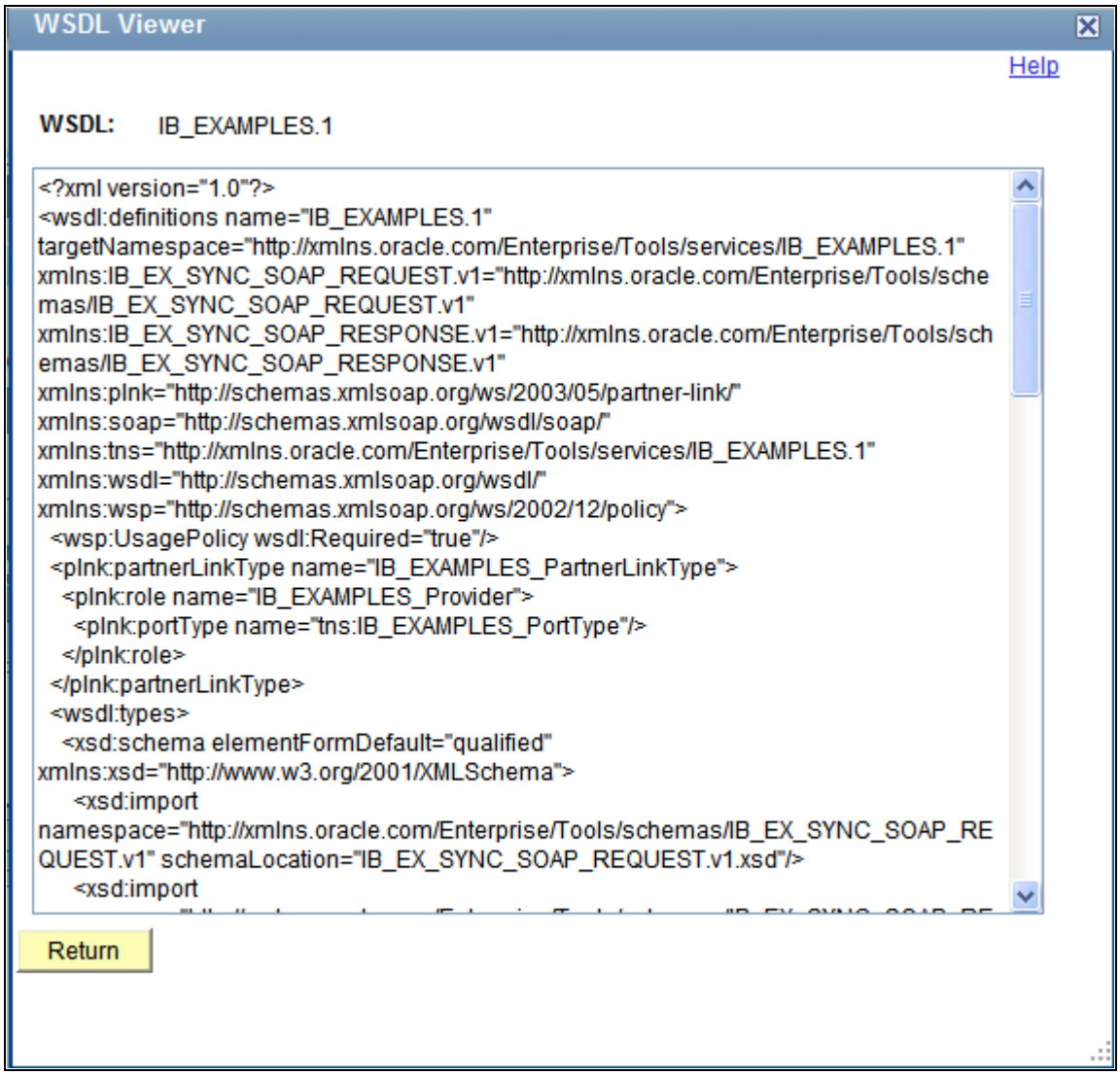
The following example shows the WSDL Repository page for the SOAP-based service IB_EXAMPLES.



The WSDL Repository page shows that WSDL has been exported for one service operation in the IB_EXAMPLES service.

Click the View WSDL link on the WSDL Repository page to view the WSDL document in the WSDL Viewer page (IB_SERVICEWSDL_SEC). The following example shows the WSDL document for the IB_EXAMPLES page in the WSDL Viewer page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Viewing a WSDL document in the WSDL Viewer page.

To view a WSDL document generated for a service:

1. Access the service definition. (PeopleTools, Integration Broker, Integration Setup, Services.)
2. Click the View WSDL link.

   The WSDL Repository page appears.
3. Click the View WSDL link.

   The document appears in the WSDL Viewer page.
4. Click the Return button to return to the WSDL Repository page.
5. Click the Return button again to return to the Services page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Service Operation Information

The Existing Service Operations section of the Services page contains an Operation tab that displays service operations and service operation versions that are associated with the service. It also displays descriptions of the service operations, the type of operation, and whether the service operation is active.

When you click the name of a service operation, the operation opens on the Service Operations page, where you can view and modify service operations information, work with the service operation handlers, routing definitions, and do much more.

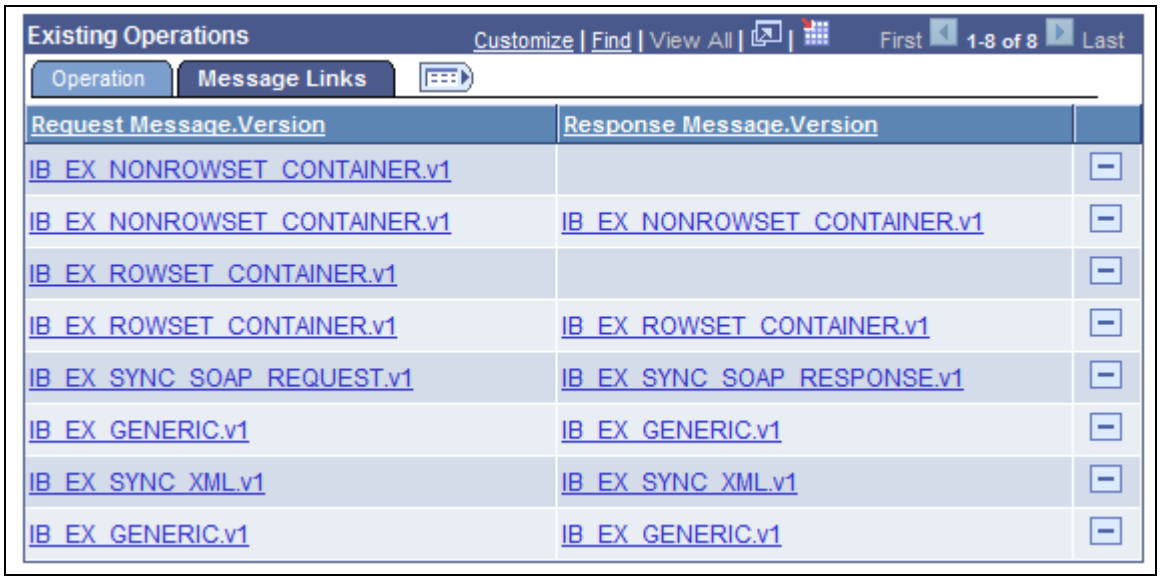
See Also

Chapter 10, "Managing Service Operations," page 247

Viewing Messages Defined for Service Operations

The Existing Service Operations section of the Services page contains a Messages Links tab that displays the request and response messages defined for each service operation.

The following example shows the Message Links tab displaying request and response messages for the service operation that is associated with the IB_EXAMPLES service:



The request and response message for the IB_EXAMPLES service

Click the request or response message name to open the message in the Message Definitions page, where you can view and modify message definition information, message schema information, and more.

See Also

Chapter 6, "Managing Messages," page 75

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Adding Service Definitions

This section discusses how to add service definitions to the PeopleSoft system.

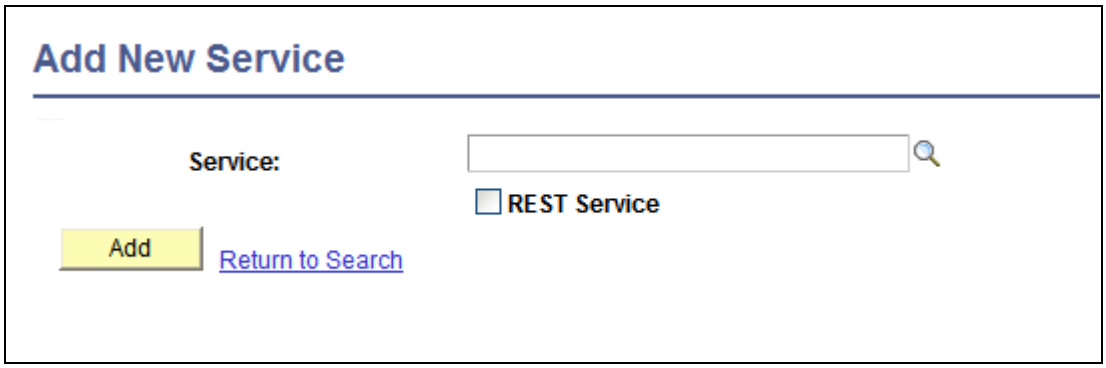
Understanding Naming Services and Service Aliases

Service names and service alias names can be exposed as web services. As a result, the following restrictions apply when you name a service or a service alias. Service names and service alias names:

- Cannot begin with *XML, xml, or Xml.*
- Cannot begin with a digit.
- Cannot begin with a character.

Adding Service Definitions

To add a service definition to the system, use the Add a New Value link on the Services search page. To access this page, select PeopleTools, Integration Broker, Integration Setup, Services. Then select the Add a New Value link.



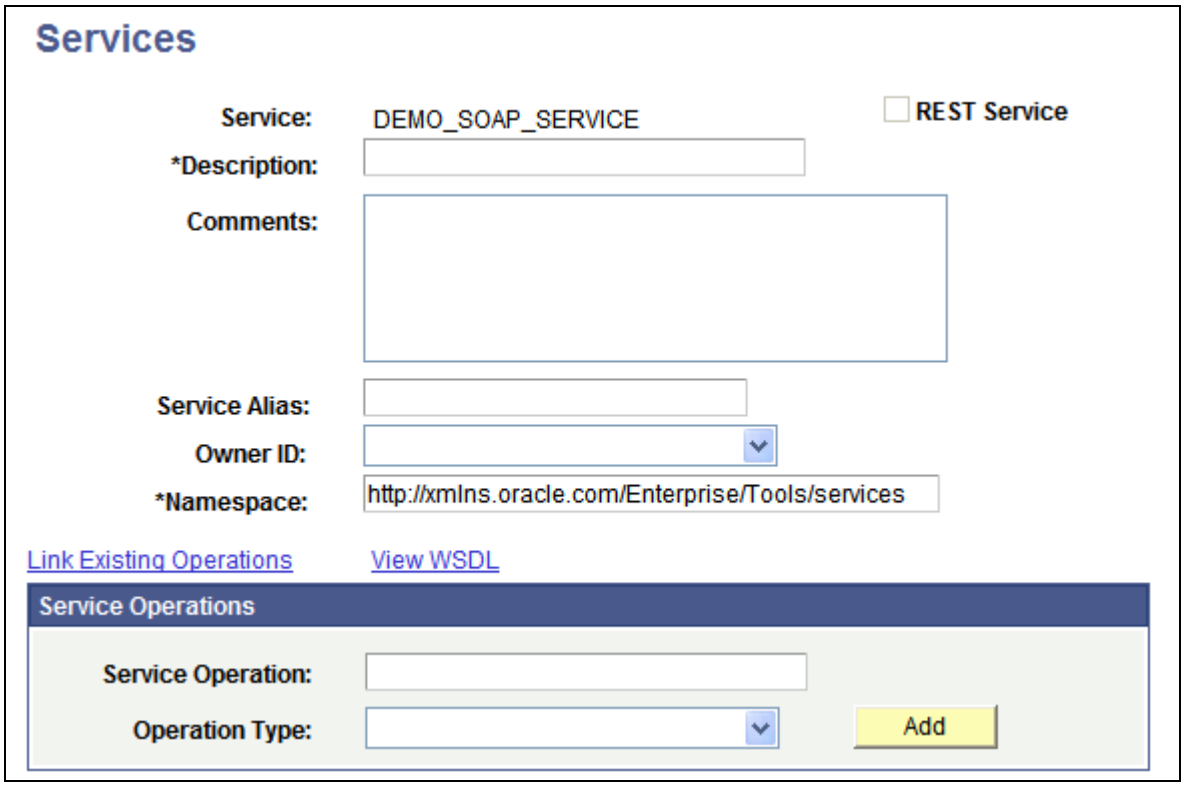
Add a New Service page.

Note. Before you can add a service, you must configure PeopleSoft Integration Broker to handle services using the Service Configuration page.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services.

After you add a service definition to the system, the Services page (IB_SERVICEDEFN) appears where provide additional service definition information. The following example shows the Services page for a non-REST service called DEMO_SOAP_SERVICE:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The Service page after a non-REST service is added to the system.

To add a service definition:

1. Access the Services page (PeopleTools, Integration Broker, Integration Setup, Services).
2. Click the Add a New Value link.
3. In the Service field, enter a name for the service.

   Restrictions for naming services are discussed elsewhere in this section.

   See Chapter 9, "Managing Services," Understanding Naming Services and Service Aliases, page 230.
4. Click the Add button.

   The Services page appears.
5. In the Description field, enter a description for the service.
6. (Optional.) If the service is REST-based, check the Is Provider box to define the service as a provider service.

   By default, when you add a REST-based service to the system it is a provider service.
7. (Optional.) In the Comments field, enter comments about the service or the service.
8. (Optional.) In the Service Alias field, enter an alias name for the service.

   Restrictions for naming service aliases are discussed elsewhere in this section.

   See Chapter 9, "Managing Services," Understanding Naming Services and Service Aliases, page 230.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

9. (Optional.) From the Object Owner ID drop-down list box, select the owner of the service.
10. In the Namespace field, enter a namespace URI for the service.

    The default value is the namespace that is declared in the Service Namespace field on the Service Configuration page.

11. Click the Save button.

See Also

Chapter 9, "Managing Services," Adding Service Operations to Service Definitions, page 232

Adding Service Operations to Service Definitions

This section discusses how to:

- Add existing service operations to service definitions.
- Define new service operations for SOAP-based service definitions.
- Define new service operations for REST-based service definitions.

Understanding Adding Service Operations to Service Definitions

You can add a service operation that already exists in the database to a service definition, or you can add and configure a new service operation and then add it to a service.

Adding Existing Service Operations to Service Definitions

To add an existing service operation to a service:

1. Access the Add Existing Operations page (PeopleTools, Integration Broker, Integration Setup, Services. Click the Link Existing Operations link).

   The Add Existing Operations page appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Select a service operation to add to the service.

   a. In the Service Operation field, enter the name of the service operation to add to the service and click the Search button. The service operation appears in the Available Service Operations grid.

      Check the box next to the service operation name.

   b. In the Service Operation field, click the Lookup button. The Look Up Service Operation page appears.

      Select a service operation from the list to add to the service.

   c. Click the Search button to show a list of service operations in the database.

      Select one or more service operations to add to the service.

3. On the Add Existing Operations page, click the Save button.

   The Services page appears and the service operations that you added appear in the Existing Operations grid.

4. On the Services page, click the Save button.

Defining New Service Operations for SOAP-Based Service Definitions

This section describes how to define new service operations for SOAP-based services.

To add and configure a new service operation for a SOAP-based service:

1. Access the Services page (PeopleTools, Integration Broker, Integration Setup, Services).

   The Services page appears.

2. In the Service Operations section, locate the Service Operation field and enter a name for the new service operation.

3. From the Operation Type drop-down list, select the type of service operation you are adding.

   The valid values are:

   - *Asynch Request/Response.*
   - *Asynch to Synch.*
   - *Asynch–One Way.*
   - *Synchronous.*

   Service operation types are discussed elsewhere in this PeopleBook.

   See Chapter 10, "Managing Service Operations," Services Operation Types, page 248.

4. Click the Add button.

   The Service Operations-Definitions page appears. Use the page to configure the service operation.

   See Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Creating and Managing Integration Groups

This section provides and overview of integration groups and discusses how to:

- Add integration groups.
- Add integration subgroups.
- Add services to integration groups and subgroups.
- Rename and delete integration groups.
- Copy integration groups using Project Copy.

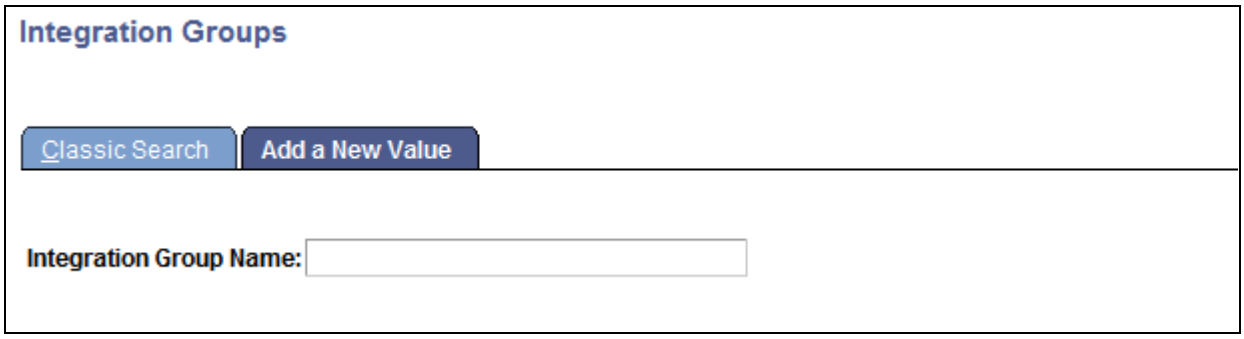
Understanding Integration Groups

PeopleSoft provides you with the ability to great groups and subgroups of integrations that you can then deploy using the Integration Network. Your PeopleSoft integration partners can then use the Integration Network to introspect and activate those integrations on their systems.

You can assign services to one or more integration groups or subgroups. However, there can only be one instance of a service within a group or subgroup assignment.

Adding Integration Groups

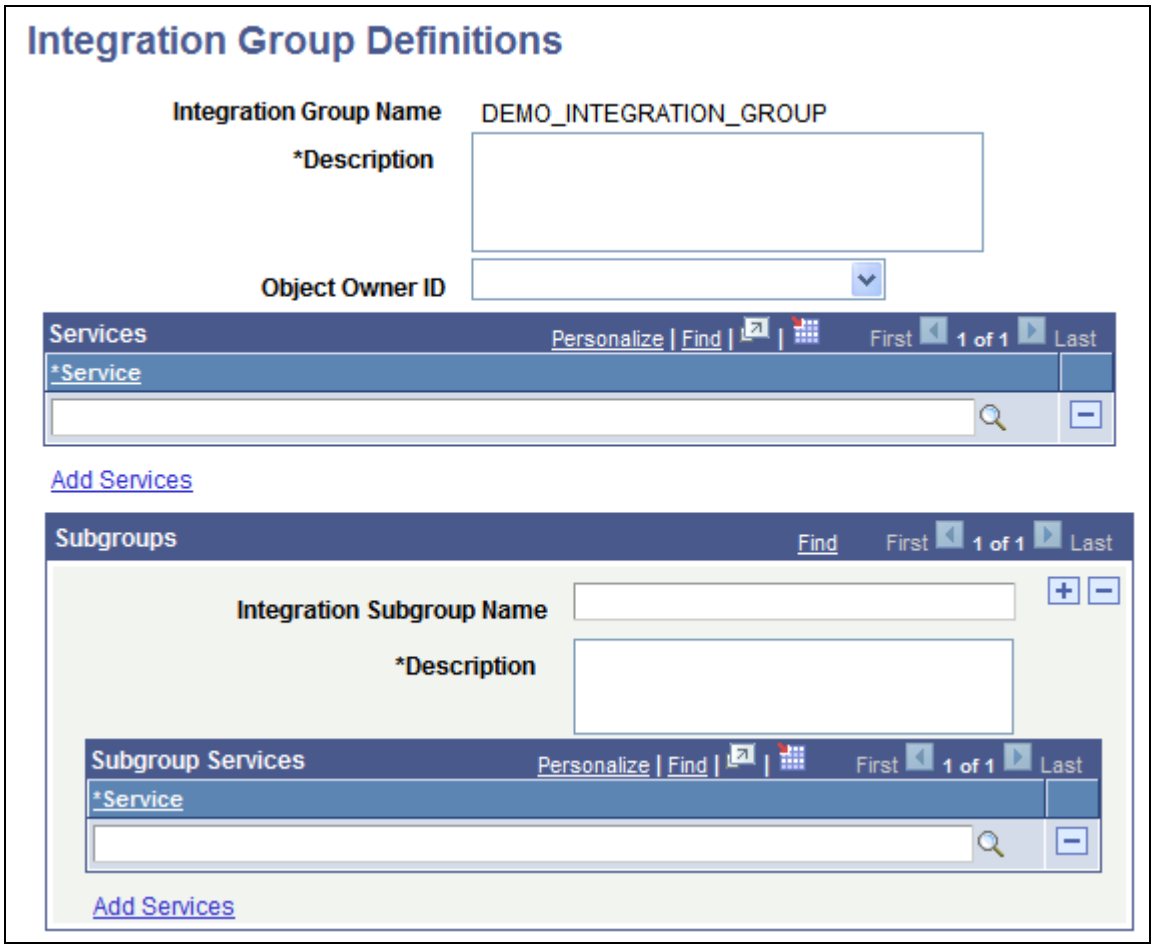
To add an integration group to the system, use the Integration Group search page shown in the following example:



Use the Integration Groups – Add a New Value tab to add an integration group to the system.

After you name the integration group, the Integration Group Definitions page (IB_INTGROUPDEFN) appears where you can add an optional description for the group and save the definition. The following example shows the Integration Group Definitions page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Integration Group Definitions page.

To add a service integration group:

1. Select PeopleTools, Integration Broker, Integration Setup, Integration Groups.

   The Integration Group search page appears.

2. Click the Add a New Value tab.

3. In the Integration Group Name field, enter a name for the integration group.

4. Click the Add button.

   The Integration Group Definitions page appears.

5. In the Description field, add a description for the group.

6. (Optional.) From the Object Owner ID field, select the owner of the object.

7. Click the Save button.

After you have added the definition to the system you can proceed to add services to the integration group, as well as create integration subgroups and add services to them.

The process for adding services to integration groups and to integration subgroups is identical and is described elsewhere in this section.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 9, "Managing Services," Adding Services to Integration Groups, page 236.

Adding Services to Integration Groups

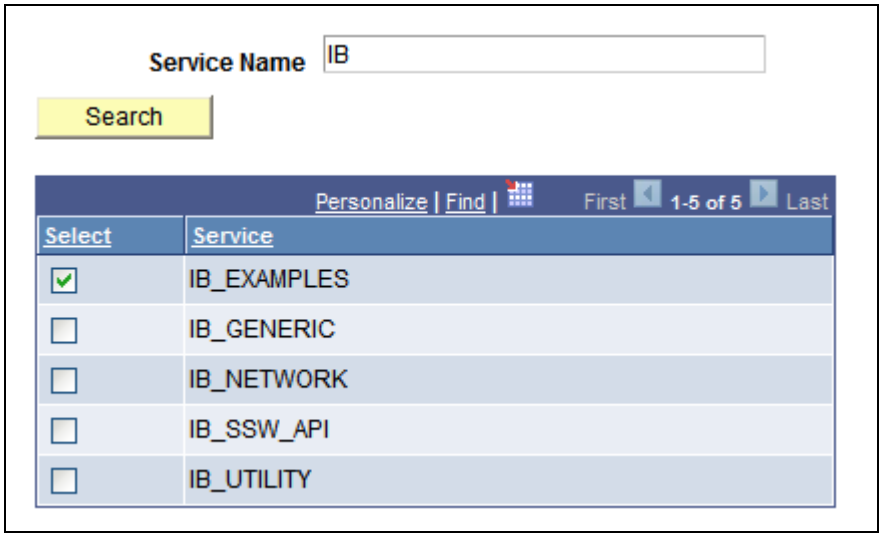
Use the Integration Group Definitions page shown in the previous section to add services to integration groups.

Note. The information presented in this section for searching for services to add to an integration group also applies to searching for services to add to integration subgroups.

You can search for the services to add to a group (or subgroup) by using the Add Services link on the Integration Group Definitions page. The following example shows the search page (IB_INTGROUP_SEC) that appears when you click the Add Services link:

Click the Add Services link to search for a specific service.

You can search for a service by entering a partial or complete service name in the Service Name field and then clicking the Search button. You can also leave the Search Name field blank and click the Search button to display a list of all services in the database. The following example shows the results of searching for a service by entering a partial name.



Searching for services by entering a partial service name.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

After you locate the service or services to add to the group, check the Select box next to each service name to add to the group and click the OK button.

To add services to an integration group:

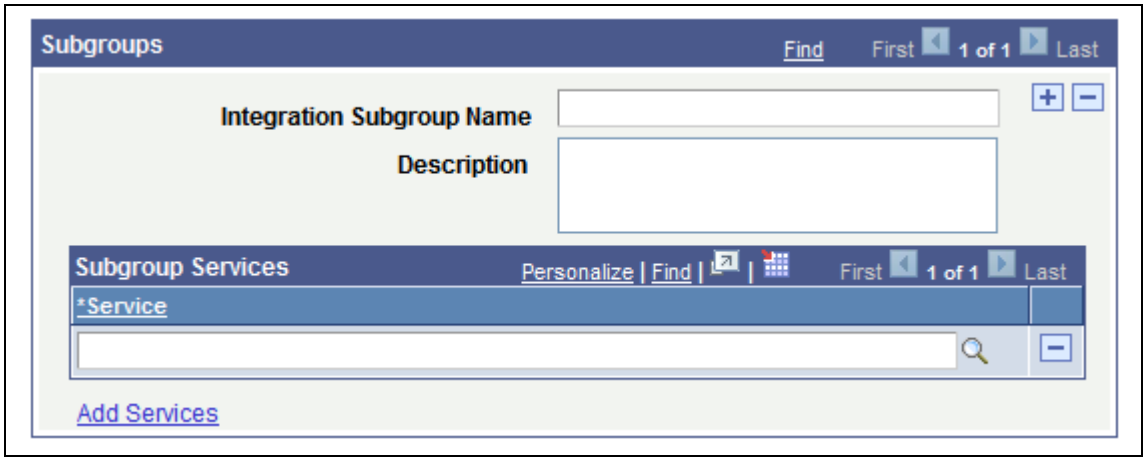
1. Access the Integration Group Definitions page. (Select PeopleTools, Integration Broker, Integration Setup, Integration Groups.)
2. Click the Add Services link.

   The services search page appears.
3. Search for one or more services to add to the integration group, as described previously in this section.
4. Click the OK button to return to the Integration Group Definitions page.
5. Click the Save button.

Adding Integration Subgroups

The Integration Group Definitions page features a Subgroups section where you define subgroups for an integration group and then add services to the subgroups.

The following example shows the Subgroups section of the Integration Group Definitions page:



Use the Subgroups section of the Integration Group Definitions page to add subgroups to an integration group.

To add a service integration subgroup:

1. Access the Integration Group Definitions page. (Select PeopleTools, Integration Broker, Integration Setup, Integration Groups.)
2. Locate the Subgroup section of the page.
3. In the Integration Subgroup Name field, enter a name for the subgroup.
4. In the Description field, enter a description for the subgroup.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Click the Add Services link to add services to the subgroup.

   The process for adding services to integration groups and to integration subgroups is identical and is described elsewhere in this section.

   See Chapter 9, "Managing Services," Adding Services to Integration Groups, page 236.

6. Click the Save button.

To add additional subgroups to the integration group, in the Subgroups section of the page, click the plus (+) button to add another subgroups grid to the page and repeat the steps described in this section to define the subgroup and add services to it.

Deleting Services from Integration Groups and Integration Subgroups

To delete a service from an integration group or an integration subgroup:

1. Access the Integration Group Definition page. (PeopleTools, Integration Broker, Integration Setup, Integration Group.)
2. In the Services or Subgroup Services section of the page, click the minus (-) button next to the name of the service to delete.
3. Click the Save button.

Renaming and Deleting Integration Groups

This section discusses how to:

- Rename integration groups.
- Delete integration groups.

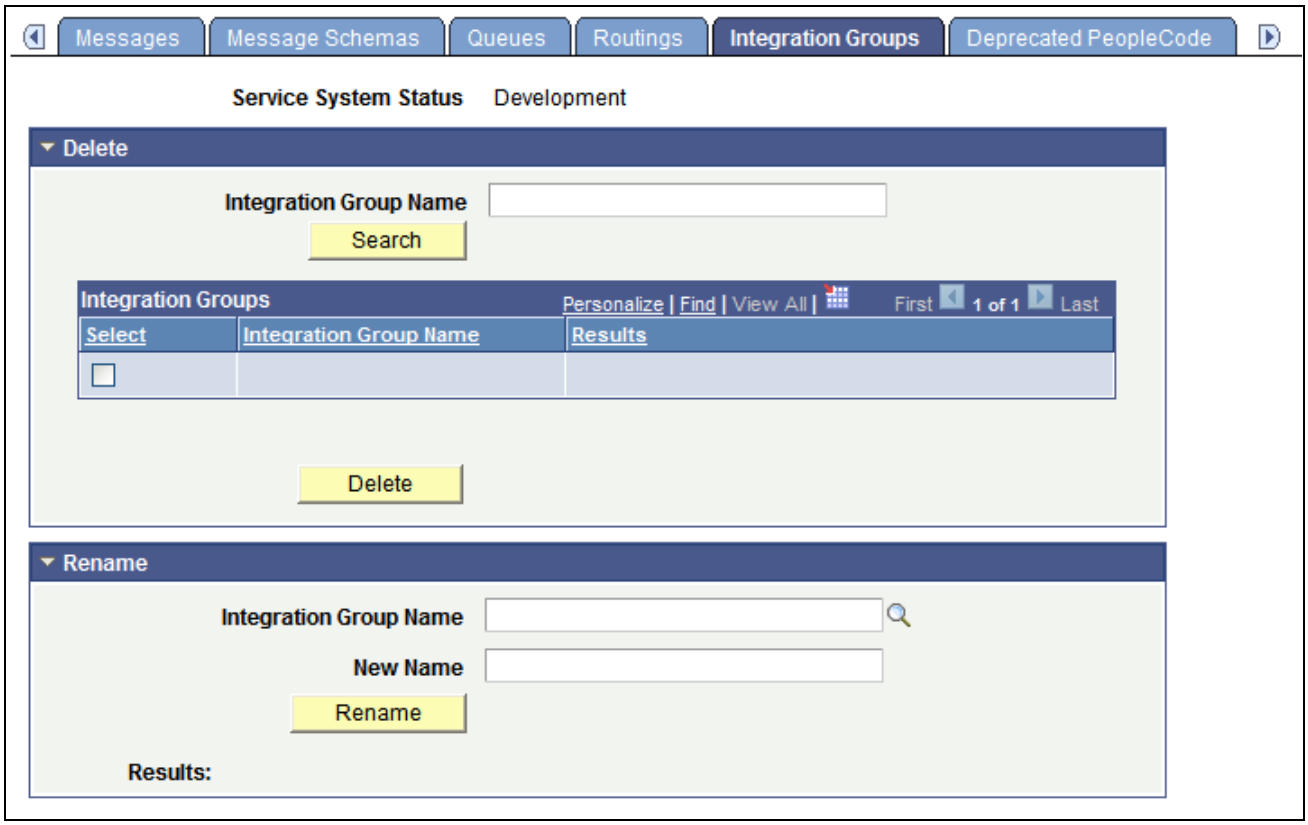
Understanding Renaming and Deleting Integration Groups

You can rename and delete integration groups using the Integration Groups tab (IB_INTGROUPADM) in the Service Administration component (IB_HOME_PAGE).

The Integration Group tab contains two sections: a Delete section that enables you to delete integration groups and a Rename section that enables you to rename integration groups. When you first access the Integration Groups tab, both sections are collapsed. Click the section header arrow buttons to expand and collapse each section.

The following example shows the Integration Groups tab with both Delete and Rename sections expanded.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Use the Integration Groups tab to rename and delete integration groups.

The service system status has no impact on renaming or deleting integration groups.

Renaming Integration Groups

To rename and integration group:

1. Access the Service Administration – Integration Groups page. (Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Click the Integration Groups tab.)
2. Click the arrow next to the Rename section header to expand the section.
3. In the Integration Group Name field, enter the service to rename, or click the Lookup button to search for and select the group to rename.
4. In the New Name field, enter the new name for the integration group.
5. Click the Rename button.

After you click the Rename button, the Results field displays a message that the action was successful or displays a warning or error message with a description of the problem.

Deleting Integration Groups

To delete and integration group:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Service Administration – Integration Groups page. (Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Click the Integration Groups tab.)
2. Click the arrow next to the Delete section header to expand the section.
3. In the Integration Group Name field, enter the integration group to delete, and click the Search button. Search results display in the results grid.
4. In the results grid, check the box next to the integration group or groups to delete.
5. Click the Delete button.

Copying Integration Groups Using Project Copy

You can copy integration groups between PeopleSoft databases using Project Copy.

When using Project Copy to copy integration groups select the Integration Groups as one of the definition types. To copy all of the service information, such as service, service operation, handler, and so on, select all of the definitions in the Related Definitions box.

Note. At a minimum you must select Services in the Related Definitions box.

Restricting and Enabling Write Access to Service Definitions

This section provides an overview of restricting access to services and discusses how to:

- Restrict write access to services.
- Enable write access to services.

Note. The information in this section applies to REST and non-REST service definitions.

Understanding Restricting Write Access to Service Definitions

When you restrict write access to a service, sensitive fields of the service definition and of associated service operations appear in read-only mode. The following table lists the components and pages that contain fields and data that are related to services and describes the impact that restricting access to services has to each of them.

Restricted Component or Page	***Restriction***	***Comments***
Service	All fields are read-only.	NA

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

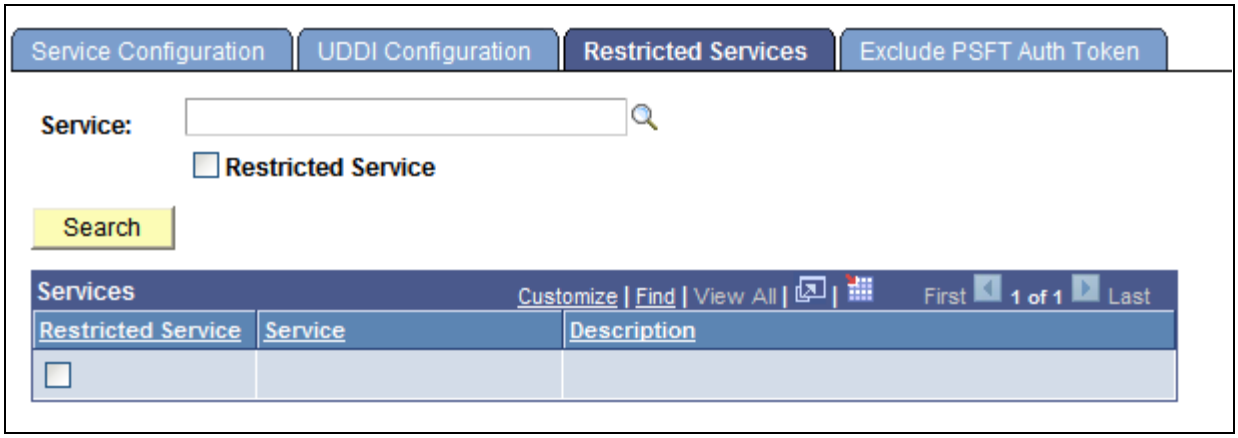
Restricted Component or Page	*Restriction*	*Comments*
Service Operation	All fields are read-only, with the following exceptions: • User Password Required. • Non-Repudiation. • Runtime Schema Validation.	When a service is restricted, you cannot regenerate routings.
Handlers	All fields are read-only except. • The Status drop-down list box. • The plus button that is used to add new handlers.	When a service is restricted, you can still activate or inactivate handlers.
Routings	All fields are read-only except: • The Inactivate Selecting Routings and Activate Selected Routings buttons. • The Add button.	When a service is restricted, you can: • Activate and deactivate routings of service operations that are associated with the service. • Add new routings to service operations that are associated with the service.

You cannot delete or rename a restricted service. In addition, you cannot change, rename or delete any service operation that is defined as part of a restricted service.

Restricting Write Access to Service Definitions

Use the Restricted Service page (IB_SVCSETUP3) in the Service Configuration component (IB_SVCSETUP) to restrict a service.

The following example shows the Restricted Services page.



Restricted Services page

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

To restrict write access to services:

1. Select PeopleTools, Integration Broker, Service Configuration. Click the Restricted Services tab. The Restricted Services page appears.
2. In the Service field, enter a service name and click the Search button, or click the Lookup button to search for a service.

   The service name or search results display in the Services list.
3. Check the Restricted Service check box next to the service name to which you want to restrict access.
4. Click the Save button.

Enabling Write Access to Service Definitions

This section describes how to enable write access to services that you previous restricted.

To enable write access to services that you previously restricted:

1. Select PeopleTools, Integration Broker, Service Configuration. Click the Restricted Services tab.

   The Restricted Services page appears.
2. Select the service to write-enable using one of the following methods:
   - In the Service field, enter a service name and click the Search button.
   - Click the Lookup button to search for a service.
   - Check the Restricted Services check box, and click the Search button to display and select from all currently restricted services in the system.

   The service name or search results appear in the Services list.
3. Clear the check box next to the service name to write access enable.
4. Click the Save button.

Renaming and Deleting Service Definitions

This section discusses renaming and deleting service definitions.

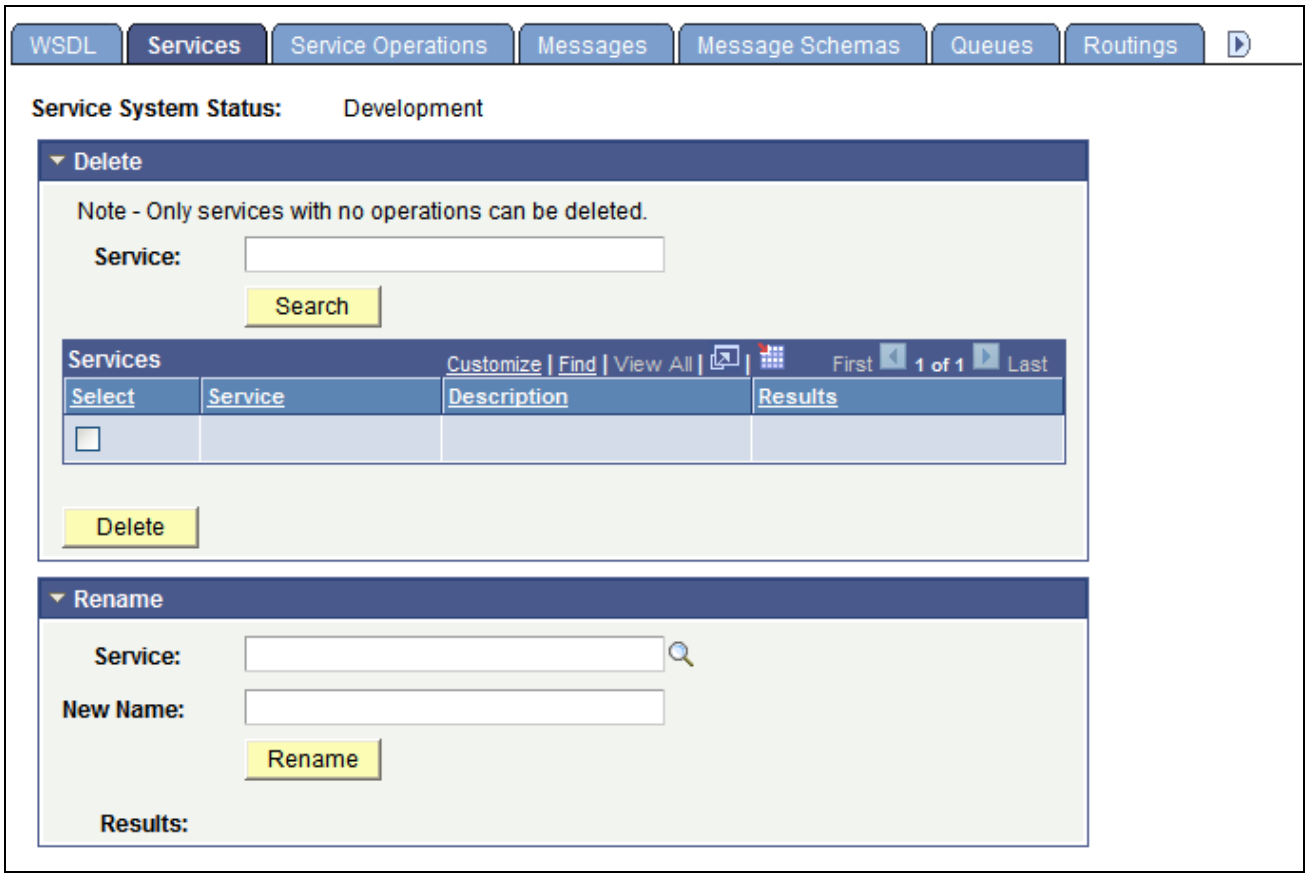
Note. The information in this section applies to REST and non-REST service definitions.

You can rename and delete services using the Services tab in the Service Administration component (IB_HOME_PAGE). The Services tab contains two sections: a Delete section that enables you to delete services and a Rename section that enables you to rename services.

When you first access the Services tab, both sections are collapsed. Click the section header arrow buttons to expand and collapse each section.

The following example shows the Services tab with both Delete and Rename sections expanded.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Services Administration – Services page with the Delete and Rename sections expanded

Renaming Service Definitions

The service system status that you set on the Services Configuration page affects the ability to rename services.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Using the Service Configuration Page to Set Service Configuration Properties.

To rename a service:

1. Select PeopleTools, Integration Broker, Service Utilities. Click the Service tab.

   The Service page displays.

2. Click the arrow next to the Rename section header to expand the section.
3. In the Service field, enter the service to rename, or click the Lookup button to search for and select the service to rename.
4. In the New Name field, enter the new name for the service.
5. Click the Rename button.

After you click the Rename button, the Results field displays a message that the action was successful or displays a warning or error message with a description of the problem.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Deleting Service Definitions

You can delete services only when the service has no service operations associated with it. When you search for a service to delete, only such services that have no service operations associated with them are retrieved from the system.

To delete a service:

1. Select PeopleTools, Integration Broker, Service Utilities. Click the Service tab.

   The Service tab displays.

2. Click the arrow next to the Delete section header to expand the section.
3. In the Service field, enter the service name to delete, and click the Search button.

   Search results display in the results grid.

4. In the results grid, check the check box next to the service or services to delete.
5. Click the Delete button.

Activating and Deactivating Service Definitions in Bulk

This section describes activating and deactivating service definitions in bulk.

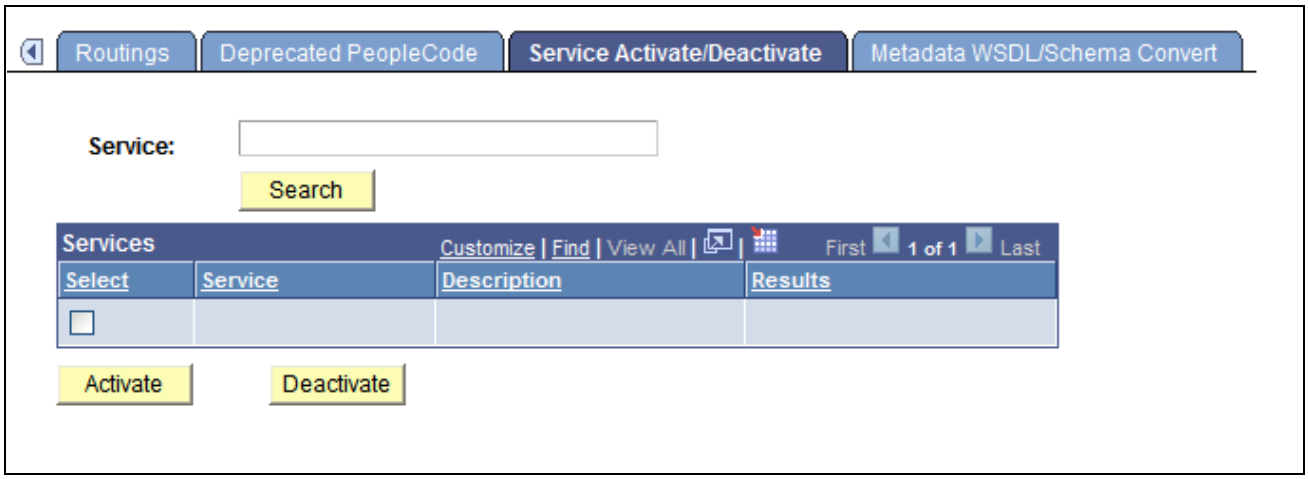
Note. The information in this section applies to REST and non-REST service definitions.

The Service Activate/Deactivate page (IB_HOME_PAGE9) in the Service Administration component enables you to activate and deactivate services in bulk.

This page enables you to quickly activate or deactivate services. If any problems occur during activation, the system displays a message indicating a problem activating a certain object, for example routing, handler, and so on. If such a message occurs, you must look on the corresponding object definition to determine the problem. This page is not intended to be used to debug activation problems or problems with integration metadata definitions.

The following example shows the Service Activate/Deactivate page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Bulk activating three service operations on the Service Activate/Deactivate page

When you activate one or more services, all of the default service operation associated with the services become active. In addition, the handler definitions, routing definitions, and queues associated with the service operations become active as well.

When you deactivate one or more services, all selected services become deactivated. The default service operations for the services become deactivated as well. In addition, the handler definitions and routing definitions associated with the service operations become deactivated. Queues do not become deactivated as they can be shared with other service operations.

To activate or deactivate services in bulk:

1. Access the Service Activate/Deactivate page (PeopleTools, Integration Broker, Service Utilities, Service Administration).
2. In the Service field, enter part or all of the service name you want to activate or deactivate.
3. Click the Search button.
4. Check the Select box next to each service to activate or deactivate.
5. Click the Activate button to activate the selected services; click the Deactivate button to deactivate the selected services.

   When the system has performed the selected action, the Results column displays *Success* for each service you activated or deactivated.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 10

Managing Service Operations

This chapter provides an overview of managing service operations and discusses how to:

- Access and view service operation definitions.
- Add service operation definitions.
- Configure service operation definitions.
- Set permissions to service operations.
- Manage service operation versions.
- Attach files to service operations.
- Assign multiple queues to process inbound service operations.
- Invoke multiple service operations.
- Rename and delete service operations.

Note. Unless noted otherwise the information in this chapter pertains to non-REST service operations.

Understanding Managing Service Operations

This section discusses:

- Service operations.
- Service operation types.
- Naming conventions for service operation metadata.
- Service operation aliases.
- Service operation versions.
- Monitoring service operations.

Note. Before you can provide or consume services in a PeopleSoft system, you must configure the system for handling services.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

Service Operations

A *service operation* definition consists of general information about an operation, such as its name, description, and so on. It also specifies an operation type, which determines how the operation is to be processed, synchronously or asynchronously. In addition, it contains routings, which determine the direction, inbound or outbound, of the service operation. A service operation has one or more handlers, which contain and run the programming logic for sending or receiving the message, manipulating message content, and so on.

Note. Beginning with the PeopleTools 8.48 release, service operations house the processing logic found in messages, transactions and relationships used in earlier versions of PeopleSoft Integration Broker.

Services Operation Types

When you create a web service operation you must specify a service operation type. *Service operation types* determine the type of message processing.

There are four service operation types:

Asynchronous Request/Response	The sending system invokes a service operation asynchronously and processes the response from the receiving system asynchronously. Unlike a synchronous operation type, the response is not processed on the same thread as the response, and it is processed sometime in the future.
Asynchronous to Synchronous	The sending system's asynchronous process sends a synchronous request to a remote system. The sending asynchronous system expects the receiving system to send a synchronous response back. The sending asynchronous system transforms the response and puts it back in the queue for asynchronous consumption.
Asynchronous – One Way	The service operation is queued and sent in near real-time. Processing on the sending system continues without a response from the receiving system.
Synchronous	The service operation is processed in real-time. Processing on the sending system does not continue until it receives a response from the receiving system.

Naming Conventions for Service Operation Metadata

When naming the following service operation metadata, names cannot start with "xml," digits or special characters:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Service operation names.
- Service operation aliases.

Service Operation Aliases

A *service operation alias* or *operation alias* is the service operation name that displays for the service operation when WSDL or WADL is provided.

Service operation aliases may be mixed case.

Duplicate service operation alias names within a service are not allowed.

Service Operation Versions

When you create a service operation, the operation that you create automatically becomes the default service operation version.

If you add a new version to the operation, the newly added version automatically becomes the default. The previous version of the service operation remains in the database as a non-default version and you still may use that the non-default version.

See Chapter 10, "Managing Service Operations," Using Non-Default Service Operation Versions, page 262.

Monitoring Service Operations

This section describes several options for monitoring service operation performance.

Monitoring Service Operations in the Integration Broker Service Operations Monitor

PeopleSoft provides an Integration Broker Service Operations Monitor that enables you to monitor asynchronous and synchronous service operation information, node status, queue status, manage domains, and more, from within the PeopleSoft Pure Internet Architecture.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook*

Monitoring Service Operations in the Integration Network Monitor

The Integration Network provides a Network Monitor that enables you to monitor local integrations and integrations from all remote PeopleSoft nodes defined in the Integration Network.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using the Integration Network," Using the Integration Network Monitor.

Monitoring Service Operations in the Integration Network Transactional Tracker

The Integration Network provides a Transactional Tracker that enables you to monitor transactions between the local system and remote PeopleSoft nodes defined in the Integration Network.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using the Integration Network," Using the Integration Network Transactional Tracker.

Monitoring Service Operations in Performance Monitoring Tools Using Module and Action Information

Several system monitoring tools report metrics by capturing Module and Action information. On an Oracle database, PeopleSoft provides service operation name and PeopleCode event name as Module name and Action name respectively. On an IBM DB2 database, PeopleSoft provides service operationname as Module name (no Action information is provided for DB2 systems). This information enables you to associate service operations with database transactions for monitoring and troubleshooting.

To enable capturing service operation and PeopleCode event metrics you must set the EnableDBMonitoring option in PSADMIN.

See *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on Oracle," Monitoring PeopleSoft MODULE and ACTION Information; *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on DB2 UDB for z/OS," Associating PeopleSoft Users, Modules, and Actions with DB2 UDB Threads and *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on DB2 UDB for Linux, UNIX, and Windows," Monitoring Module and Action Information.

Accessing and Viewing Service Operation Definitions

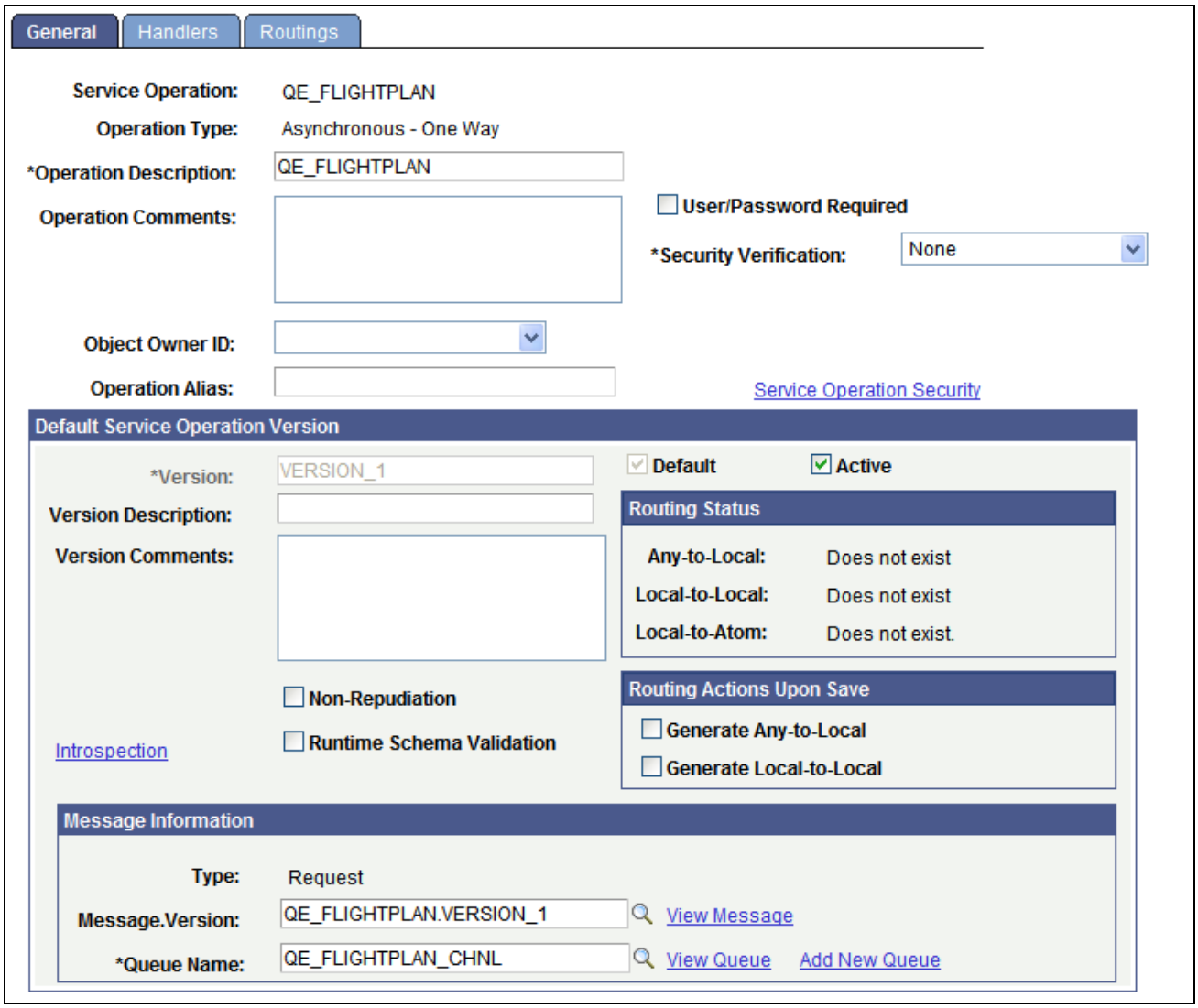
This section discusses how to:

- Access service operation definitions.
- View service operation definitions.

Accessing Service Operation Definitions

Use the pages in the Service Operations component (IB_SERVICE) to access and view service operation definitions. The following example shows the General tab of the Service Operations component for the QE_FLIGHTPLAN web service operation:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



QE_FLIGHTPLAN web service operation appearing in the Service Operations – General tab.

To access a service operation definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Service Operations.

   The Service Operations– Search page appears.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Search for a service operation definition to view.

   You can search for an operation in one of two ways:

   - Click the Search button to display all service operation definitions in the system.
   - Enter search criteria in one or more of the following fields, and then click the Search button:

   Service Enter the service name that contains the service operation that you want to view, or click the Lookup button to search for and select a service name.

   Service Operation Enter the name of the service operation to view, or click the Lookup button to search for and select an operation.

   Operation Type This search option is for web service operations only.

   From the Operation Type drop-down list box, select an operation type. Values are:

   - *Asynch Request/Response*
   - *Asynch to Synch*
   - *Asynchronous – One Way*
   - *Synchronous*

3. In the Service Operations results grid, click the name of the service operation to view.

   The Service Operations – General tab appears with data for the service operation that you selected.

Viewing Service Operation Definitions

The Service Operations component includes three pages:

General Tab Features general-service and default-service operation information.

Handlers Tab Provides summary information about handlers that have been added to an operation. Service operation handlers contain the programming logic for sending and receiving service operations and their contained messages, and for manipulating content.

Routings Tab Provides summary information about service operations routings. Routing definition determine the direction—inbound, outbound or hub—of service operations.

Viewing General Service Operation Information

When you access a service operation (PeopleTools, Integration Broker, Integration Setup, Service Operations, the Service Operations - General page (IB_SERVICE) appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The top portion of this page contains basic information about a service operation, including its name, description, and so on. The Service Operation Security link opens the permission list for the service. Note that the Service Operation Security link appears only after a service operation definition is saved.

The Default Service Operation Version section displays service-operation version information, whether nonrepudiation is in effect, and whether runtime schema validation is enabled. The Introspection link enables you to access the Introspection and Validation page.

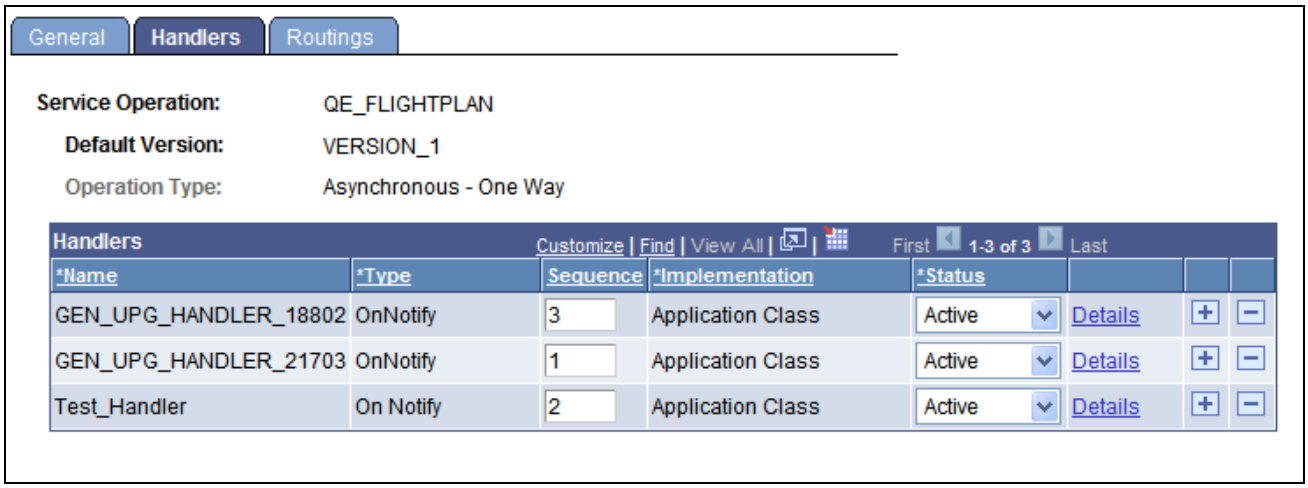
The Routing Status group box shows if any-to-local, local-to-local, or in the case of feeds, local-to-Atom, routing definitions have been generated for the service operation. Click the Routings tab to view detailed information about routing definitions that exist for a service operation.

The Routing Actions Upon Save group box shows the possible routings that the system can generate when the service operation definition is saved.

The Message Information section displays the request message, response message information, and fault message for the service operation. The View Message links in this section open the displayed message on the Message Definition page, where you can view additional information about the message. For messages defined for web services, all service operations with operation types other than Synchronous, the queue to which a message belongs also appears. Click the View Queue link or the Add New Queue link to open the Queue Definition page to view additional queue definition information or to add or change a queue to which a message belongs.

Viewing Handler Information

To view service operation handler information, click the Handler tab. Doing so displays the Service Operations - Handlers page (IB_SERVICEHDLR) which lists summary information about handlers that have been added to an operation.

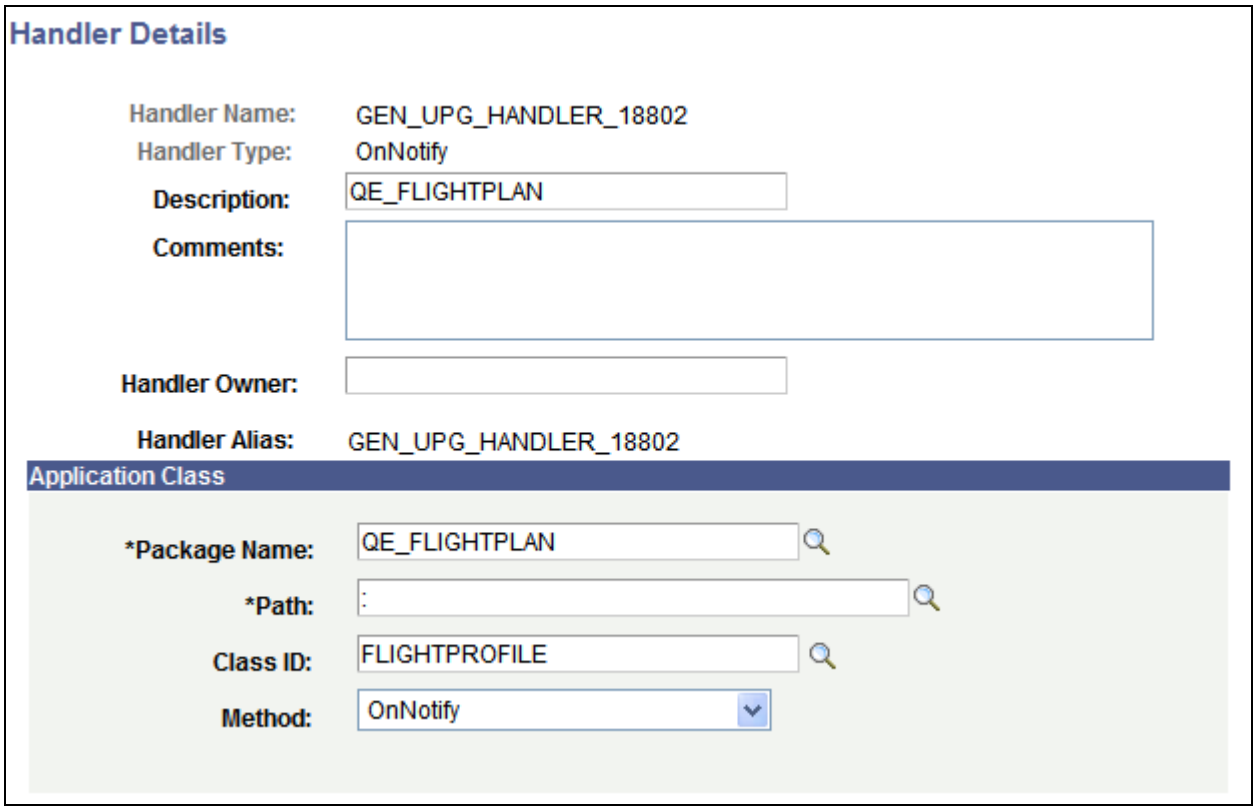


Service Operations – Handlers page

The summary information includes the handler name, the handler type, and the implementation method for the handler. The status of the handler, active or inactive, also appears.

Click the Details link to open the Handler Details page (IB_SERVICEHDLR_SEC) for the handler. The following example shows the Handler Details page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



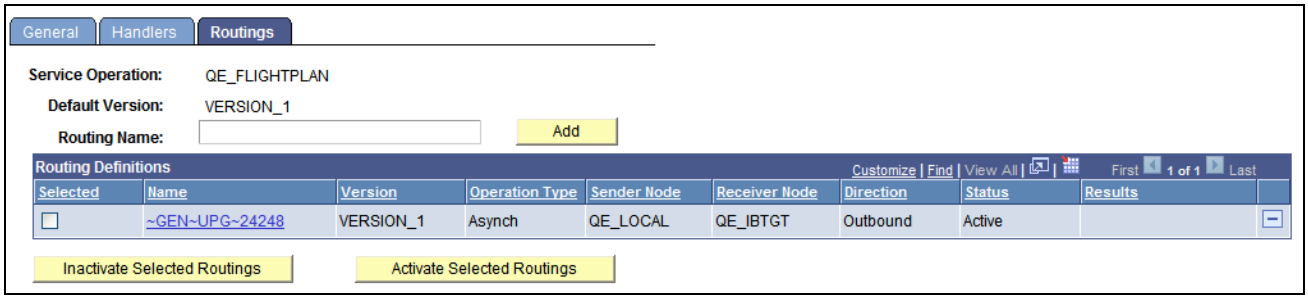
Handler Details page

The Handler Details page shows additional information about the handler, including the owner and application class or component interface details.

You can also use this page to specify the handler details.

Viewing Routing Information

Click the Routing tab to open the Service Operations - Routings page (IB_SERVICERTNGS) to view a summary of routing definitions for an operation.



Service Operations–Routings page

The Routings Definition grid on the page lists summary information for routings that are defined for a service operation. Summary information that is displayed includes the routing definition name, service operation version, routing type, sending node, receiving node, direction of the routing and the routing status.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Click a routing definition name to open the routing in the Routing Definitions component, where you can view additional information about the routing.

See Chapter 17, "Managing Service Operation Routing Definitions," Viewing Routing Definitions in Graphical Format, page 404.

You can also use this page to add routing definitions to a service operation and to activate or deactivate routings for an operation.

See Chapter 10, "Managing Service Operations," Adding Routing Definitions, page 259 and Chapter 10, "Managing Service Operations," Activating and Inactivating Routing Definitions, page 260.

Adding Service Operation Definitions

This section describes how to add web service operation definitions to the PeopleSoft system.

To add a service operation definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Service Operations.

   The Service Operations – Search page appears.

2. Click the Add a New Value link.
3. In the Service field, enter the service name to which the new operation will belong or click the Lookup button to search for a service name.
4. In the Service Operation field, enter a name for the service operation.
5. From the Operation Type drop-down list box, select an operation type. Values are:
   - *Asynchronous – One Way*
   - *Synchronous*
   - *Asynch Request/Response*
   - *Asynch to Synch*
6. Click the Add button.

The new definition appears on the General tab of the Service Operation component, and you can now define the service operation.

Configuring Service Operation Definitions

After you add a service operation definition to the system, you can define the service operation.

This section discusses how to:

- Specify general service operation information.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Define service operations version information.
- Add handlers to service operations.
- Add routing definitions.
- Activate and inactivate routings.

Specifying General Service Operation Information

To specify general service operation information:

1. Access the Service Operations-General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Operation Description field, enter a description for the operation.
3. (Optional.) Check the User ID/ Password Required check box to require a user ID and password for inbound service operations.

   See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Managing User Authentication.
4. From the Security Verification drop-down list, select the level of security with which inbound integrations must be sent.

   See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Validating Security on Inbound Integrations.
5. (Optional.) In the Operation Comments field, enter comments about the service operation.
6. (Optional.) From the Object Owner ID field, select the owner of the definition.

   The owner ID helps to determine the application team that last made a change to a service definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.
7. (Optional.) In the Operation Alias field, enter an alias name for the service operation.

When working with a web service operation, the general information section of this page includes a Service Operation Security link. Granting permissions to service operations is discussed elsewhere in this chapter.

Before you can save the service operation definition, you specify messages for the service operation, as described in the next section.

See Chapter 10, "Managing Service Operations," Setting Permissions to Service Operations, page 260.

Defining Service Operation Version Information

When you first create a service operation definition, the definition that you initially define is the default version.

When the newly created service operation definition opens, the Default check box is enabled and is read-only.

You can subsequently define additional service operation versions and assign them as the default.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 10, "Managing Service Operations," Managing Service Operation Versions, page 261.

Defining General Version Information

To define the service operation default version:

1. Access the Service Operations - General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Version field, enter a version identifier.

   The default is *v1*.
3. (Optional.) In the Version Description field, enter a description for the operation version.

   If you enter no information, the description by default is the name of the service operation when you save the definition.
4. (Optional.) In the Version Comments box, enter comments about the version.
5. (Optional.) Check the Non-Repudiation check box to apply nonrepudiation to the message.
6. (Optional.) Check the Runtime Schema Validation check box to enable service schema validation at runtime.

You can click the Introspection link to employ introspection to generate point-to-point routings.

See Chapter 17, "Managing Service Operation Routing Definitions," Using Introspection to Create Routing Definitions, page 396.

Continue to the next section to specify messages for service operations. You cannot save the service operation definition until you define messages for it.

Specifying Messages for Service Operations

You specify messages for service operations in the Message Information section of the Service Operations – General page.

The messages that you specify define the structure of the data that is contained in the service operation.

The service operation type determines the number of messages and message types (request or response) that you specify.

To specify messages for a service operation:

1. Locate the Message Information section on the Service Operations – General page.
2. Locate the Type field, and take note of the message type to define.
3. In the Message.Version field, enter the message name followed by a dot and version, or click the Lookup button to search for one.

   After you select the message, you can click the View Message link to view the message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Specify the queue for the message.

   Note. If you are defining a message for a synchronous operation type, you do not need to define a queue.

   Your options are:

   - In the Queue Name field, enter the queue name.
   - Click the Lookup button to search for a queue.
   - Click the Add Queue link to open the Queue Definitions page and define a new queue for the message.

     See Chapter 13, "Managing Service Operation Queues," Adding Queue Definitions, page 321.

   - Click the Multi-Queues link to open the Queue List page and assign multiple queues to process the message.

     The Multi-Queue link appears only when you have enabled the multi-queue feature and when you are working with asynchronous service operations.

     See Chapter 10, "Managing Service Operations," Assigning Multiple Queues to Process Service Operations, page 265.

5. Repeat steps 1 through 4 for each message type that appears in the Message Information section.
6. Click the Save button.

Specifying Fault Messages for Service Operations

You can specify fault messages for service operations for error handling.

Note the following about fault messages:

- You cannot add fault messages to asynchronous service operations.
- Fault messages must be nonrowset-based messages, container messages, or document messages. Fault messages cannot be rowset-based messages.

To specify a fault message:

1. Locate the Default Service Operation Version section on the Service Operations – General tab.
2. Click the Add Fault Type button.

   A new row appears in which to specify a message. Note that the Type field in the new row displays *Fault.*

3. In the Message.Version field, enter the message name, or click the Lookup button to search for one.

   After you select the message, you can click the View Message link to view the message.

4. Click the Save button.

To delete a fault message, in the Default Service Operation Version section, click the Delete Fault Type button. Then click the Save button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Generating Local-to-Local and Any-to-Local Routing Definitions

Use the Service Operations-General page to initiate generating local-to-local and any-to-local routing definitions.

See Chapter 17, "Managing Service Operation Routing Definitions," Initiating System-Generated Routing Definitions, page 377.

Adding Handlers to Service Operations

Adding handlers to service operations is discussed elsewhere in this PeopleBook

See Chapter 16, "Managing Service Operation Handlers," page 347.

Adding Routing Definitions

This section describes how to create point-to-point service operation routing definitions from the Service Operations – Routing page.

Note. You can also create routings using the Routings component, the Introspection component, or the Routings page in the Node Definitions component.

See Chapter 17, "Managing Service Operation Routing Definitions," Adding Routing Definitions, page 379; Chapter 17, "Managing Service Operation Routing Definitions," Using Introspection to Create Routing Definitions, page 396 and *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Adding and Configuring Nodes."

See Chapter 17, "Managing Service Operation Routing Definitions," Routing Types, page 371.

To add a routing to a service operation:

1. Access the Service Operations - Routings page (PeopleTools, Integration Broker, Integration Setup, Service Operations, click the Routings tab).
2. In the Routing Name field, enter a name for the routing.
3. Click the Add button.

   The Routing Definition page appears.

   Creating and defining a routing is discussed elsewhere in this PeopleBook.

   See Chapter 17, "Managing Service Operation Routing Definitions," Adding Routing Definitions, page 379.

The next section describes how to activate routings.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," page 371

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Activating and Inactivating Routing Definitions

To activate or inactivate a routing:

1. On the Service Operations component, click the Routings tab.

   The Routings page appears.

2. Check the box in the Select column next to the routing definition names that you want to activate or inactivate.

3. Activate or inactivate the routing definition.

   - To activate the routings, click the Activate Selected Routings button.
   - To inactivate the routings, click the Inactivate Selected Routings button.

4. Click the Save button.

Setting Permissions to Service Operations

This section describes how to use the Service Operations component to set permissions to access service operations. You can also set these permission in the Security component.

Understanding Setting Permission to Service Operations

Security operations are secured using permission lists.

When you select the User/Password Required check box on the Service Operations-General page, on inbound integrations, your integration partners must supply a valid user ID that is associated with the permission list you specify to invoke service operations.

Setting Permission Access to Service Operations

To grant permissions to service operations:

1. Select PeopleTools, Integration Broker, Integration Setup, Service Operations, and select a service operation with which to work.

   The Service Operations - General page appears.

2. Click the Service Operation Security link.

   The Web Service Access page appears.

3. In the Permission List field, enter a permission list for the service operation, or click the Lookup button to search for and select one.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. From the Access drop-down list, select an access level for the service operation. Values are:
   - *Full Access.* (Default.)
   - *No Access.*
5. Click the Save button.

See Also

PeopleTools 8.52: Security Administration, "Setting Up Permission Lists"

Managing Service Operation Versions

This section discusses how to:

- Create service operation versions.
- Use non-default service operation versions.

Creating Service Operation Versions

The information in this section pertains to REST and non-REST service operations.

When you create a new service operation version, the new version automatically becomes the active default version.

If you have generated WSDL or WADL for the current service operation, after you create the new version you are prompted to generate WSDL or WADL for the new version.

To create a new service operation version:

1. Select PeopleTools, Integration Broker, Integration Setup, Service Operations.

   Select the service operation with which to work. The Service Operations–General page appears.
2. At the bottom of the page, click the Add Version link.

   The Service Operation Versions page appears.
3. In the Service Operations Version field, enter the new version and click the OK button.

   The Service Operations Version page appears.
4. Complete the fields as appropriate for the new service operation version.

   See Chapter 10, "Managing Service Operations," Defining Service Operation Version Information, page 256 and Chapter 12, "Managing REST Service Operations," Configuring REST Service Operation Definitions, page 297.
5. Click the Save button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The new service operation version appears in the Service Operations–General page.

A Non-Default Versions grid appears at the bottom of the page that lists and provides access to the previous service operation version. Note that all versions that display in this grid have a status of *Inactive.*

Using Non-Default Service Operation Versions

PeopleCode processing for service operation handlers takes place on the default service operation version.

To continue using non-default service operation version you must write and apply transformations to transform message shapes contained in the non-default service operation version to the message shapes contained in the default version so that handler processing of the service operation can occur.

You need to write and apply transformations only for the changed message shapes. For example, if a service operation contains request and response messages, but only the request message shape has changed between versions, you need only write and apply a transform program to transform the request message on the request message that is contained in the non-default service operation version to the shape of the request message in the default version.

The non-default versions are inactive until the transformations are entered and the status is changed to Active. Then the grid shows the version with Active.

Attaching Files to Service Operations

This section provides an overview of attaching files to service operations and discusses how to:

- Use the FTP Attachment utility.
- Send attachment information with service operations.
- Process attachment information that is included in service operations.

Understanding Attaching Files to Service Operations

PeopleSoft Integration Broker provides an FTP Attachment Upload utility that enables you to upload attachments from any directory to your FTP server and then provide the location of the attachments in service operation PeopleCode. The attachments can be in any file format, such as text files, EDI files, word processing files, and so on.

As the operation is consumed, you can access these attachments using the attachment API. The recipient can get the necessary information about the attachment and can retrieve it using a URL or file location that you provide.

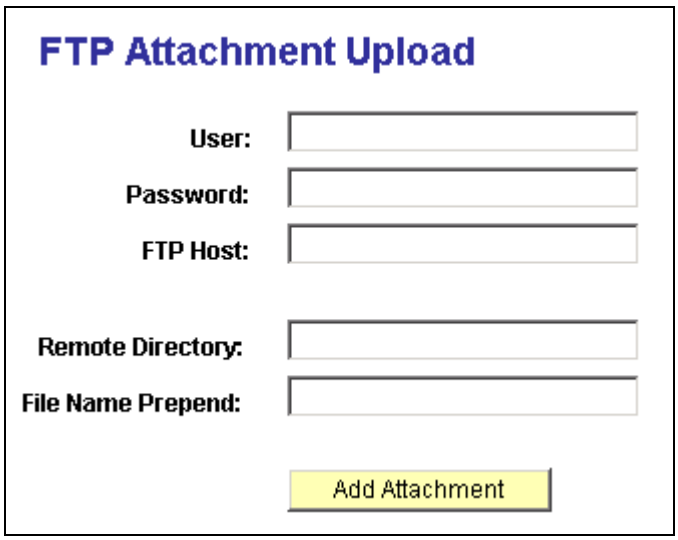
Note. The FTP Attachment Upload utility does not support uploading attachments from the database. To upload attachments from the database, manually retrieve and copy them to the FTP server.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Using the FTP Attachment Utility

Use the FTP Attachment Upload page in the Files Utilities component (IB_FILEUPLOAD) to upload files to your FTP server for attaching to service operations.

To access the FTP Attachment Upload page, select PeopleTools, Integration Broker, File Utilities, File Upload. The page is shown in the following example:



FTP Attachment Upload page

You work with the following page elements:

User	Indicates the user ID of the FTP server.
Password	Indicates the password to the FTP server.
FTP Host	Indicates the machine name of the FTP server.
Remote Directory	Indicates the directory path to the file to upload.
File Name Prepend	Enter text to prepend the file name to build the final file name to copy to the target directory.
Add Attachment	Click to upload the indicated file.

Sending Attachment Information with Service Operations

The following example shows sample PeopleCode for sending attachment information:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &MSG;
Local Rowset &Flight_Profile;
Local String &Attachment_id;

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage (Operation.ASYNC_RR);

&Attachment_id = &MSG.IBInfo.AddAttachment (c:\\temp\\myfile.txt);

&attachReturn = &MSG.IBInfo.SetAttachmentProperty(&Attachment_id,
     %Attachment_Encoding, "UTF-8");

&attachReturn = &MSG.IBInfo.SetAttachmentProperty(&Attachment_id,
     %Attachment_Base, "Standard");

&attachReturn = &MSG.IBInfo.SetAttachmentProperty(&Attachment_id,
     %Attachment_Disposition, "Pending");

&attachReturn = &MSG.IBInfo.SetAttachmentProperty(&Attachment_id,
     %Attachment_Language, "English");

&attachReturn = &MSG.IBInfo.SetAttachmentProperty(&Attachment_id,
     %Attachment_Description, "Parts data");

&MSG.CopyRowset (&FLIGHT_PROFILE);

%IntBroker.Publish(&MSG);
```

Processing Attachment Information Included in Service Operations

The following example shows sample PeopleCode for processing an attachment from a notification:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class FLIGHTPROFILE implements PS_PT:Integration:INotificationHandler;
     method FLIGHTPROFILE();
     method OnNotify(&MSG As Message);

end-class;

/* Constructor */
method FLIGHTPROFILE
     %Super = create PS_PT:Integration:INotificationHandler();
end method;

method OnNotify
     /+ $MSG as Message +/
     /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/

     Local Rowset &rs;
     Local integer &count;
     Local string &Attachment_ID &Results;

     &rs = &MSG.GetRowset();

     &count = &MSG.IBInfo.NumberOfAttachments;
     If &count > 0 Then

          &Attachment_ID = &MSG.IBInfo.GetAttachmentContentID(1);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_Encoding);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_Type);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_URL);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_Base);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_Location);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
            %Attachment_Disposition);
          &Results = &MSG.IBInfo.GetAttachmentProperty(&Attachment_ID,
             %Attachment_Description);

     End-If;
     /* Process data from message */

end-method;
```

Assigning Multiple Queues to Process Service Operations

This section discusses how to:

- Enable multi-queue service operation processing.
- Specifying multiple queues to process service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Assigning Multiple Queues to Process Service Operations

You can assign a service operation to multiple queues for unordered processing.

The Service Operations page features a Multi-Queues link that enables you to specify multiple queues to handle inbound or outbound service operations. When you use the multi-queue option, the requests are split across multiple queues.

The Multi-Queues link is enabled only when a message is assigned to an unordered queue.

Before you can use this feature you must enable it in the Service Configuration page.

Enabling Multi-Queue Service Operation Processing

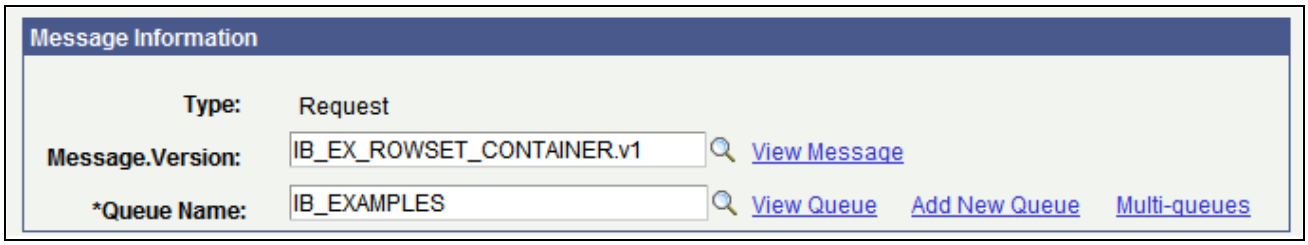
Before you can assign multiple queues to process a service operation, you must enable the multi-queue processing feature on the Service Configuration page.

To enable multi-queue service operation processing:

1. Access the Service Configuration page (PeopleTools, Integration Broker, Configuration, Service Configuration).
2. Select the Enable Multi-Queue box.
3. Click the Save button.

Specifying Multiple Queues to Process Service Operations

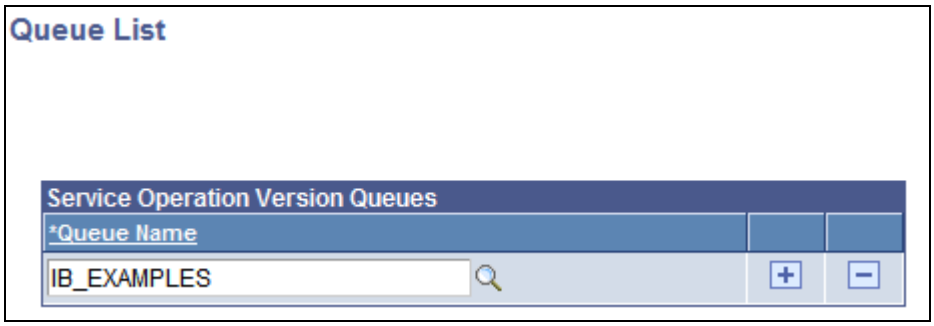
When a service operation contains a message assigned to an unordered queue, the Service Operations–General page displays a Multi-Queues. The Multi-Queues link displays in the Message Information section of the page, as shown in the following example:



The Multi-Queues link appears in the Message Information section of the Service Operations-General page on the same row as Queue Name, View Queue and Add New Queue links

When you click the Multi-Queues link, the Queues List page shown in the following example appears:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The Queues List page shows the queues that have been assigned to process a service operation.

The Queues List page shows the queues assigned to process a service operation. Use Plus button (+) to add new rows and use the Lookup button to search for and specify additional queues to process the service operation. Note that when you use the Lookup button to search for a queue, only those queues that are active and unordered are available to select.

To assign multiple queues to process an inbound service operation:

1. Access the Service Operations–General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Message Information section, click the Multi-Queues link. The Queue List page appears.
3. Click the Plus button (+) to add a new row. A new row appears.
4. Click the Lookup button to search for an select an unordered queue.
5. Repeat steps 3 and 4 to assign additional queues to process the service operation.
6. Click the OK button. The Service Operations–General page appears.
7. Click the Save button.

To remove a queue, click the Minus button (-) next to the queue name to remove. Click the OK button to return to the Service Operations–General page. Then, click the Save button.

Invoking Multiple Service Operations

You can invoke multiple service operations on the receiving system using a single inbound service operation. This process is sometimes referred to as service operation fan out.

Invoking multiple service operations using a single inbound service operation applies to asynchronous service operations only.

To invoke multiple service operations on the receiving system using one inbound service operation, you must specify the name of the inbound service operation in the External Alias field on the routing definition of each service operation on the receiving system that you want to invoke.

The service operations on the receiving system do not need to be assigned to the same queue; PeopleSoft Integration Broker will invoke service operations assigned to different queues.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

If the sending system sends in a partition key (using the SubQueue HTTP listening connector or PeopleSoft services listening connector parameter), the receiving PeopleSoft system will honor it. If there is a transformation defined for the service operation, partitioning occurs after the transformation. If the sending system does not send in a partition key, the receiving PeopleSoft system applies partitioning according to the queue definition defined for the service operation.

Renaming and Deleting Service Operations

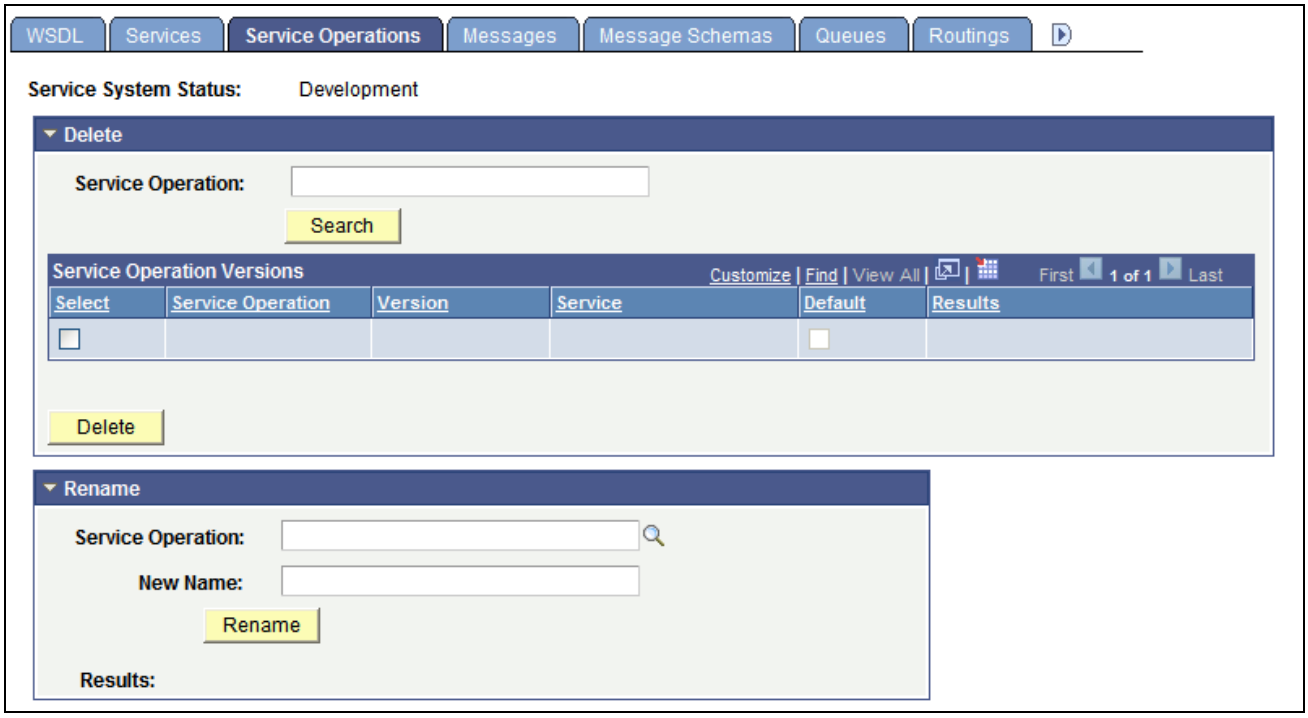
The information in this section pertains to REST and non-REST service operations.

You can rename and delete service operations using the Services Operations page in the Service Administration component. The Services Operations page contains three sections: a Delete section that enables you to delete service operations, a Rename section that enables you to rename service operations, and a Service Change section that enables you to change the service with which a service operation is associated.

To access the page, select PeopleTools, Integration Broker, Service Utilities, Services Administration and click the Service Operations tab.

When you first access the Services Operations page, all sections are collapsed. Click the section header arrow buttons to expand and collapse each section.

The following example shows the Services Operation page with the Delete and Rename sections expanded:



Services Administration – Service Operations page with the Delete and Rename sections expanded

At the top of the page, the Service System Status displays the current setting. The service system status, set in the Services Configuration page, impacts the ability to rename and delete messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services and *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Using the Service Configuration Page to Set Service Configuration Properties.

Renaming Service Operations

Renaming a service operation is allowed only if the operation is not referenced in any runtime table. If a service operation is referenced in a runtime table, you must archive the data before you can rename the operation.

You cannot rename service operations associated with the restricted services *IB_UTILITY* and *IB_GENERIC* .

To rename a service operation:

1. Access the Services Administration – Service Operations page.
2. Click the arrow next to the Rename section header to expand the section.
3. In the Service Operation field, enter the service to rename, or click the Lookup button to search for and select the service operation to rename.
4. In the New Name field, enter the new name for the service.
5. Click the Rename button.

Deleting Service Operations

You can delete service operations individually, with the exception of the default service operation version. If you mark the default service operation for deletion, the system marks all versions for deletion and the entire service operation is deleted.

When you delete a service operation, you are actually deleting a version of a service operation. A service operation cannot exist without at least one default version.

Note. If you delete the default version of a service operation, the system deletes all versions of the service operation.

Deleting a service operation is allowed only if the operation is not referenced in any runtime table. If a service operation is referenced in a runtime table, you must first archive the data before deleting the service operation. Use the Service Operations Monitor to archive data.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Archiving Service Operation Instances."

You cannot delete service operations associated with the restricted services *IB_UTILITY* and *IB_GENERIC* .

To delete a service operation:

1. Access the Services Administration – Service Operations page.
2. Click the arrow next to the Delete section header to expand the section.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. In the Service Operations field, enter the service operation name to delete and click the Search button. Search results appear in the results grid.
4. In the results grid, check the box next to the service operation or service operations to delete.
5. Click the Delete button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 11

Managing REST Services

This chapter discusses how to:

- Search for REST service definitions.
- Add REST service definitions.
- Add service operations to REST service definitions.
- Restrict write access to REST service definitions.
- Rename and delete REST service definitions.
- Activate and deactivate REST services in bulk.

Important! Before you can provide or consume services in a PeopleSoft system, you must configure the system for handling services.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

Understanding Managing REST Services

As non-REST services are used to logically group a set of non-REST service operations, REST services are used to logically group a set of REST service operations.

For example, if you have a number of REST service operations that are related to customers, such as those pertaining to customer information, adding customers, updating customers, deleting customers, and so on, you can create a customer REST service and then associate the related REST service operations with that REST service.

Common Elements Used in this Chapter

Comments	(Optional.) Enter comments about the service or service definition.
Description	Description of the service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Is Provider This option appears when the service is defined as a REST-based service.

Check the box to define the service as a provider service.

Link Existing Operations Click to add service operations already defined in the system to a service.

Object Owner ID (Optional.) Indicates the owner of the service.

The owner ID helps to determine the application team that last made a change to a service definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.

Operation Type Specifies how the service is transmitted.

On the Service page this field defines the operation type of the service operation added.

Provide Web Service Click to launch the Provide Web Services component and export a PeopleSoft REST service as a WADL document.

This link appears only when the service is a provider REST service.

REST Method Indicates the HTTP method for manipulating REST resources.

REST Service The service is defined as a REST-based service when this box is checked.

Schema Namespace Provides qualification for attributes and elements within an XML schema document (XSD).

The default is *http://xmlns.oracle.com/Enterprise/Tools/schemas.*

The namespace on the message definition defaults to the schema namespace you set as the default on the Service Configuration page.

Note. If you change the namespace, all future messages will have the new namespace.

Service The name of the service.

Service Alias (Optional.) Overrides the service name and will be the name of the service when the WSDL is provided or exported. The alias enables you to use mixed case in the name.

Service Operation The name of the service operation to associate with the service.

On the Services page, use this field to add new service operations for the current service.

Namespace The namespace field on the Service pages provides qualification for attributes and elements within a WADL document.

The value defined in the Service Namespace field in the Service Configuration page is used as the default service namespace on the Services page. The default value is *http://xmln.oracle.com/enterprise/tools/service.*

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service System Status	The status that is selected restricts rename, delete, and other administrative actions that users can perform on integration metadata in the Services Administration component. Values are: • *Production.* • *Development.* See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services.
View WADL	Click the link to view WADL documents in the WSDL repository that have been generated for the service This link appears only when the service you are viewing is a REST-based provider service.

Accessing and Viewing REST Service Definitions

This section discusses how to:

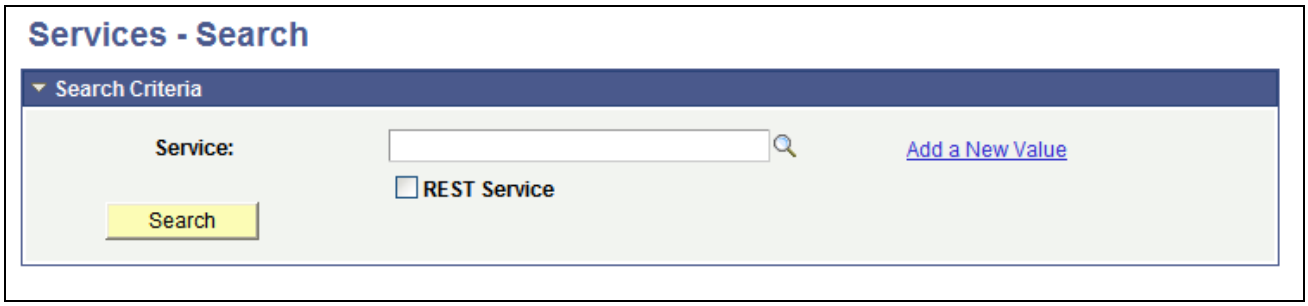
- Access REST service definitions.
- View WADL documents for REST services.
- View REST service operation information.
- View messages defined for REST service operations.

Accessing REST Service Definitions

REST service definitions appear on the Services page (IB_SERVICEDEFN) in the Service component (IB_SERVICEDEFN).

Use the Services-Search page to search for and access REST definitions in the database. The following example shows the Service-Search page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

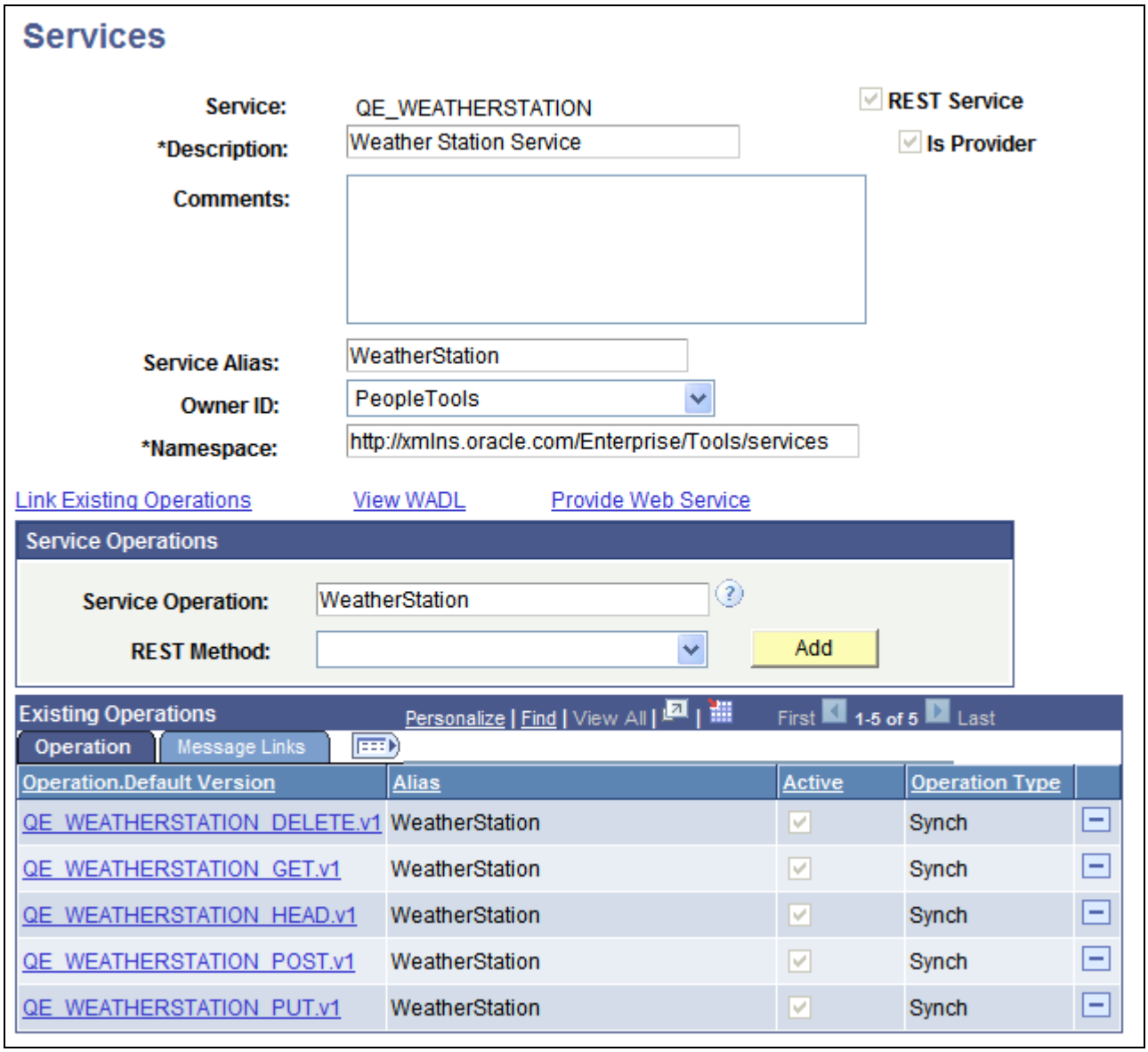


Services-Search page

Note that the Services-Search page features a REST Service check box. To access a REST service in the database you must select the REST Service check box. If you do not select the box, the system searches the database for non-REST services.

After a successful search for a REST service, the system displays the definition in the Services page. The following example shows a REST service definition in Services page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The provider REST service QE_WEATHERSTATION

The top portion of the page features a read-only REST Service box that, when selected, indicates that the service is a REST-based service. The page also features a read-only Provider box that, when selected, indicates the service is a provider service; if the Provider box is clear, the service is a consumer REST service.

To access a REST service definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Services.

   The Services-Search page appears.

2. Select the REST Service box.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Use one of the following methods to search for a definition:

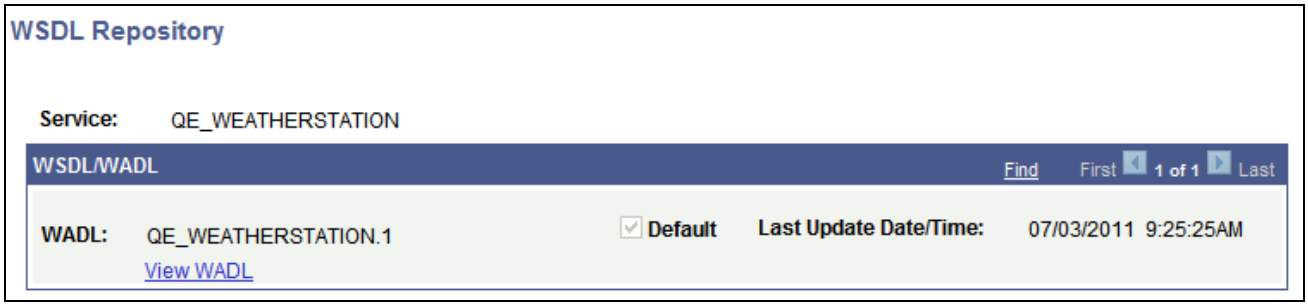
    - To search for a specific service, in the Service field enter the name of the service and click the Search button.
    - To view and select from all REST services in the database, click Search button, and in the results grid select the service to access.

    The definition for the selected service appears in the Services page.

Viewing WADL Documents for REST Services

Click the View WADL link to display the WSDL Repository page (IB_SERVICEDEFN_SEC) and access any generated WADL documents for the service.

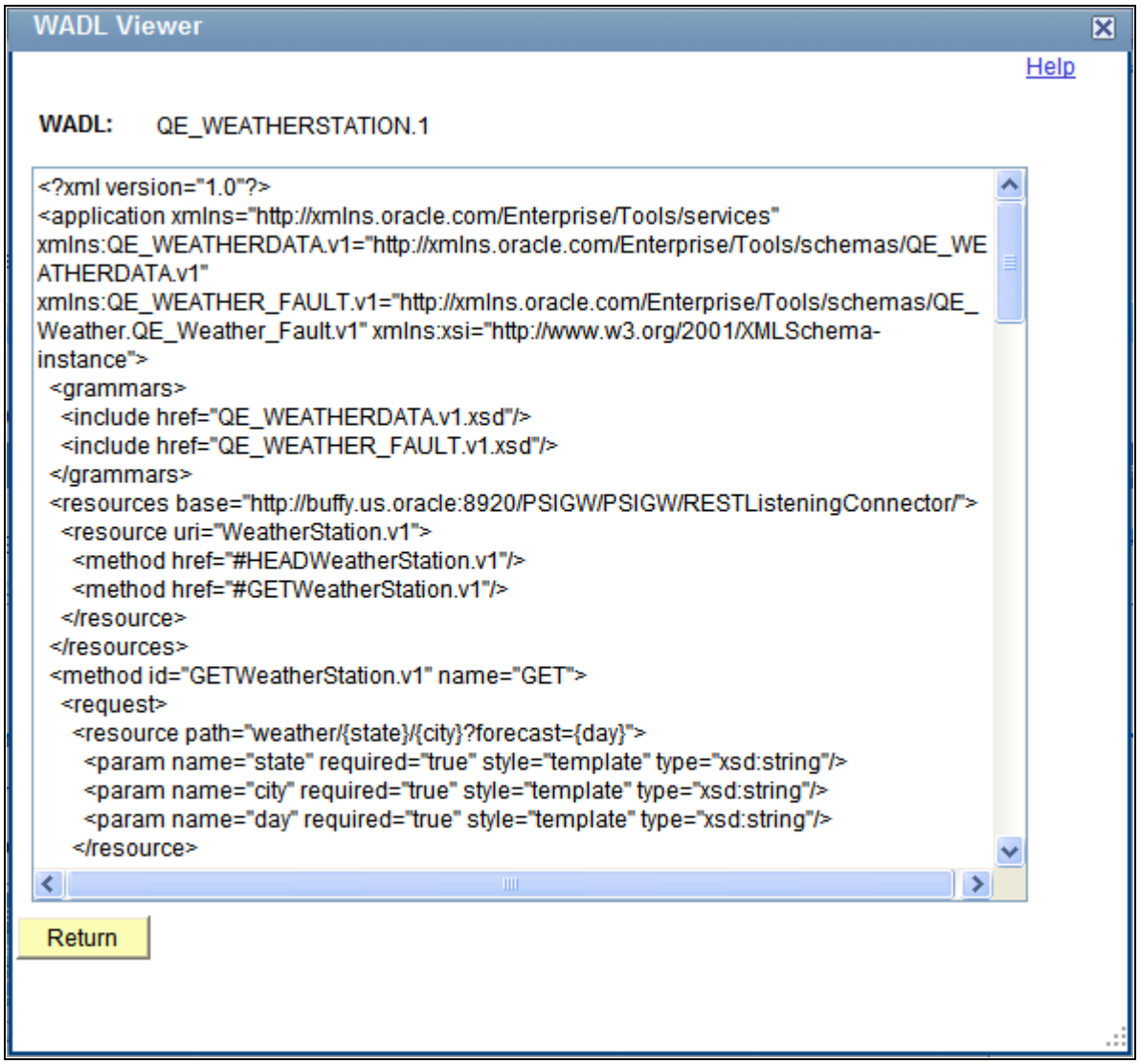
The following example shows the WSDL Repository page for the QE_WEATHERSTATION service.



Use the WSDL Repository page to access WADL documents for a REST service.

When you click the View WADL link on the WSDL Repository page, the WADL Viewer page (IB_SERVICEWSDL_SEC) appears and you can view the WADL document for the service. The following example shows generated WADL for the QE_WEATHERSTATION service appearing in the WADL Viewer page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Viewing a WADL document for a REST service in the WADL Viewer page.

To view a WADL document for a REST service:

1. Access the service definition. (PeopleTools, Integration Broker, Integration Setup, Services.)
2. Click the View WADL link.

   The WSDL Repository page appears.
3. Click the View WADL link.

   The document appears in the WADL Viewer page.
4. Click the Return button to return to the WSDL Repository page.
5. Click the Return button again to return to the Services page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing REST Service Operation Information

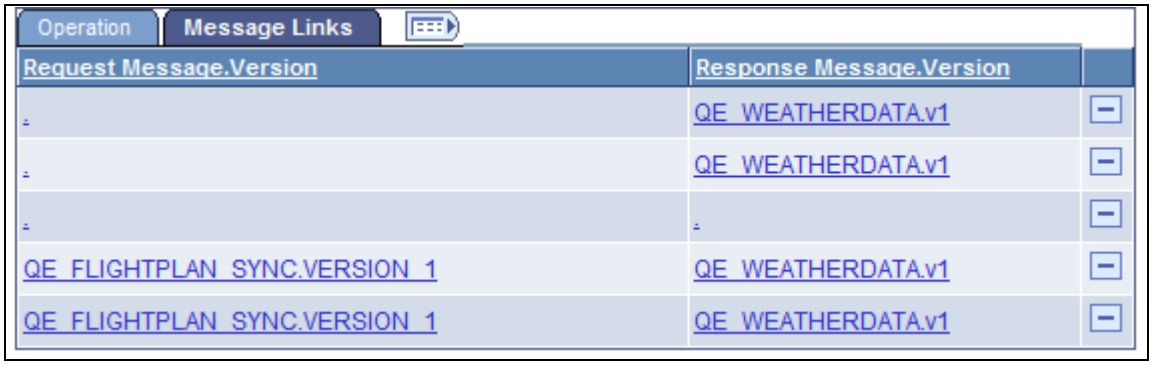
The Existing Service Operations section of the Services page contains an Operation tab that displays service operations and service operation versions that are associated with the service. It also displays descriptions of the service operations, the type of operation, and whether the service operation is active. When you click the name of a service operation, the operation opens on the Service Operations page, where you can view and modify service operations information, work with the service operation handlers, routing definitions, and do much more.

See Also

Chapter 12, "Managing REST Service Operations," page 287

View Messages Defined for REST Service Operations

The Existing Service Operations section of the Services page contains a Messages Links tab that displays the request and response messages defined for each REST service operation in the service. The following example shows the Message Links tab displaying request and response messages for the service operations that are associated with the QE_WEATHERSTATION service:



Request and response messages of service operations associated with the QE_WEATHERSTATION service.

Click the request or response message name to open the message in the Message Definitions page, where you can view and modify message definition information, message schema information, and more.

See Also

Chapter 6, "Managing Messages," page 75

Adding REST Service Definitions

This section discusses how to add REST service definitions to the PeopleSoft system.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

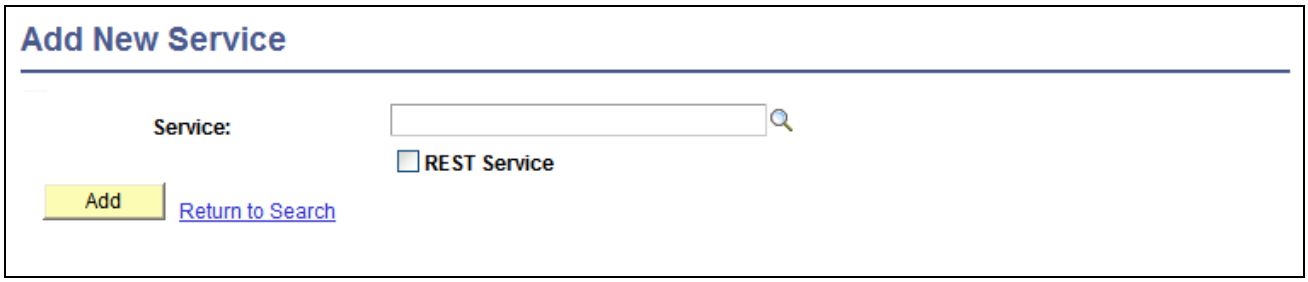
Understanding Naming Services and Service Aliases for REST Service Definitions

REST service names and REST service alias names can be exposed as REST services. As a result, the following restrictions apply when you name a REST service or a REST service alias. REST service names and REST service alias names:

- Cannot begin with *XML, xml, or Xml.*
- Cannot begin with a digit.
- Cannot begin with a character.

Adding REST Service Definitions

To add a REST service definition to the system, use the Add a New Value link on the Services search page. To access this page, select PeopleTools, Integration Broker, Integration Setup, Services, then select the Add a New Value link.



Use the Add New Service page to add a REST service to the system.

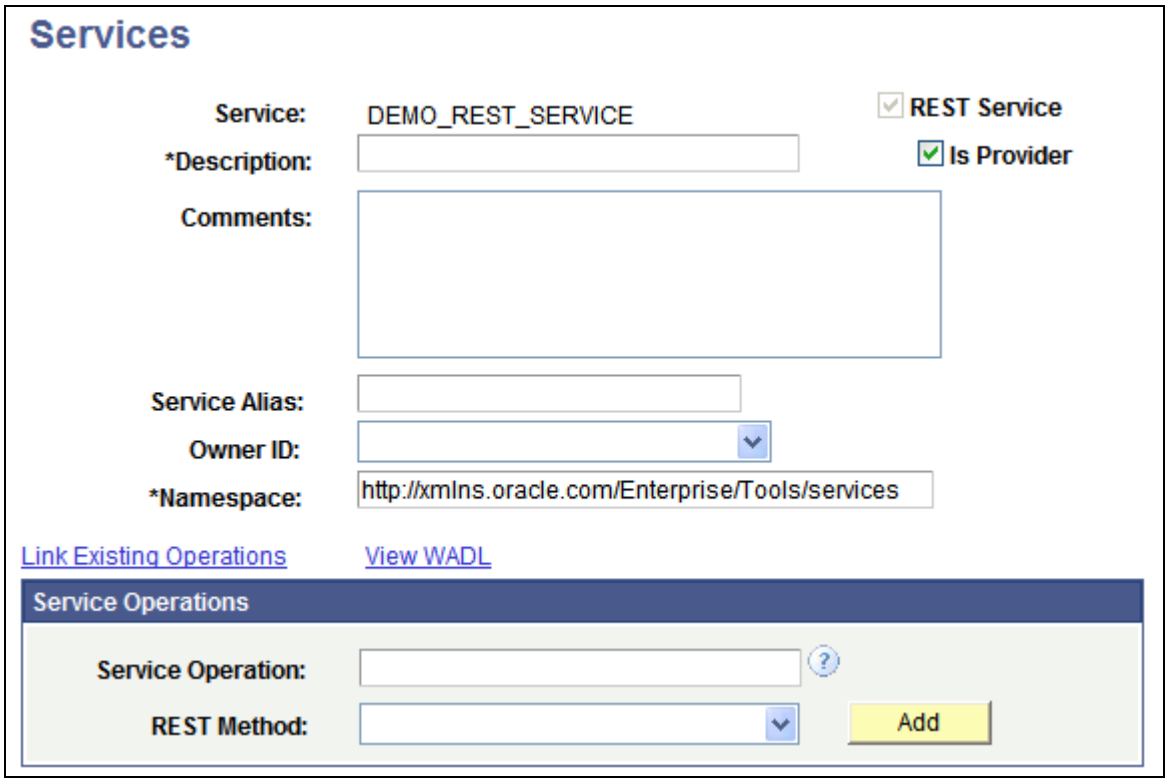
Note. Before you can add a service, you must configure PeopleSoft Integration Broker to handle services using the Service Configuration page.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services.

Important! To create a REST service you must select the REST Service option when you add the service.

After you add a service definition to the system, the Services page (IB_SERVICEDEFN) appears that you use to configure the service definition. The following example shows the Services page after a REST-based service called DEMO_REST_SERVICE is added to the system. Note that the REST Service box at the top of the page is checked, signifying that the service is REST-based.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The Service page after a REST-based service is added to the system.

To add and configure a REST service definition:

1. Access the Services page (PeopleTools, Integration Broker, Integration Setup, Services).
2. Click the Add a New Value link.
3. In the Service field, enter a name for the service.

   Restrictions for naming services are discussed elsewhere in this section.

   See Chapter 11, "Managing REST Services," Understanding Naming Services and Service Aliases for REST Service Definitions, page 279.
4. Check the Rest Service box.
5. Click the Add button.

   The Services page appears.
6. In the Description field, enter a description for the service.
7. (Optional.) Check the Is Provider box to define the service as a provider service.

   By default, when you add a REST-based service to the system it is a provider service.
8. (Optional.) In the Comments field, enter comments about the service or the service.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

9. (Optional.) In the Service Alias field, enter an alias name for the service.

   Restrictions for naming service aliases are discussed elsewhere in this section.

   See Chapter 11, "Managing REST Services," Understanding Naming Services and Service Aliases for REST Service Definitions, page 279.

10. (Optional.) From the Object Owner ID drop-down list box, select the owner of the service.

11. In the Namespace field, enter a namespace URI for the service.

    The default value is the namespace that is declared in the Service Namespace field on the Service Configuration page.

12. Click the Save button.

Adding Service Operations to REST Service Definitions

This section discusses how to:

- Add existing service operations to REST service definitions.
- Define new service operations for REST service definitions.

Understanding Adding Service Operations to REST Service Definitions

You can add a REST service operation that already exists in the database to a REST service definition, or you can add a new REST service operation to the database.

Adding Existing Service Operations to REST Service Definitions

The process for adding existing REST service operations to REST services is identical to the process for adding non-REST services to non-REST service operations.

Important! You can add only those service operations defined as REST service operations to REST service definitions; you cannot add non-REST service operations to REST services and you cannot add REST service operations to non-REST services.

See Chapter 9, "Managing Services," Adding Existing Service Operations to Service Definitions, page 232.

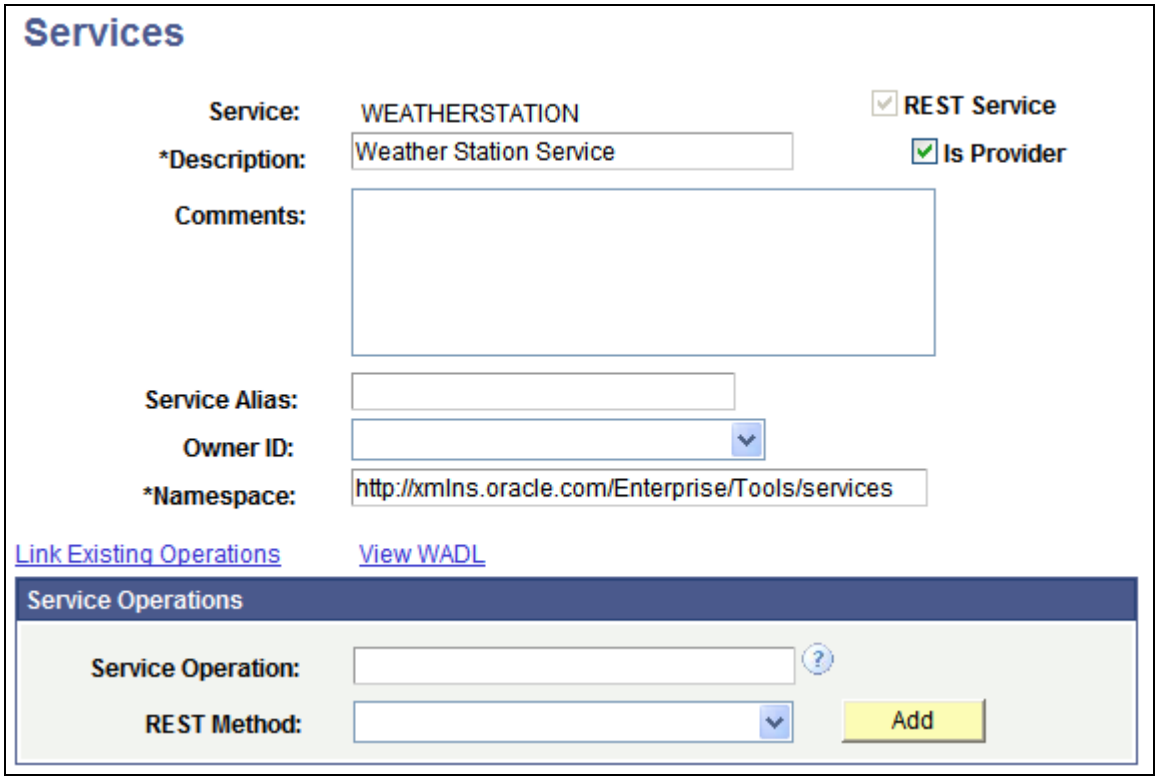
Adding New Service Operation Definitions for REST Services

The only way to add a new REST service operation to the database is from a REST service, using the Add button in the Service Operations section of the Services page (IB_SERVICEDEFN).

The type of service, provider or consumer, determines the type of service operation that the system adds. If you are working with a provider service and add a service operation, a provider service operation is added to the service; if you are working with a consumer service and add a service operation, the system adds a consumer service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following example shows a REST provider service definition:



Use the Service Operations section on the Services page to add REST service operation definitions to the system

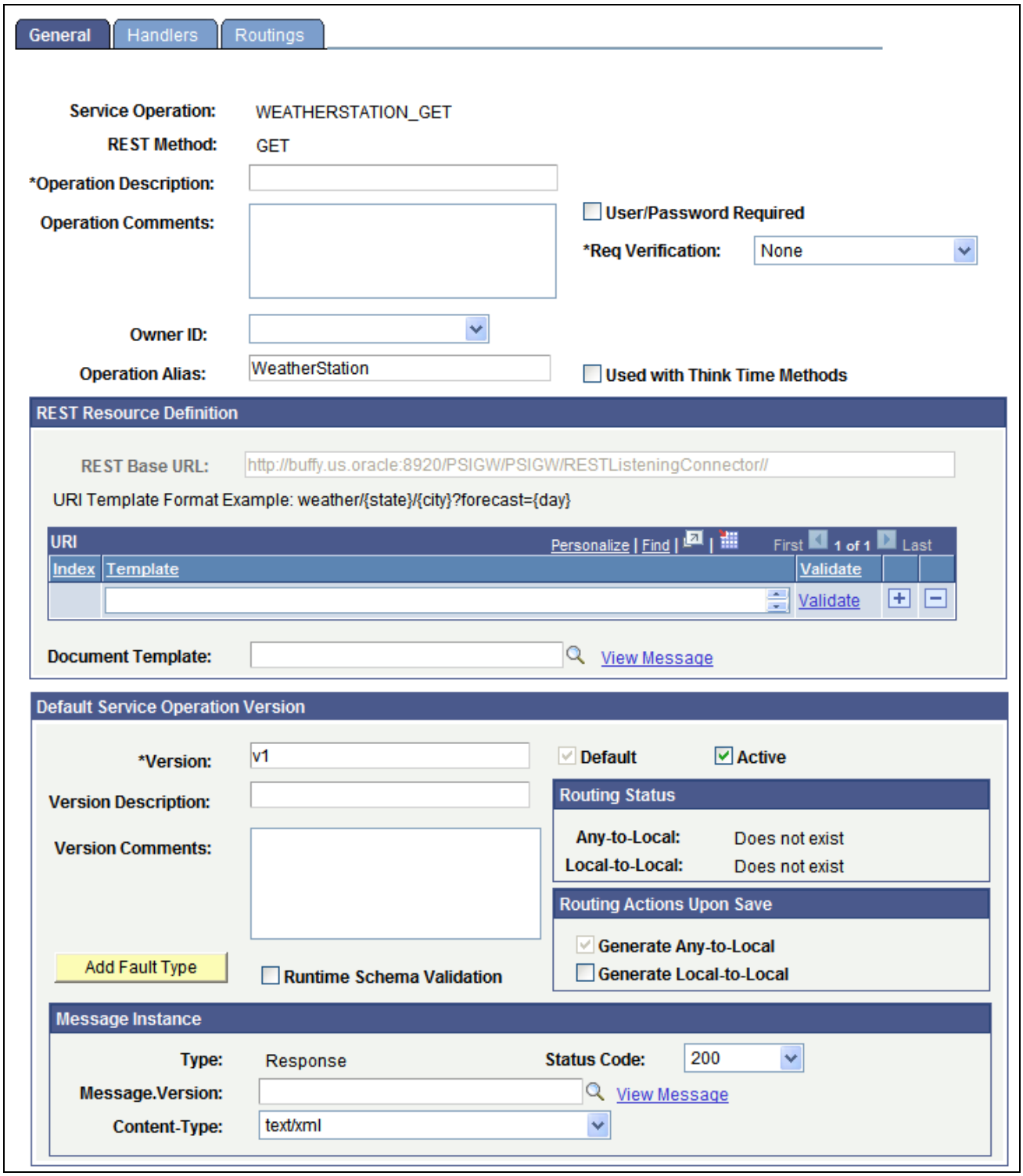
The Services page features a Service Operations section where you define the service operation name and the REST method. The system automatically appends the service operation name with the name of the selected REST method. This functionality allows you to reuse the service operation name using other method types. The system allows you to enter a service operation name in mixed-cased. The service operation name you enter becomes the default service operation alias.

So for example, if you enter a service operation name of *WeatherStation* and select the GET method from the REST Method drop-down list, the system names the operation *WEATHERSTATION_GET* and the service operation alias is *WeatherStation.*

If a service is defined as a provider service, all service operations you create from the service definition are provider services operations. If a service is defined as a consumer service, all service operations you create from the service definition are consumer service operation.

After you provide a service operation name and a REST method and click the Add button, the new service operation appears in the Service Operations component and you can proceed to configure the operation. The following example shows a new REST service operation called *WEATHERSTATION_GET* appearing in the Service Operations page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



A REST provider service operation called *WEATHERSTATION_GET*

To add a new service operation for a REST-based service:

1. Access the Services page (PeopleTools, Integration Broker, Integration Setup, Services).

   The Services page appears.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. In the Service Operations section, locate the Service Operation field and enter a name for the new service operation.
3. From the REST Method drop-down list, select the HTTP method for manipulating the resources defined in the service operation. The valid values are:
   - *Delete.*
   - *Get.*
   - *Head.*
   - *Post.*
   - *Put.*

   REST methods are defined elsewhere in this PeopleBook.

   See Chapter 12, "Managing REST Service Operations," REST Methods, page 288.
4. Click the Add button.

The Service Operations-Definitions page appears. Use the page to configure the service operation.

See Also

Chapter 12, "Managing REST Service Operations," Configuring REST Service Operation Definitions, page 297

Restricting and Enabling Write Access to REST Service Definitions

You can restrict write access to a REST service so that sensitive fields of the service definition and of associated service operations appear in read-only mode. Use the Restricted Service page (IB_SVCSETUP3) in the Service Configuration component (IB_SVCSETUP) to restrict a service. Use the process for restricting and enabling write access to non-REST service definition for restricting and enabling write access to REST service definitions.

See Chapter 9, "Managing Services," Restricting and Enabling Write Access to Service Definitions, page 240.

Renaming and Deleting REST Service Definitions

You can rename and delete services using the Services tab in the Service Administration component (IB_HOME_PAGE). Use the process for renaming and deleting non-REST service definitions for renaming and deleting REST service definitions.

See Chapter 9, "Managing Services," Renaming and Deleting Service Definitions, page 242.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Activating and Deactivating REST Service Definitions in Bulk

You can activate and deactivate services in bulk using the Service Activate/Deactivate page (IB_HOME_PAGE9) in the Service Administration component. Use the process for activating and deactivating non-REST services in bulk for activating and deactivating REST services in bulk.

See Chapter 9, "Managing Services," Activating and Deactivating Service Definitions in Bulk, page 244.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 12

Managing REST Service Operations

This chapter discusses how to:

- Access and view REST service operations.
- Configure REST service operation definitions.
- Manage provider REST service operations.
- Manage consumer REST service operations.
- Manage REST service operation versions.
- Set compression for REST service operations.
- Rename and delete REST service operations.

Note. Before you can provide or consume services in a PeopleSoft system, you must configure the system for handling services.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

Understanding Managing REST Service Operations

This section discusses:

- REST service operations.
- REST service operation types.
- REST methods.
- REST resource definitions.
- REST messages.
- Naming conventions for REST service operation metadata.
- REST service operation aliases.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- REST service operation versions.
- Monitoring service operations.

REST Service Operations

A *REST service operation* definition consists of general information about an operation, such as its name, description, request and response messages, and so on. A REST service operation definition also contains a handler, which contains and runs the programming logic for providing or consuming the service operation, manipulating message content, and so on.

Like a non-REST service operations, a REST service operation contain a routing, which determines the direction, inbound or outbound, of the service operation.

For a provider REST service operation the system creates an any-to-local routing definition when you save the service operation. You have the option to generate a local-to-local routing definition. For consumer REST service operations the system creates an outbound local-toWADL routing definition when you save the service operation. You cannot add routing definitions to provider or consumer REST service operations.

REST Service Operations Types

All REST service operations are synchronous service operations.

A synchronous service operations is processed in real time. Processing on the sending system does not continue until it receives a response from the receiving system.

REST Methods

REST methods determine the HTTP method for manipulating the resources defined in the service operation. The valid values are listed and described in the following table and can all be used for provider and consumer service operations:

DELETE	Delete an existing resource.
GET	Retrieve a representation of a resource.
HEAD	Identical to a GET except that no message body is returned in the response.
POST	Create a new resource to an existing URL.
PUT	Create a new resource to a new URL or modify an existing resource to an existing URL.

REST Resource Definitions

REST service operation definitions contain a REST Resource Definition section where you define the REST base URL, resource URI templates, and document template. The system generates a REST index for each URI template defined. This section describes these REST resource definition components.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

REST Base URL

The REST base URL is the common part of the URL to invoke the provider or consumer web service.

For a REST service provider this value is always the REST target location entry defined on the Service Configuration page for provider service operations and the external alias from the any-to-local routing definition created by the system. When the Provide Web Service wizard is used to create the WADL, the option to select a secure REST target location can be selected. For provider service operations this URL is simply informational to show what would be used when creating the complete URL(s).

The REST base URL is constructed as part of the initialization/retrieval of the service operation and uses the following format:

```
<REST target location>/< external alias from any-to-local routing>/
```

For a REST consumer the common part of the URL can be found in the provider's WADL document.

URI Template

Resources are constructed using URI templates. URI templates use a simple syntax where braces denote variables to be replaced when the templates are converted to actual URLs.

Note. PeopleSoft supports direct variable substitution.

The following example shows a sample URI template syntax:

```
weather/{state}/{city}?forecast={day}
```

The denoted variables identified with the braces ({}) are replaced with the actual value as defined in a document template.

The system validates the string entered for the URI template to ensure that there is no equivalent or ambiguously– defined template currently entered on the grid. Moreover the values used within the braces ({}) must be defined on the document template.

REST Document Template

A document template is a Document type message that you assign to the service operation. The document message has primitive elements defined with names used for value replacement within the URI template. This document can only contain primitive elements. The document cannot contain any imports (compounds). The advantage of using documents is that they cannot be inadvertently changed, as is not the case with defined record and/or fields.

Note. In the Document definition, the variables defined with the ({}) must be the exact name of the element name defined on the Document page, not the Tag Name defined on the XML page.

A document template is a URI on the wire with values in it.

URI Index

The system assigns an integer index value to each URI template you create for a REST service operation upon save.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

You will later use these index numbers in PeopleCode to call and manipulate the associate URI template.

URI Template Validation

When you construct a URI template, you can verify the URL produced.

The REST Resource Definition URI grid features a Validate link that provides access to the Validate URI page where you can populate the URI with values from the document template and then test the link.

REST Messages

This section provides information about request and response messages used in REST service operations.

Request and Response Message Combinations

The REST method you select for a REST service operation determines the request and response message combination that you can use for the operation. The following table lists the allowable request and response message combinations for each REST method:

REST Method	*Request Message*	*Response Message*
Delete	Optional.	Optional.
Get	Optional.	Optional.
Head	No.	Optional.
Post	No.	Yes.
Put	No.	No.

MIME Content-Types

When you specify a request or response message for a REST service operation, must also specify the MIME content-type of the message. PeopleSoft supports the following content types for request and response messages used for REST service operations:

- application/json
- application/xml
- text/xml
- text/plain
- text/html

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Naming Conventions for REST Service Operation Metadata

The same naming convention restrictions that apply to non-REST service operations apply to REST service operations.

See Chapter 10, "Managing Service Operations," Naming Conventions for Service Operation Metadata, page 248.

REST Service Operation Aliases

A *service operation alias* or *operation alias* is the service operation name that displays for the service operation when WADL is provided.

REST service operation aliases may be mixed case.

Service operation alias names for REST service operations can be the same across types. For example, you can have the same alias, for example *MY_ALIAS,* for Delete, Get, Put, Post and Head REST service operations.

REST Service Operation Versions

The same information about non-REST service operation versions applies to REST service operation versions.

See Chapter 10, "Managing Service Operations," Service Operation Versions, page 249.

WADL Node

PeopleTools delivers a WADL node that is the default node for consumer REST services.

Monitoring REST Service Operations

You can use the same tools to monitor REST service operations as those you use to monitor non-REST service operations. These tools include:

- The Service Operations Monitor.
- The Integration Network Monitor.
- The Integration Network Transactional Tracker.
- Performance monitoring tools.

The Managing Service Operations chapter in this PeopleBook provides a brief summary of each of these tools.

See Chapter 10, "Managing Service Operations," Monitoring Service Operations, page 249.

You can also find complete documentation for these tools elsewhere in PeopleBooks.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See *PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook*

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using the Integration Network," Using the Integration Network Monitor.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using the Integration Network," Using the Integration Network Transactional Tracker.

See *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on Oracle," Monitoring PeopleSoft MODULE and ACTION Information; *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on DB2 UDB for z/OS," Associating PeopleSoft Users, Modules, and Actions with DB2 UDB Threads and *PeopleTools 8.52: Data Management*, "Administering PeopleSoft Databases on DB2 UDB for Linux, UNIX, and Windows," Monitoring Module and Action Information.

Accessing and Viewing REST Service Operation Definitions

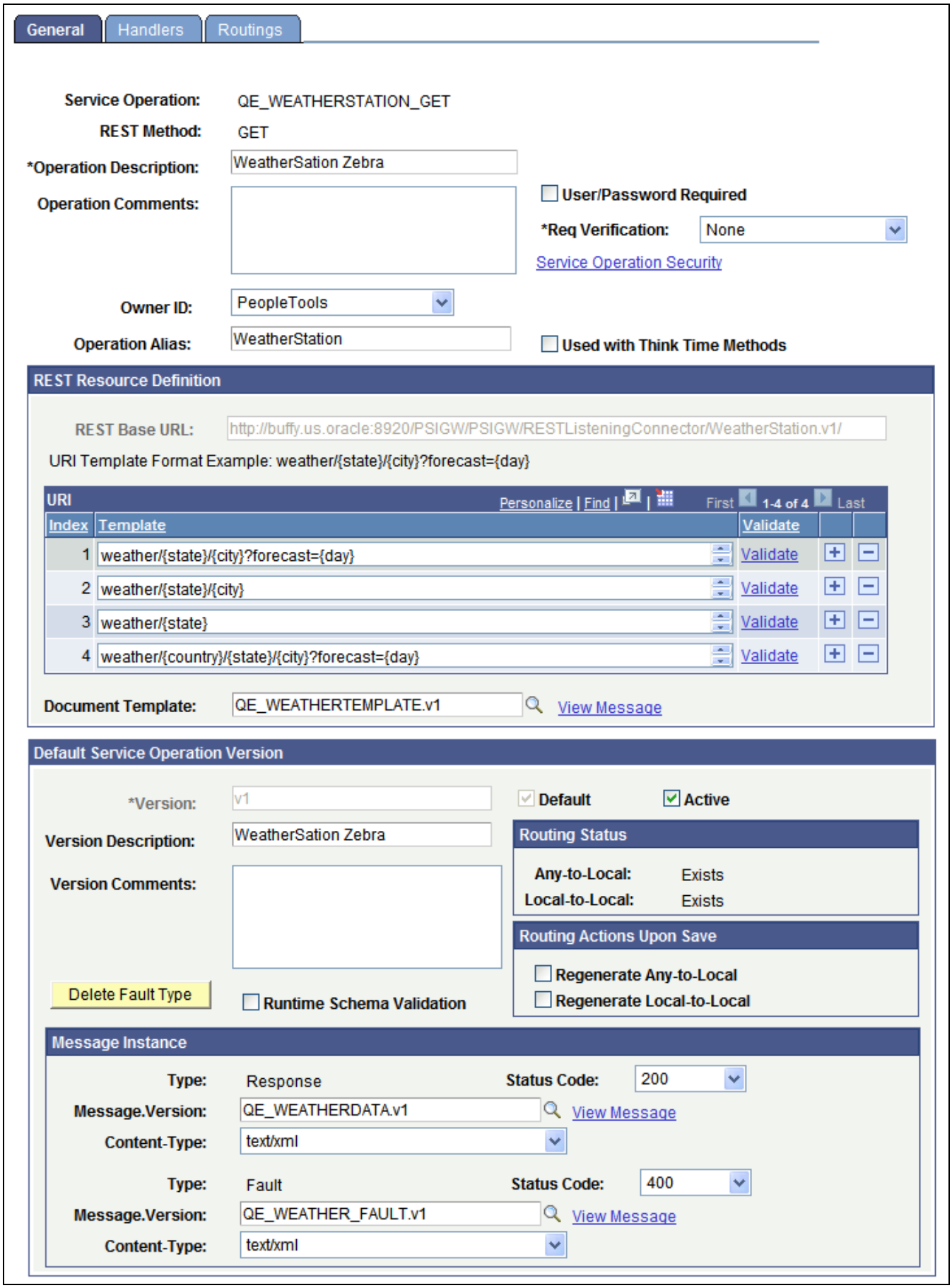
This section discusses how to:

- Access REST service operation definitions.
- View REST service operation definitions.

Accessing REST Service Operation Definitions

Use the Service Operations component (IB_SERVICE) to access and view REST service operation definitions. The following example shows the General tab of the Service Operations component for the QE_WEATHERSTATION provider service operation:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

QE_WEATHERSTATION_GET REST service operation appearing in the Service Operations – General tab.

To access a REST service operation:

1. Select PeopleTools, Integration Broker, Integration Setup, Service Operations.

   The Service Operations– Search page appears.

2. Search for a service operation definition to view.

   You can search for an operation in one of two ways:

   - Click the Search button to display all service operation definitions in the system.
   - Enter search criteria in one or more of the following fields, and then click the Search button:

   Service Enter the service name that contains the service operation that you want to view, or click the Lookup button to search for and select a service name.

   Service Operation Enter the name of the service operation to view, or click the Lookup button to search for and select an operation.

   Note. The Operation Type field that appears in the search box is not applicable to searching for REST service operations. The field is used only to search for non-REST service operations.

   The results of the search appear in the Service Operations results grid.

3. In the Service Operations results grid, click the name of the service operation to view.

   The Service Operations – General tab appears with data for the service operation that you selected.

Viewing REST Service Operation Definitions

The Service Operations component includes three pages:

General Tab Features general-service and default-service operation information.

This page also features REST resource definition information if the service operation you are viewing is a REST service operation.

Handlers Tab Provides summary information about handlers that have been added to an operation. Service operation handlers contain the programming logic for sending and receiving service operations and their contained messages, and for manipulating content.

Note that handlers are not used for REST consumer-based service operations and the Handler tab does not appear when you view or work with a consumer REST service operation.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Routings Tab Provides summary information about service operations routings. Routing definition determine the direction — inbound, outbound — of service operations.

Viewing General Service Operation Information

When you access a service operation (PeopleTools, Integration Broker, Integration Setup, Service Operations, the Service Operations - General page (IB_SERVICE) appears.

The top portion of this page contains basic information about a service operation, including its name, description, and so on. The Service Operation Security link opens the permission list for the service. Note that the Service Operation Security link appears only after a service operation definition is saved.

The page features a REST Resource Definition section that displays the REST base URL, resources defined for the service operation, and the name of the document template associated with the service operation.

The Default Service Operation Version section displays service-operation version information and whether runtime schema validation is enabled.

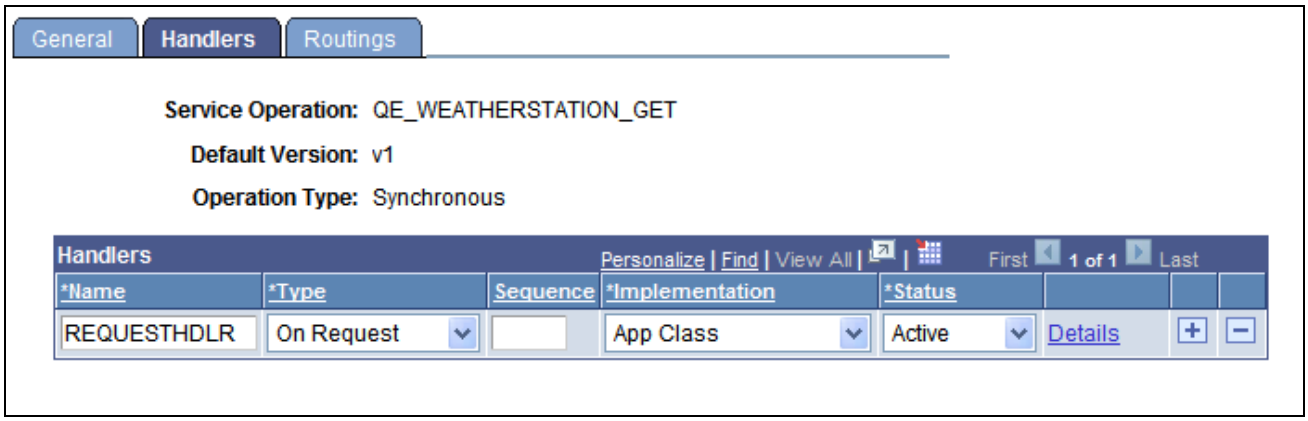
The Routing Status group box shows if any-to-local or local-to-local routing definitions have been generated for the service operation. Click the Routings tab to view detailed information about routing definitions that exist for a service operation.

The Routing Actions Upon Save group box shows the possible routings that the system can generate when the service operation definition is saved.

The Message Information section displays the request message, response message information, and fault message for the service operation. The View Message links in this section open the displayed message on the Message Definition page, where you can view additional information about the message.

Viewing Handler Information

To view service operation handler information, click the Handler tab. Doing so displays the Service Operations - Handlers page (IB_SERVICEHDLR) which lists summary information about handlers that have been added to an operation.



Service Operations – Handlers page

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

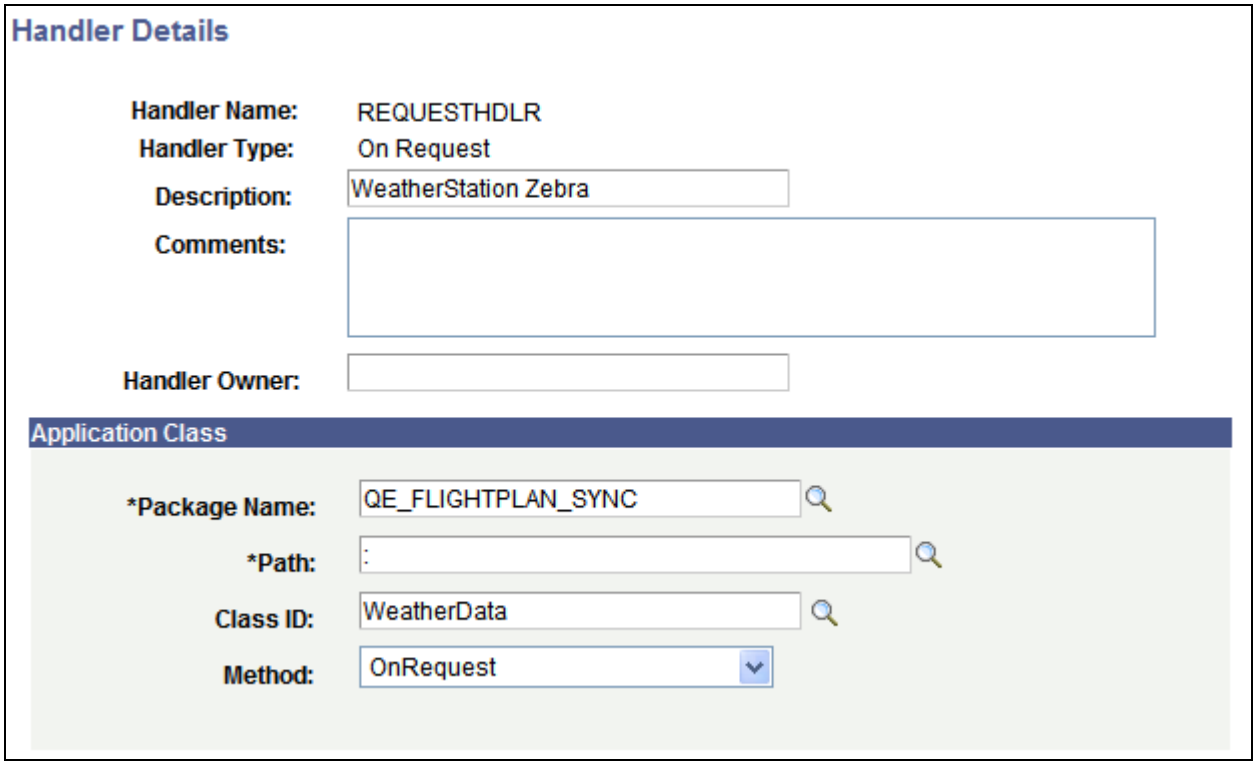
Each REST-based provider service operation has a single handler; REST-based consumer service operations do not have handlers.

Note. There are no handlers for REST-based consumer service operations and the Handler tab does not appear when you view and work with consumer REST service operations.

The only Handler Type available for provider REST services is the OnRequest handler. When working with REST provider service operations you can implement the OnRequest handler as an application class only. The application class is implemented as any other OnRequest integration broker event.

The summary information includes the handler name, the handler type, and the implementation method for the handler. The status of the handler, active or inactive, also appears.

Click the Details link to open the Handler Details page (IB_SERVICEHDLR_SEC) for the handler. The following example shows the Handler Details page:



Handler Details page

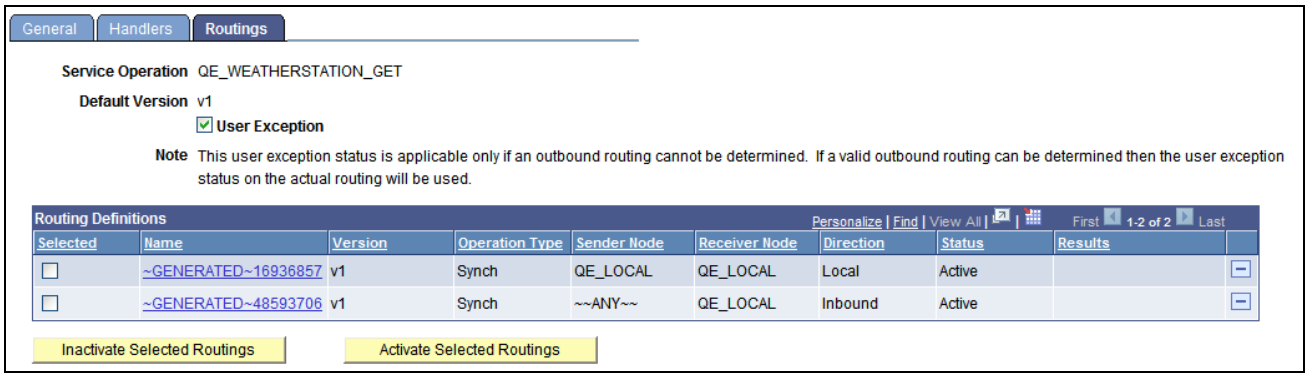
The Handler Details page shows additional information about the handler, including the owner and application class details.

You can also use this page to specify the handler details.

Viewing Routing Information

Click the Routing tab to open the Service Operations - Routings page (IB_SERVICERTNGS) to view a summary of routing definitions for an operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Service Operations–Routings page

The Routings Definition grid on the page lists summary information for routings that are defined for a service operation. Summary information that is displayed includes the routing definition name, service operation version, routing type, sending node, receiving node, direction of the routing and the routing status.

Click a routing definition name to open the routing in the Routing Definitions component, where you can view additional information about the routing.

When you create a provider REST service operation the system automatically creates an any-to-local routing. You can also generate local-to-local routings for a provider REST service operation.

When you create a consumer REST service operation the system automatically creates a local-to-WADL routing definition. Local-to-WADL is the only routing type you can use with consumer REST service operations.

See Chapter 17, "Managing Service Operation Routing Definitions," Viewing Routing Definitions in Graphical Format, page 404.

You can also use this page to activate or inactivate routings for an operation.

See Chapter 10, "Managing Service Operations," Adding Routing Definitions, page 259 and Chapter 10, "Managing Service Operations," Activating and Inactivating Routing Definitions, page 260.

Configuring REST Service Operation Definitions

This section discusses how to:

- Specify general service operation information.
- Manage REST resource definitions.
- Define REST service operation version information.

Specifying General Service Operation Information

To specify general service operation information:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Service Operations-General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Operation Description field, enter a description for the operation.
3. (Optional.) Check the User ID/ Password Required check box to require a user ID and password for inbound service operations.

   See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Managing User Authentication.
4. The Security Verification drop-down list appears when you are configuring a provider service operation, and you may select the level of security with which inbound integrations must be sent. The valid options for REST based provider/inbound service operations are:

   - *None.*
   - *SSL.*
   - *Basic Authentication.*
   - *Basic Authentication and SSL.*

   Validating security on inbound integrations is described elsewhere in PeopleBooks

   > See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Validating Security on Inbound Integrations.
5. (Optional.) In the Operation Comments field, enter comments about the service operation.
6. (Optional.) From the Object Owner ID field, select the owner of the definition.

   The owner ID helps to determine the application team that last made a change to a service definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.
7. (Optional.) In the Operation Alias field, enter an alias name for the service operation.

   When working with REST service operations the service operation alias is automatically populated with the service operation name as you entered it when you created the service operation.

Managing REST Resource Definitions

This section discusses how to:

- Configure REST resource definitions.
- Validate REST URI templates.

Prerequisites for Managing REST Resource Definitions

The REST target URL must be specified on the Service Configuration – Target Locations page. The steps for defining the REST target location are described elsewhere in PeopleBooks.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services."

You must have created a Document type message that contains the values for any variable data you plan to use in the URI templates. For this reason you must have an understanding of PeopleSoft Documents Technology to create and manipulate document type messages.

PeopleTools 8.52: PeopleSoft Documents Technology PeopleBook

Configuring REST Resource Definitions

When you configure a REST resource definition you create one or more URI templates and assign a document template to the definition. Overviews of these topics are provided elsewhere in this chapter.

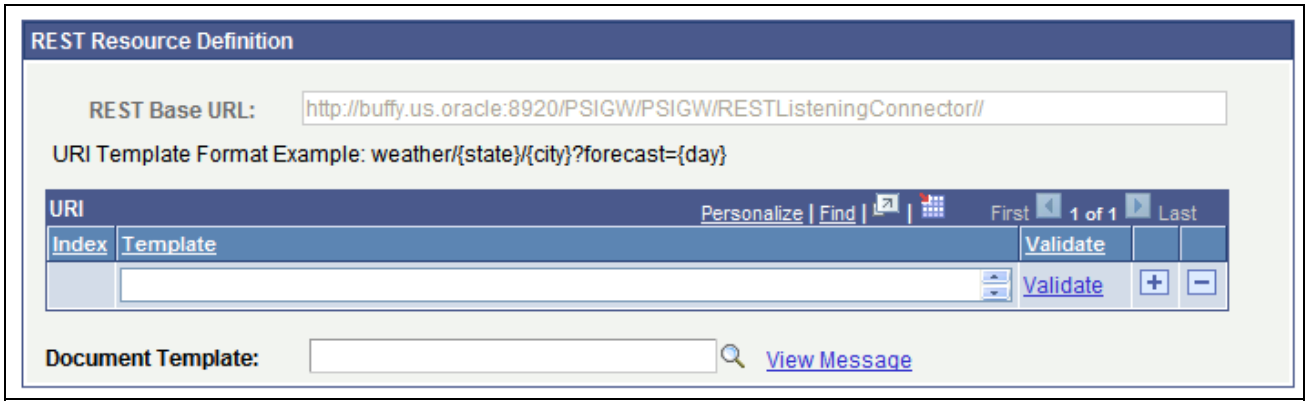
See Chapter 12, "Managing REST Service Operations," Understanding Managing REST Service Operations, page 287.

The system auto-generates an index for each URI template that you add. You use the index the PeopleCode that you write in conjunction with the service operation.

When you configure a REST resource definition the end result is the URL that the service consumer uses to invoke the service.

The information defined in the REST resource definition is included in the WADL document that the service provider furnishes to the service consumer. The service consumer then extracts the information from the WADL document and defines it in the consumer service operation it creates for the service.

Use the REST Resource Definitions section of the Service Operations page to configure REST resource definitions. The following example shows the section:



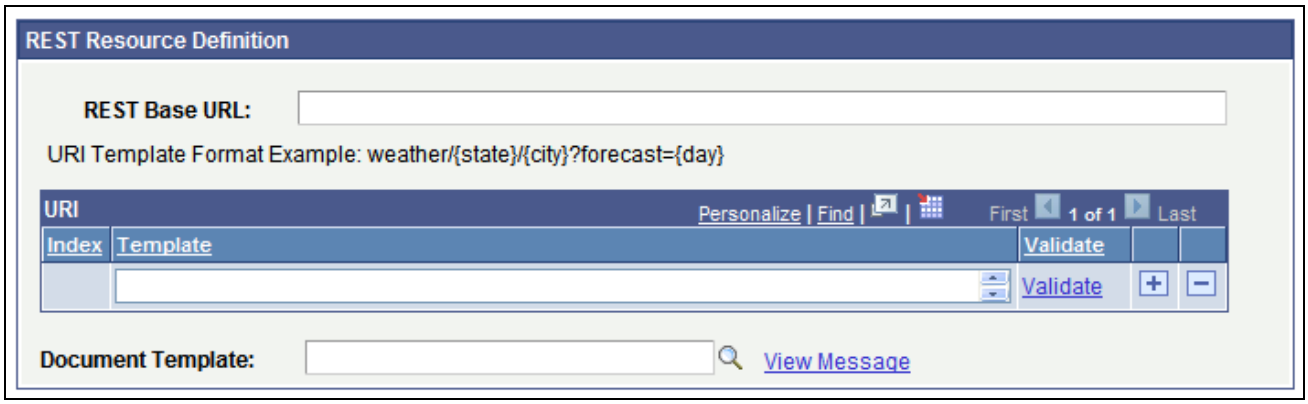
Rest Resource Definition section for a provider service operation.

For provider service operations the REST base URL that appears in the section is read-only. The value that appears is the value configured for the REST target location in the Service Configuration page. Information about REST base URLs and target locations is discussed elsewhere in this PeopleBook.

See Chapter 12, "Managing REST Service Operations," REST Resource Definitions, page 288.

For consumer service operations the REST BASE URL field can be edited and must be populated with the integration provider's base URL for the service as defined in the WADL document furnished by the service provider. The following example shows the REST Resource Definition section of the Service Operations page when configuring a consumer service operation:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



REST Resource Definition section for a consumer service operation.

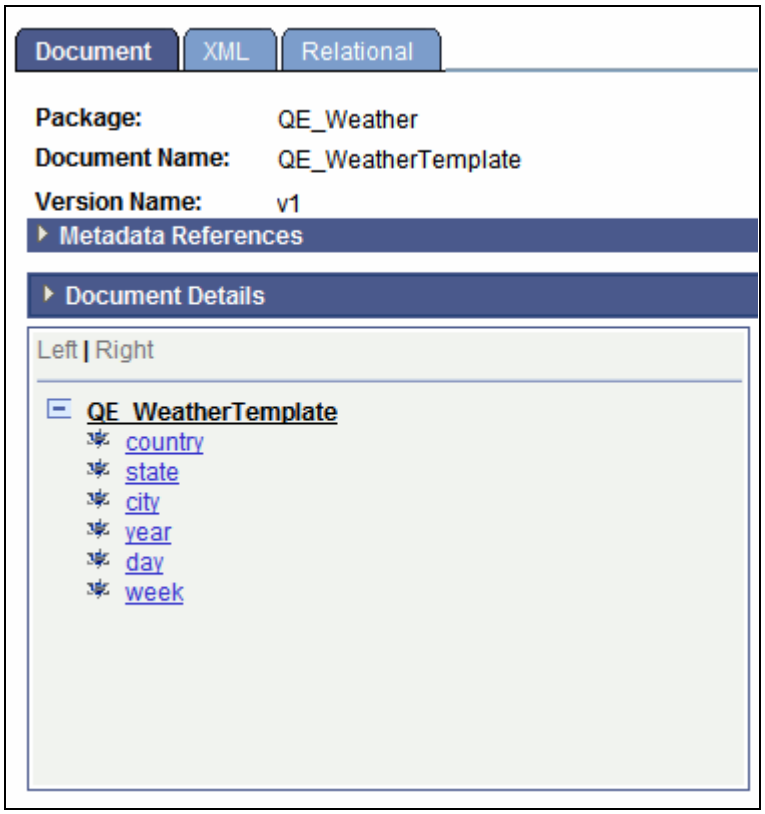
The page allows you to add one or more template URI strings to define one or more resources for the definition. The following example shows a REST Resource Definition configured for a provider service:

A REST Resource Definition configured with four URI templates and a document template.

In the example shown, the QE_WEATHERTEMPLATE.v1 message contains the primitive elements to supply values for country, state, city, and day, as indicated by the following *{country}*, *{state}*, *{city}*, and *{day}* variables defined in URI templates.

The following example shows the content of the QE_WEATHERTEMPLATE.VI message where the values for the variables in the URI template are defined:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



QE_WEATHERTEMPLATE.V1 document message.

To configure a REST resource definition:

Note. When configuring consumer services the information necessary to configure the REST resource definition should be furnished to you in a WADL document by the integration partner providing the service.

---

1. Access the Service Operations - General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. Locate the REST Resource Definition section of the page.
3. Enter the REST base URL

   For provider service operations, this is a read-only field. The value populated in the field is the REST target location defined in the Service Configuration component.
4. In the Template field, enter the syntax for the URI template.

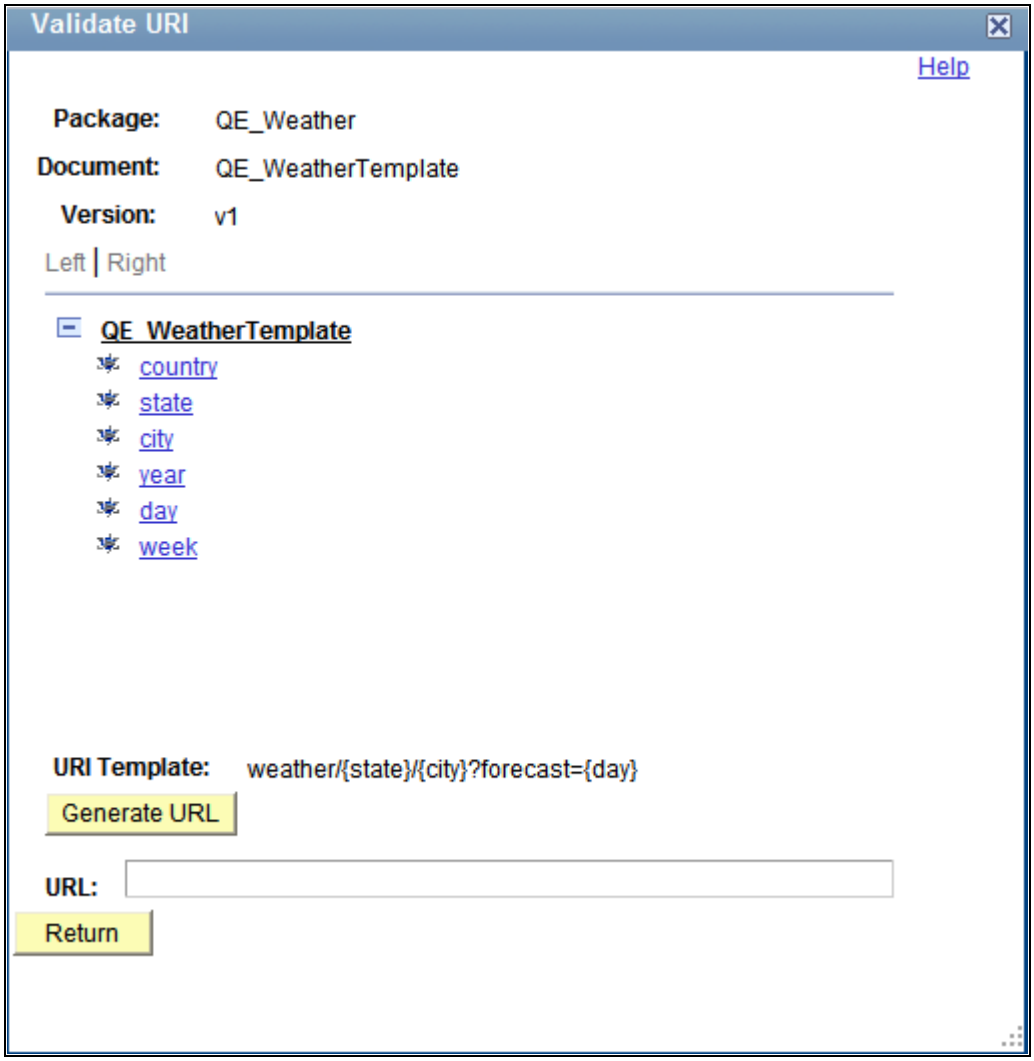
   Upon save the system auto-generates an index for the URI template.
5. To add additional URI templates, in the URI grid click the plus (+) button to add a row.
6. In the Document Template field, enter the name of the document message to associate to the definition, or click the Lookup button to search the database.

   After you select a message, you can click the View Message link to view it.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Validating URI Templates

After you construct a URI template you can verify the URL produced by using the Validate URI page (IB_SVCVERIFY_SEC). To access the page, click the Validate link next to the URI template for the URL to validate. The following example shows the Validate URI page:



Use the Validate URI page to view and validate URLs create from URI templates.

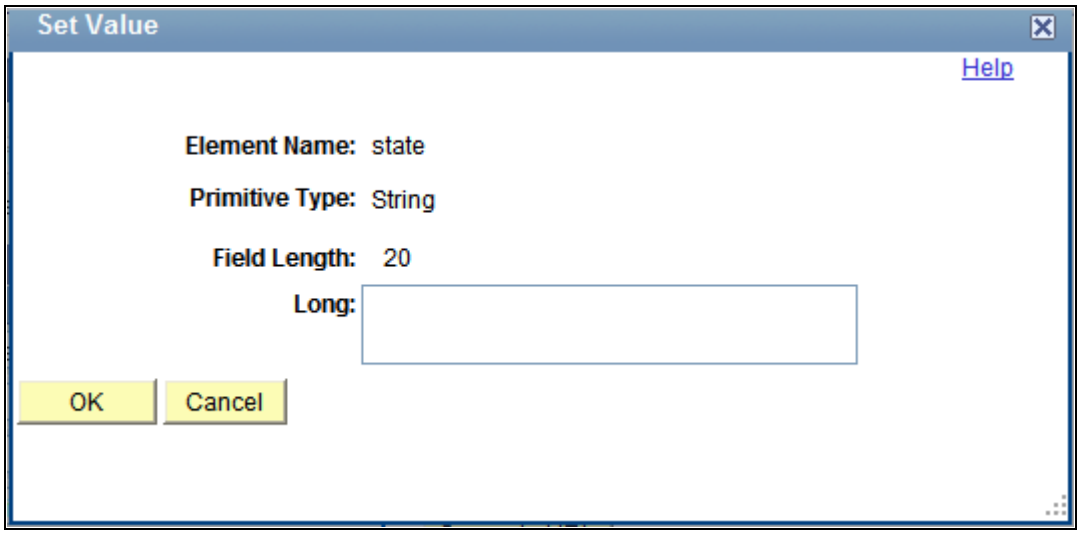
The Validate URI page displays the document information and primitive elements defined in the document template specified for the REST resource definition.

The name of the URI template that is selected for validation appears in the URI Template field. In the previous example, the following URI template appears in the URI Template field and is selected for validation:

```
weather/{state}/{city}?forecast={day}
```

To validate the URI template, you click each primitive element in the document tree that corresponds to each variable in the URI template, and enter a test value for use in validation. When you click a primitive element in the document tree, the Set Value page (IB_LSTESTER_SEC) appears shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



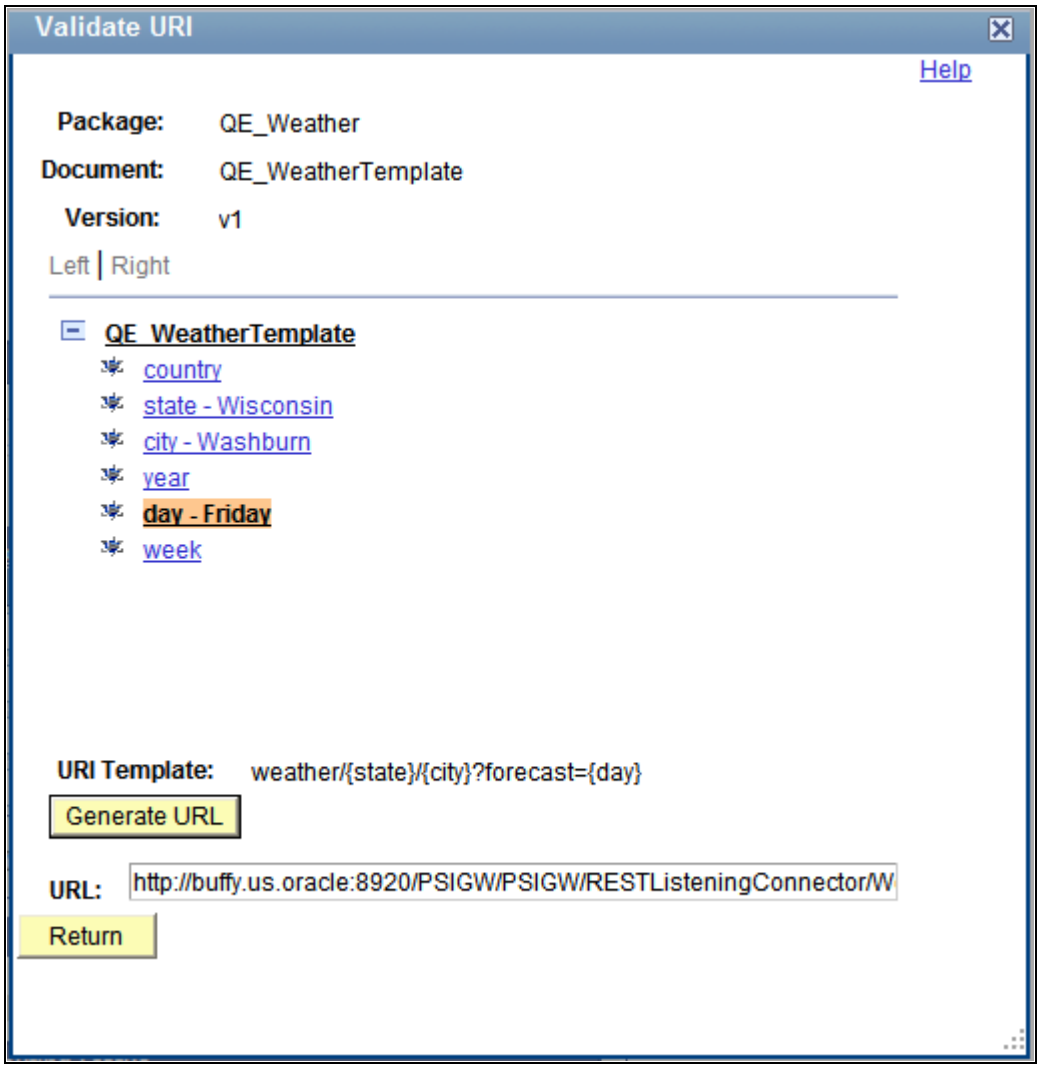
Use the Set Value page to enter a test value to validate for a URI template variable.

The Set Value page provides information about the primitive element selected to assist you in providing a useful value to test. The information that appears varies, depending on the primitive type for which you are providing a value. After you set a test value for an element, the test value appears next to the element name on the document tree.

When you have defined test values for all variables in the URI template, click the Generate URL button to generate a URL based on the test values you entered.

The following example shows the Validate URI page with test values defined for each of the variables in the `weather/{state}/{city}?forecast={day}` URI template. The URL field contains the generated URL.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Test values defined for the URI template and URL generated.

The following example shows the complete URL generated by the system based on the test values entered:

```
http://buffy.us.oracle:8920/PSIGW/PSIGW/RESTListeningConnector/WeatherStation.v1/weather/
```

Defining REST Service Operation Version Information

When you first create a service operation definition, the definition that you initially define is the default version.

When the newly created service operation definition opens, the Default check box is enabled and is read-only.

You can subsequently define additional service operation versions and assign them as the default.

See Chapter 10, "Managing Service Operations," Managing Service Operation Versions, page 261.

Defining General Version Information

To define the REST service operation default version:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Service Operations - General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Version field, enter a version identifier.

   The default is *v1*.
3. (Optional.) In the Version Description field, enter a description for the operation version.

   If you enter no information, the description by default is the name of the service operation when you save the definition.
4. (Optional.) In the Version Comments box, enter comments about the version.
5. (Optional.) Check the Runtime Schema Validation check box to enable service schema validation at runtime.

Continue to the next section to specify messages for service operations. You cannot save the service operation definition until you define messages for it.

Specifying Messages for REST Service Operations

You specify messages for service operations in the Message Information section of the Service Operations – General page.

The messages that you specify define the structure of the data that is contained in the service operation.

The REST method type determines the number of messages and message types (request or response) that you specify. The possible request and response message combinations by REST method type are described elsewhere in this chapter.

See Chapter 12, "Managing REST Service Operations," Understanding Managing REST Service Operations, page 287.

To specify messages for a REST service operation:

1. Locate the Message Information section on the Service Operations – General page.
2. Locate the Type field, and take note of the message type to define.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. From the Status Code drop-down list, select the HTTP response code to return to the integration partner.

   The valid HTTP status code values are:

   The Status Code edit box is the actual HTTP Response code that will be returned to the Consumer. The Content-Type defines the acceptable HTTP media type.

   - *200.* (Default.)
   - *201*. This value is valid only when the REST method is Post.
   - *202*
   - *203.* This value is valid only when the REST method is Get.
   - *204*
   - *205*
   - *206*
   - *207*

4. In the Message.Version field, enter the message name followed by a dot and version, or click the Lookup button to search for one.

   After you select the message, you can click the View Message link to view the message.

5. From the Content-Type drop-down list, select the MIME content type for the message. The valid values are:

   - application/json
   - application/xml
   - text/xml
   - text/plain
   - text/html

6. Repeat steps 1 through 5 for each message type that appears in the Message Information section.

7. Click the Save button.

Specifying Fault Messages for Service Operations

You can specify fault messages for service operations for error handling.

Note the following about fault messages:

- You cannot add fault messages to asynchronous service operations.
- Fault messages must be nonrowset-based messages, container messages, or document messages. Fault messages cannot be rowset-based messages.
- The HTTP status code for fault message is *400.*

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

To specify a fault message:

1. Locate the Default Service Operation Version section on the Service Operations – General tab.
2. Click the Add Fault Type button.

   A new row appears in which to specify a message. Note that the Type field in the new row displays *Fault.*
3. In the Message.Version field, enter the message name, or click the Lookup button to search for one.

   After you select the message, you can click the View Message link to view the message.
4. From the Content-Type drop-down list, select the MIME content type of the message.

   The valid values are:

   - application/json
   - application/xml
   - text/xml
   - text/plain
   - text/html
5. From the Status drop-down list, select a status code. The default and only value is *400.*
6. Click the Save button.

To delete a fault message, in the Default Service Operation Version section, click the Delete Fault Type button. Then click the Save button.

Managing Provider REST Service Operations

This section describes configuration tasks and considerations specific to managing provider REST service operations. This section discusses how to:

- Manage target connectors for provider REST service operations.
- Manage messages for provider REST service operations.
- Secure provider REST service operations.
- Add handlers to provider REST service operations.
- Manage routings definitions for provider REST service operations.
- Define routing header properties for provider REST service operations.

Managing Target Connectors for Provider REST Service Operations

Only the HTTP target connector is supported for providing REST service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Managing Messages for Provider REST Service Operations

For a provider REST service operations defined with a nonrowset-based message, the Root Element field on the message definition must be populated. The root element value you specify is used for proper WADL generation.

See Also

Chapter 6, "Managing Messages," Adding Rowset, Nonrowset or Part Message Definitions, page 79

Securing Provider REST Service Operations

This section discusses securing provider REST service operations.

Requiring Secured Inbound Consumer Requests

When you create provider REST services you can set the level of security with which inbound service operations must be sent using options in the Req Verification drop-down list on the Service Operations page. If you set a required inbound security level and inbound messages do not meet the requirement, the messages are rejected.

The following security levels are supported for provider REST service operations: none, SSL, basic HTTP authentication, and basic HTTP authentication and SSL.

- Basic HTTP authentication.
- Basic HTTP authentication and SSL.
- None.
- SSL.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Validating Security on Inbound Integrations.

Setting Permissions for Provider REST Service Operations

Provider REST service operations are secured using permission lists. The process for setting permissions for provider REST service operations is identical to the process for setting permissions for non-REST service operations.

See Chapter 10, "Managing Service Operations," Setting Permissions to Service Operations, page 260.

Adding Handlers to Provider REST Services Operations

The only Handler Type available for provider REST services is the OnRequest handler and the handler is implemented as an application class. The application class is implemented as any other OnRequest integration broker event.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Implementing handlers is discussed in greater detail elsewhere in this PeopleBook

See Chapter 16, "Managing Service Operation Handlers," page 347.

See Chapter 7, "Sending and Receiving Messages," Using PeopleCode to Manage REST Service Operations, page 205.

Managing Routing Definitions for Provider REST Service Operations

This section discusses how to manage routing definitions for Provider REST service operations.

Working with System-Generating Routing Definitions for REST Service Operations

An any-to-local routing is always created for a provider REST service operation you initially save the service operation definition. The routing name is a system-generated name with the format *~GENERATED~<unique number>*. You cannot delete this system-generated routing but you can inactivate it. You can also regenerate the any-to-local routing in the Routing Actions Upon Save section on the Service Operations-General tab. Note that if you regenerate the routing and request or response headers you have defined on the routing are deleted.

You can also generate a local-to-local routing for any REST-based service operation in the Routing Actions Upon Save section on the Service Operations-General tab. You may also inactivate local-to-local routings you create for a REST service.

Any-to-local and local-to-local routing types are discussed elsewhere in this PeopleBook.

See Chapter 17, "Managing Service Operation Routing Definitions," Routing Types, page 371.

You cannot create point-to-point routings for REST-based service operations.

Additional information about configuring and working with routing definitions is described elsewhere in this PeopleBook.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Managing System-Generated Routing Definitions, page 376

Chapter 17, "Managing Service Operation Routing Definitions," Configuring Routing Definitions, page 382

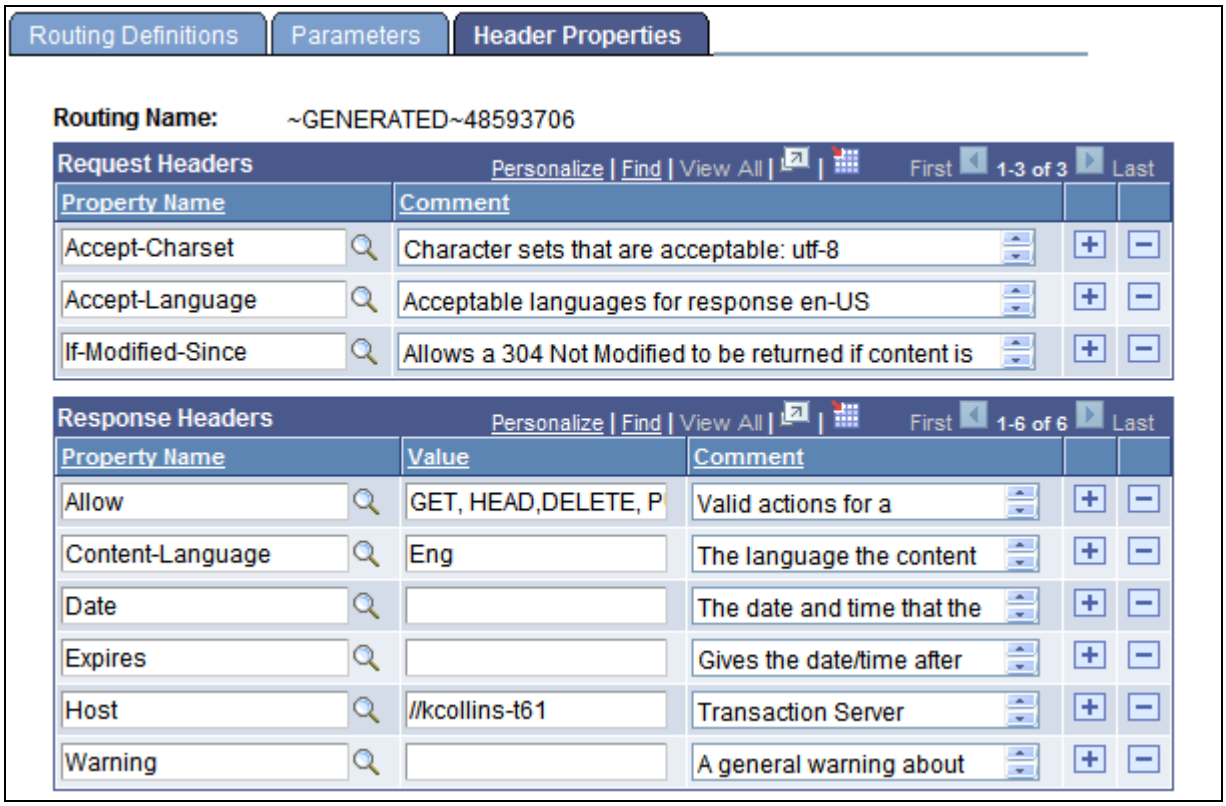
Chapter 17, "Managing Service Operation Routing Definitions," Activating and Inactivating Routing Definitions, page 403

Defining Routing Header Properties for Provider REST Service Operations

On provider REST–based service operations the any-to-local routing definition features a Header Properties tab (IB_ROUTINGRESTPROP) where you add request message and response headers to the service operation. The headers are shown in the generated WADL document for the service operation.

The following example shows the Header Properties tab:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Header Properties tab for a generated any-to-local routing for a REST-based provider service operation.

The request and response headers that you can add are standard HTTP request and response header fields. There are several sources for addition information and descriptions of HTTP header fields. One such source is the World Wide Web Consortium (W3C).

See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

Note. If you regenerate a routing any header fields defined for the routing are deleted and you must recreate them.

You also use this page to define compression on provider service operations requests and responses.

See Chapter 12, "Managing REST Service Operations," Setting Compression for REST Service Operations, page 315.

Use the LoadRESTHeaders method of the IBInfo class to load the response and request headers at runtime.

See Chapter 7, "Sending and Receiving Messages," Using PeopleCode to Manage Provider REST Service Operations, page 206.

Adding Request Header Properties

To add a request header property:

1. Access the Routings-Header Properties page (PeopleTools, Integration Broker, Integration Setup, Routings and click the Header Properties tab).

2. In the Request Headers section of the page, click the plus (+) button to add a row.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. In the Property Name field, click the Lookup button and select a property to add.

   The system adds the property to the grid.

4. In the Comments field, enter a comment about the property.

5. Repeat steps 2–4 to add additional request header properties.

6. Click the Save button.

Adding Response Header Properties

To add a response header property:

1. Access the Routings-Header Properties page (PeopleTools, Integration Broker, Integration Setup, Routings and click the Header Properties tab).

2. In the Response Headers section of the page, click the plus (+) button to add a row.

3. In the Property Name field click the Lookup button and select a property to add.

   The system adds the property to the grid.

4. In the Value field enter a value for the property.

5. In the Comment column enter a comment for the property.

6. Repeat Steps 2–5 to add additional response header properties.

7. Click the Save button.

Managing Consumer REST Service Operations

This section describes configuration tasks and considerations specific to managing consumer REST service operations. This section discusses how to:

- Add handlers to consumer REST service operations.
- Manage routing definitions for consumer REST service operations.
- Secure consumer REST service operations.

Adding Handlers to Consumer REST Service Operations

There are no handlers for consumer REST service operations. You may add connector properties to consumer service operations on the routing definition using the Routings-Connector Properties page or add them dynamically by using PeopleCode.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining and Overriding Gateway and Connector Properties, page 393 and Chapter 7, "Sending and Receiving Messages," Using PeopleCode to Manage REST Service Operations, page 205.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Manage Routing Definitions for Consumer REST Service Operations

The system creates a single outbound local-to-WADL node routing when you initially save a consumer REST service operation. The routing name is a system-generated name with the format *~GENERATED~<unique number>*. You cannot delete this system-generated routing but you can inactivate it.

Additional information about configuring and working with routing definitions is described elsewhere in this PeopleBook.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Managing System-Generated Routing Definitions, page 376

Chapter 17, "Managing Service Operation Routing Definitions," Configuring Routing Definitions, page 382

Chapter 17, "Managing Service Operation Routing Definitions," Activating and Inactivating Routing Definitions, page 403

Securing Consumer REST Service Operations

A PeopleSoft REST service operation provider may require that consumer services be sent secured. PeopleSoft REST providers may communicate to you that they require consumer REST service operation be secured using SSL, basic authentication, or basic authentication and SSL.

Securing Consumer REST Requests Using SSL/TLS

You can determine any SSL requirement based on the REST base URL the provider supplies and as defined on the Service Operations – General page. If the URL is a Hypertext Transfer Protocol Secure URL (HTTPS), then the provider has implemented SSL and your consumer request must also be sent using the SSL protocol. Information for setting up SSL/TLS is provided elsewhere in PeopleBooks.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Implementing Web Server SSL/TLS Encryption.

Securing Consumer REST Requests Using Basic Authentication

If a PeopleSoft REST provider system is defined in your integration network, you can use basic HTTP authentication or PeopleSoft tokens to secure consumer requests and the PeopleSoft providing system accepts the requests as meeting the basic authentication requirement. When using PeopleSoft tokens, the system sends the token as an HTTP header property as part of the request. This behavior allows you to securely pass user credentials for proper authentication instead of using basic authentication which requires that you set an external user name and external password on the routing. If the provider has basic authentication selected for inbound security verification and no PeopleSoft token or basic authentication credentials are passed, a basic authentication pop-up window appears and the user must enter his or her credentials at that time. The PeopleSoft token option is a way to not have to prompt for user credentials for PeopleSoft-to-PeopleSoft integrations.

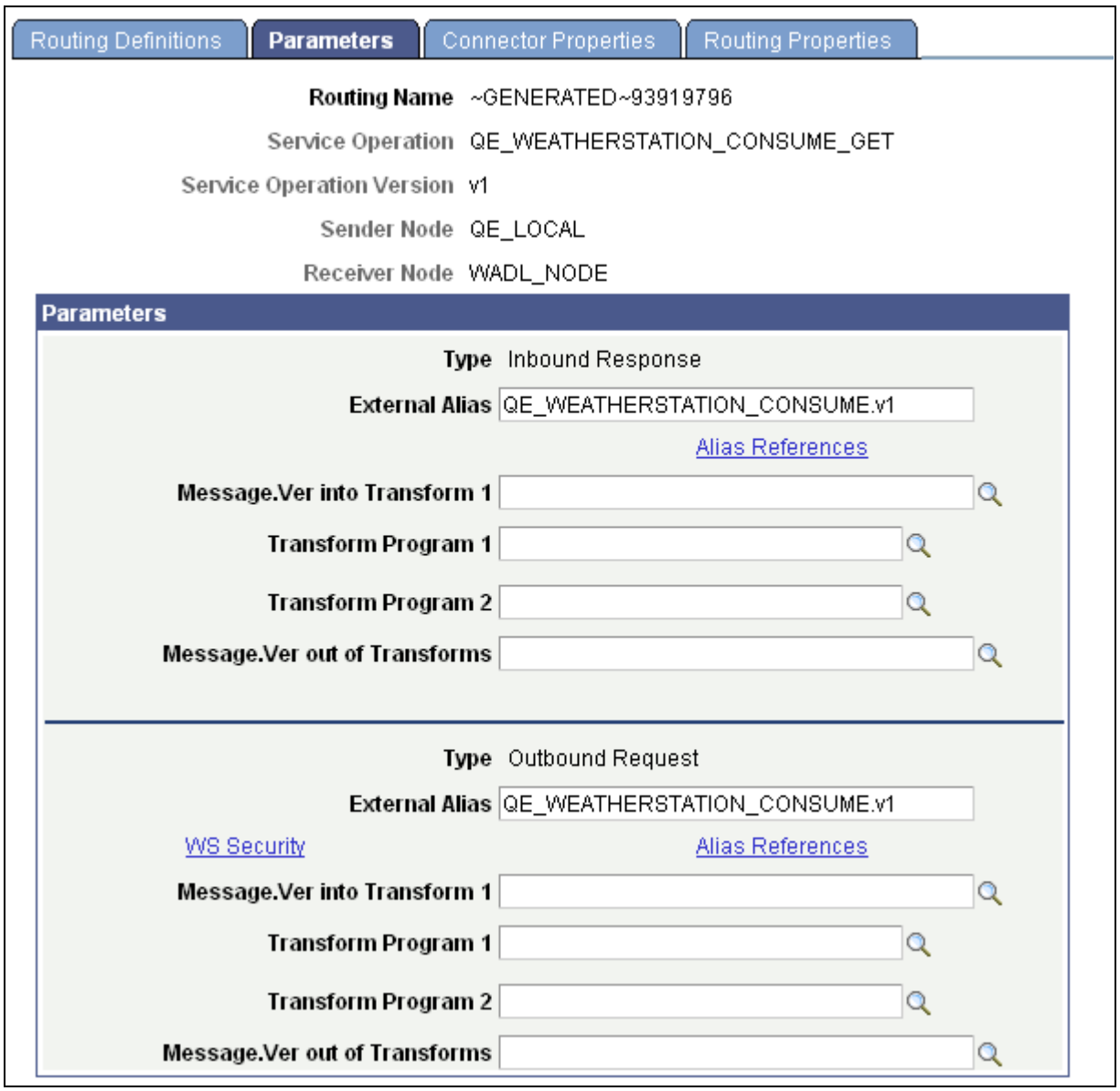
 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

If a PeopleSoft REST provider is not in your integration network and the provider requires inbound basic authentication, you cannot secure the request with a PeopleSoft token. The providing system will reject such a request. You must secure the request using basic HTTP authentication.

If the consumer system is a third-party system and the provider requires inbound basic authentication, the third-party system must secure the request using basic HTTP authentication.

You specify basic HTTP authentication for a consumer outbound request on the routing definition on the Routings-Parameters page (IB_ROUTINGDEFNDOC). The Parameters section for the outbound request features a WS Security link that enables you to set the security level for the outbound consumer request.

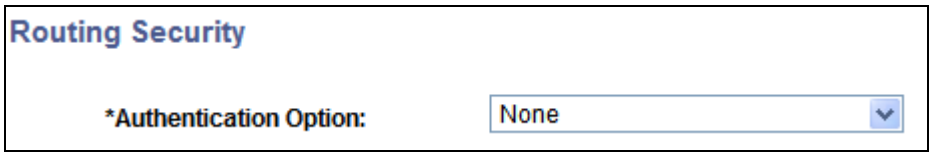
To access the Routings-Parameters page select PeopleTools, Integration Broker, Integration Setup, Routings and click the Parameters tab. The following example shows the page:



Click the WS Security link to set the security level for outbound requests.

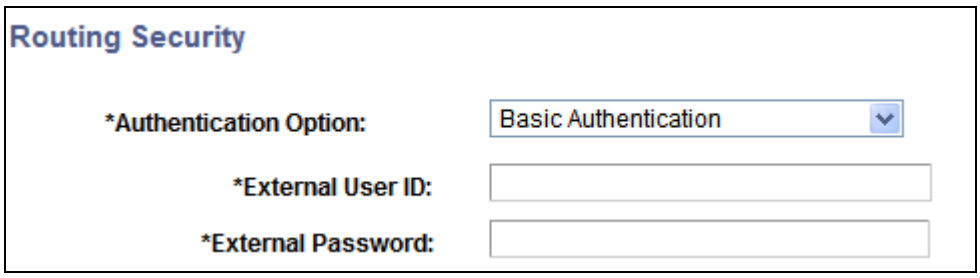
When you click the WS Security link the Routing Security page (IB_ROUTINGDEFN_SEC) appears, as shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Choose an authentication option from the drop-down list.

Use Authentication Option drop-down list to select the authentication option for the request. If you select basic authentication as the authentication option, External User ID and External Password fields appear and you must enter the external user ID and password values as furnished by the provider. The following example shows the Routing Security page when Basic Authentication is selected as the authentication option:



Specifying an External User ID and External Password for basic authentication

Keep in mind that in addition to basic authentication, the provider may require that you send the request secured by SSL.

To secure a REST consumer request using basic authentication:

1. Access the Routings-Parameters page (PeopleTools, Integration Broker, Integration Setup, Routings and click the Parameters tab).
2. In the parameters section for the outbound request, click the WS Security link.

   The Routing Security drop-down list appears:
3. From the Authentication Option drop-down list, select an authentication method. The options are:

   - *Basic Authentication.*

     If you select this option, go to step 4.
   - *None.* (Default)

     If you select this option, go to step 6.
   - *PeopleSoft Token.*

     If you select this option, go to step 6.
4. In the External User ID field, enter the ID furnished to you by the REST provider.
5. In the External Password field, enter the password furnished to you by the REST provider.
6. Click the OK button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Setting Up Secure Integration Environments," Validating Security on Inbound Integrations

Managing REST Service Operation Versions

You can create and use different versions of service operations. The process for managing REST service operation versions is identical to the process for managing non-REST service operation versions.

See Chapter 10, "Managing Service Operations," Managing Service Operation Versions, page 261.

Note, however, that REST resource definition are defined at the service operation level and not the service operation version level. As a result, REST resource definitions cannot be versioned.

Setting Compression for REST Service Operations

This section discusses how to:

- Set compression for provider REST service operations.
- Set compression for consumer REST service operations.

Understanding Setting Compression for REST Service Operations

PeopleSoft supports gzip and base64 (deflate) compression.

A REST service operation provider may request that inbound consumer requests be compressed using gzip or base64 (deflate) compression. The consumer is not required to send the data compressed.

The provider specifies the compression preference and the type of compression that the provider system can process on the routing definition using the Routings-Header Properties page. The compression preferences are then included in the WADL document that the provider furnishes to the consumer. The REST consumer can then view the compression preferences of the provider.

A REST service operation consumer may specify that a response from the provider be compressed using gzip or base64 (deflate) compression. The consumer specifies this request on the outbound request routing definition using the Routings-Connector Properties page. The compression requirement for the provider response is then included in the HTTP target connector headers. The REST provider can then ensure that its response meets the compression requirement of the consumer.

Setting Compression for Provider REST Service Operations

This section discusses how to set compression for Provider REST service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Setting Compression for Provider REST Service Operations

You use the Routings-Header Properties page (IB_ROUTINGRESTPROP) to specify the compression method that the providing system will accept on inbound requests from a consumer, as well as the compression method the provider is using on the response to the consumer.

The two properties to use to specify message compression on the provider system are Accept-Encoding and Content-Encoding.

For provider request and response headers set compression as follows:

Provider Request Header

Use the Accept-Encoding property for the Request Header to indicate the compression method the providing system will accept on inbound requests from the consumer. The information you specify here is included in generated WADL documents that you provide consumers.

There are no values to select, rather you specify the compression method in the Comments field. The valid compression methods are:

- *base64 (deflate).*
- *gzip.*

Provider Response Header

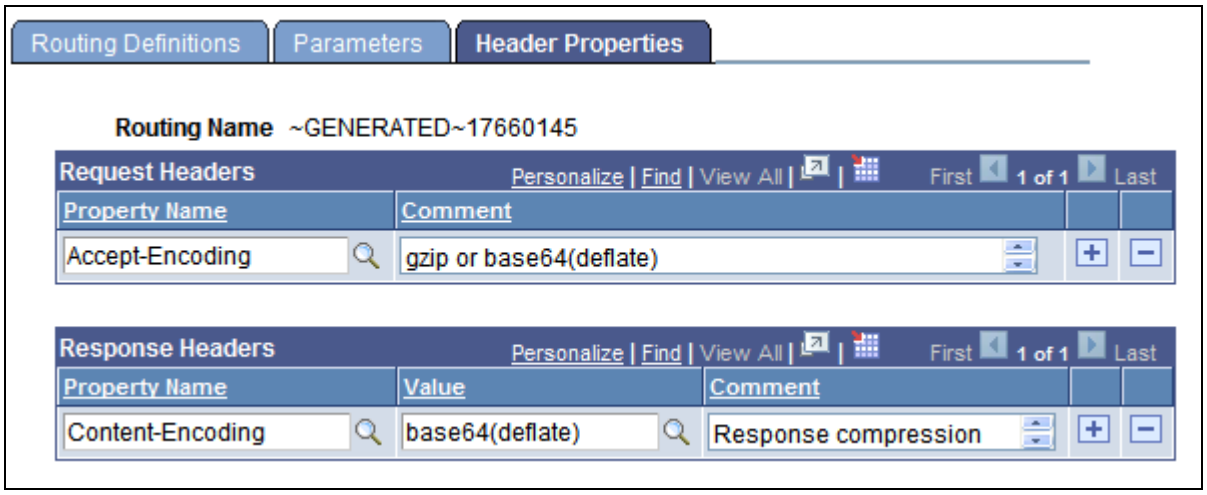
Use the Content-Encoding property for the Response Header to indicate the compression method used on the outbound response to the consumer. The information you specify here is included in generated WADL documents that you provide consumers. The compression method used should match the compression method that the consumer specifies in the connector properties in the inbound request. The valid values are:

- *base64 (deflate).*
- *gzip.*

Setting Compression for Provider REST Service Operations

The following example shows the Routings-Header Properties page that you use to set compression for provider REST service operations. To access the page, select PeopleTools, Integration Broker, Integration Setup, Routings and click the Header Properties tab:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Setting compression on the provider system.

The previous example shows that the Request Header is set to Accept-Encoding and that the provider will accept requests that are gzip or base64 (deflate) compressed. The example shows that the Response Header is set to Content-Encoding and that the provider will send its response to the consumer base64 (deflate) compressed.

To set compression for provider REST service operations:

1. Access the Routings–Header Properties page:
    a. Select PeopleTools, Integration Broker, Integration Setup, Services and in the Existing Operations section click the name of the service operation. The Service Operations – General page appears.
    b. Click the Routings tab and in the Routing Definitions grid click the name of the routing definition. The Routing-Definitions page appears.
    c. Click the Header Properties tab.
2. Set the request header.

    In the Request Headers section:
    a. In the Property Name field click the Lookup button and choose *Accept-Encoding* from the list.
    b. In the Comment field enter a comment regarding the compression method the provider system will accept for the inbound request from the consumer.
3. Set the response header.

    In the Response Headers section:
    a. In the Property Name field click the Lookup button and choose *Content-Encoding* from the list.
    b. In the Value field click the Lookup button and choose the compression used for the response message to the consumer. The valid values are*base64 (deflate)* or *gzip.*
4. Click the Save button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The other options featured on the Routings-Header Properties page are discussed elsewhere in this PeopleBook.

See Chapter 12, "Managing REST Service Operations," Defining Routing Header Properties for Provider REST Service Operations, page 309.

Defining Compression for Consumer REST Service Operations

This section discusses how to define compression for consumer REST service operations.

Understanding Setting Compression for Consumer REST Service Operations

A WADL document furnished to a REST consumer by a REST provider indicates if requests must be compressed and if so the compression method the provider system accepts. In addition, as a REST consumer you may require that responses from the provider be compressed and that the system can handle a specific compression method.

The two properties to use to specify message compression on consumer systems are Accept-Encoding and Content-Encoding.

For consumer request and response headers set compression as follows:

Consumer Response Headers Use the Accept-Encoding property to indicate to the provider the compression method to use in its response to your consumer system. The valid options are:

- *base64 (deflate).*
- *gzip.*

Consumer Request Headers Use the Content-Encoding property to indicate to the provider the compression method used on the outgoing request. The valid options are:

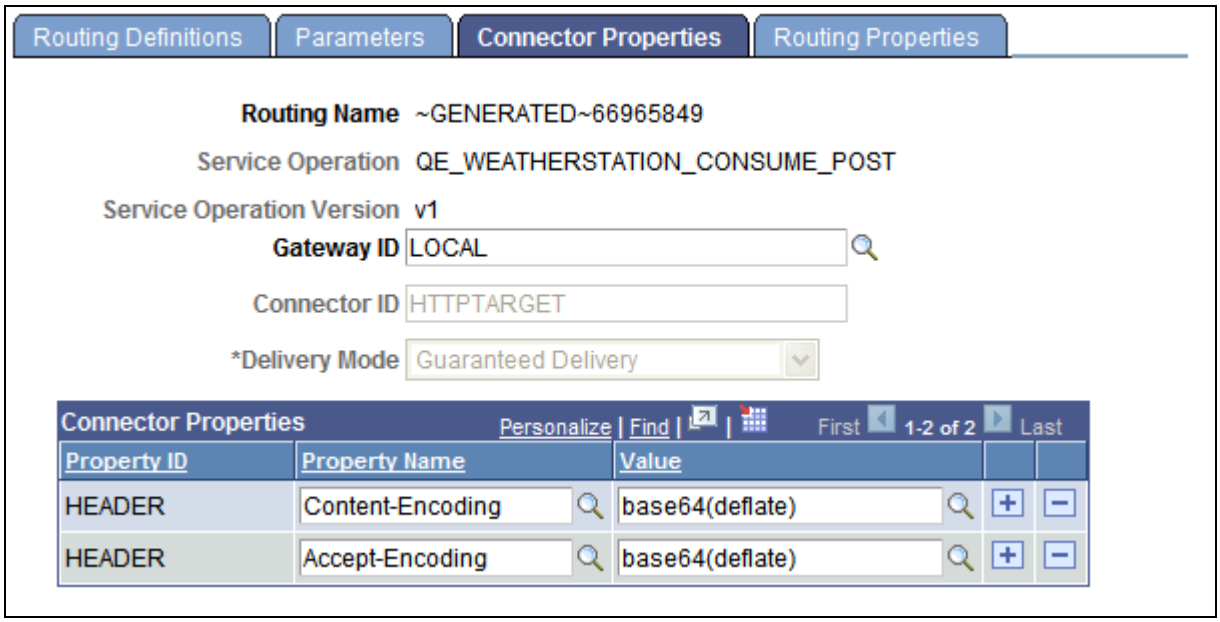
- *base64 (deflate).*
- *gzip.*

Setting Compression for REST Consumer Service Operations

You use the Routings-Connector Properties page (IB_ROUTINGRESTCPRP) to specify the compression method used for the outbound request and the compression method(s) the system can accept for the provider's response.

To access the Routings-Connector Properties page, select PeopleTools, Integration Broker, Integration Setup, Routings. The following example shows the page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Setting the compression on the consumer system.

The previous example shows that on the REST consumer has set Content-Encoding and Accept-Encoding to *base64 (deflate)*. By setting Content-Encoding to *base64(deflate)*, the consumer is indicating that it is sending the consumer request to the provider base64 (deflate) compressed. By setting the Accept-Encoding to *base64(deflate)*, the REST consumer is indicating to the REST provider that it will accept a response that is Base64 (deflate) compressed.

To set compression for consumer REST service operations:

1. Access the Routings–Connector Properties page:
    a. Select PeopleTools, Integration Broker, Integration Setup, Services and in the Existing Operations section click the name of the service operation. The Service Operations – General page appears.
    b. Click the Routings tab and in the Routing Definitions grid click the name of the routing definition. The Routing-Definitions page appears.
    c. Click the Connector Properties tab.
2. Set the request header.
    a. In the Property Name field click the Lookup button and choose *Content-Encoding* from the list.
    b. In the Value field click the Lookup button and choose the compression method used for the outbound request. The valid values are *base64 (deflate)* or *gzip*.
3. Set the response header.
    a. In the Property Name field click the Lookup button and choose *Accept-Encoding* from the list.
    b. In the Value field click the Lookup button and choose the compression method that the provider should use for the response back to the consumer. The valid values are *base64 (deflate)* or *gzip*.
4. Click the Save button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The other options featured on the Routings-Connector Properties page are discussed elsewhere in this PeopleBook.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining and Overriding Gateway and Connector Properties, page 393.

Renaming and Deleting REST Service Operations

You can rename and delete service operations using the Services Operations page in the Service Administration component. The process for renaming and deleting REST service operations is identical to the process for renaming and deleting non-REST service operations.

See Chapter 10, "Managing Service Operations," Renaming and Deleting Service Operations, page 268.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 13

Managing Service Operation Queues

This chapter discusses how to:

- Add queue definitions.
- Apply queue partitioning.
- Set permissions to queues.
- Rename and delete queue definitions.
- Delete queues during upgrade.

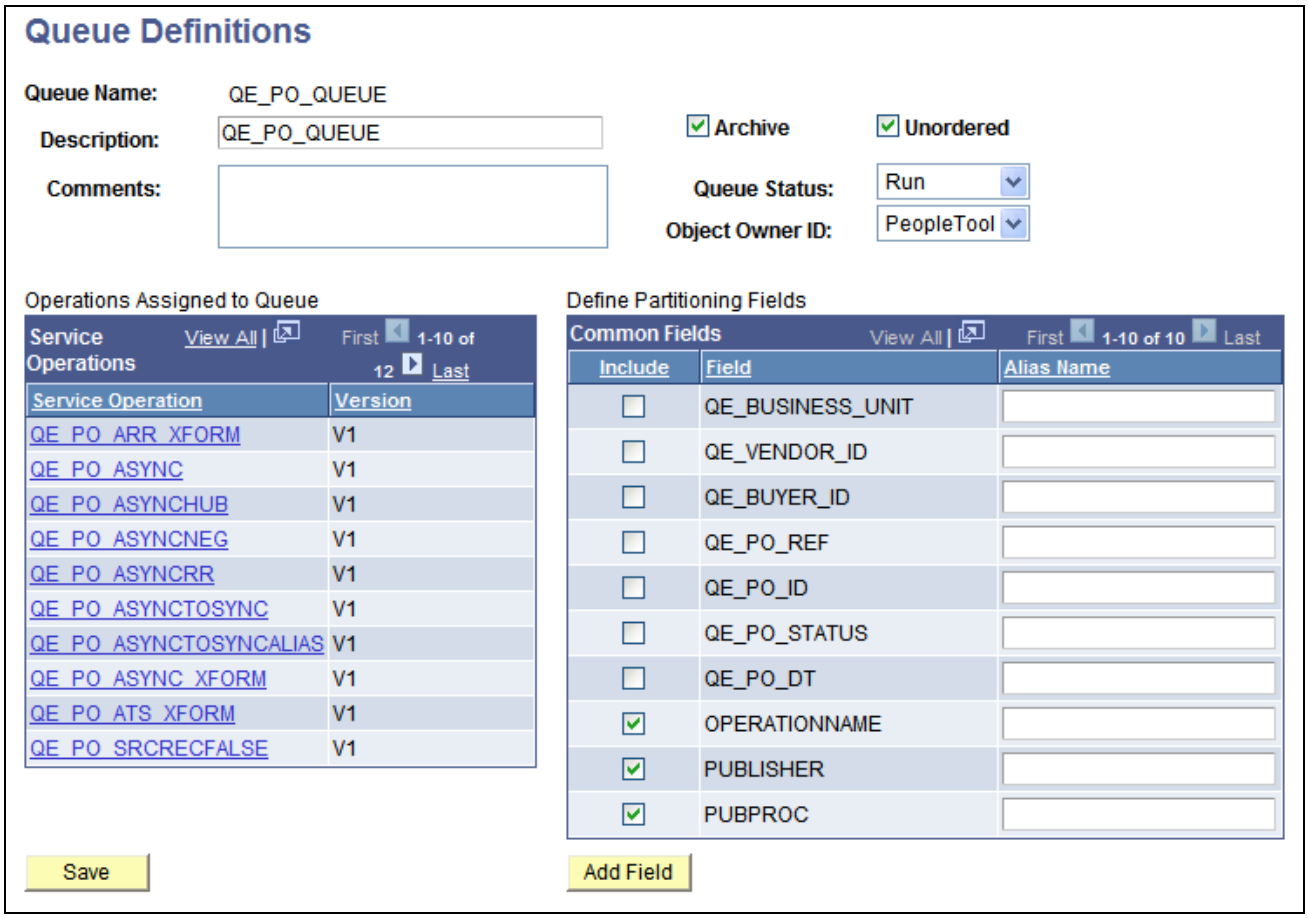
Understanding Service Operation Queues

Service operations queues are used to queue service operations for processing.

Adding Queue Definitions

Use the Queue Definitions page (IB_QUEUEDEFN) to add queue definitions to the system. To access the Queue Definitions page, select PeopleTools, Integration Broker, Integration Setup, Queues. The following example shows the Queue Definitions page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Queue Definitions page

The Operations Assigned to Queue box displays the names of the service operations assigned to a queue. Click a service operation name to open the service operation definition.

You work with the following page elements when you add a queue.

Queue Name Enter the name of the queue.

Archive Select to archive service operation instances that are assigned to the queue. By default, archiving is not enabled.

When you enable archiving, you can archive or delete transactions in the Asynchronous Details component of the Service Operations Monitor. This check box also controls whether the Archive or Delete actions are available in the Asynchronous Details component of the Services Operations Monitor.

If archiving is not enabled, the system purges the queue entries that have been processed.

Note. This setting controls whether archiving is available for asynchronous transactions only and pertains to transactions in a specific queue.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Unordered	Select to enable field partioning and to process service operations unordered. By default, the check box is cleared and inbound service operations that are assigned to a queue are processed one at a time sequentially in the order that they are sent. Select to force the channel to handle all of its service operations in parallel (unordered), which does not guarantee a particular processing sequence. This disables the channel's partitioning fields. Clear this check box if you want all of the queues's service operations processed sequentially or if you want to use the partitioning fields.
Description	Enter a description for the queue.
Queue Status	Values are: *Run:* (default) Service operations that are assigned to this queue are received and processed normally. *Pause:* Service operations are received but not processed until the status is reset to *Run.* **Note.** You can also pause and restart queues in the Service Operations Monitor.
Object Owner ID	From the drop-down list box, select the object owner. The owner ID helps to determine the application team that last made a change to the definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.
Comments	Use this area to enter comments about the definition.
Operations Assigned to Queue	This read-only section lists all service operations that are assigned to the queue.
Include	Select the Include check box next to a field name to include the field in queue partitioning.
Add Field	Click to view and select partitioning fields.

See Also

Chapter 13, "Managing Service Operation Queues," Applying Queue Partitioning, page 324

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Pausing, Testing, and Pinging Nodes"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Applying Queue Partitioning

This section provides an overview of queue partitioning and discusses how to select partitioning fields.

Understanding Queue Partitioning

This section discusses queue partitioning

Queue Partitioning

By default, all inbound service operations that are assigned to a given service operation queue are processed one at a time, in the order they are sent. Consequently, the sending node can engage in a series of transactions that modify a specific record, and the changes are applied by the receiving node in the correct order. This can be inefficient if the sequence does not matter or if the sequence is relevant only within groups of service operations with the same key values. One solution to this inefficiency is partitioning.

Note. Partitioning applies only to asynchronous messaging.

To maximize messaging efficiency and throughput, you want the system to simultaneously process as many service operations as possible. Because queues enforce service operation sequence, ideally you have a separate service operation queue for each group of service operations that must be processed in order. You can achieve this goal by designating specific fields that are common to the service operations that are assigned to a queue. These fields partition, or divide, the queues into subqueues. PeopleSoft Integration Broker creates these subqueues at runtime.

Each subqueue processes only the service operations for which the designated common fields have an identical combination of values. The service operations within each subqueue are processed in the order that they are sent, so they remain in sequence. Each subqueue works in parallel with the other subqueues—all subqueues simultaneously process their own associated service operations.

You implement partitioning by designating the partitioning fields in the queue definition; no other steps are required.

Note. The more partitioning fields that you designate, the more subqueues are generated. If you designate a combination of fields that are unique for each service operation, all service operations are processed simultaneously, in their own subqueues, without regard to sending order. This is the equivalent of selecting the Unordered check box in the queues definition.

Queue Partitioning Fields

When working with queue partitioning fields, note the following points:

- The maximum length of a queue partitioning value should not exceed 60 characters. The system truncates any value longer than 60 characters when you use the PeopleCode SubQueueName property.
- Fields of types image and long character aren't available for partitioning.
- When you are partitioning nonrowset-based or inbound messages, tags cannot be mixed case.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Selecting Partitioning Fields

You can partition queues using any combination of:

- Database record fields.
- Message header fields.
- Message XML fields.

Database Record Fields

Database record fields are the data fields in a PeopleSoft rowset-compatible message. Typically, the more service operations that you assign to the queue, the fewer fields they have in common.

The database record fields that are common to all service operations in the queues appear in the queues's Common Field list. If only one service operation is assigned to the queues, all of its fields appear on the partitioning list.

To designate a field as a partitioning key, select the check box next to its name.

Message Header Fields

The message header fields are system-maintained fields that are common to all service operations, regardless of format. If a queue includes any nonrowset-compatible service operations, the message header fields are the only ones that PeopleSoft Integration Broker recognizes as common to every service operation. You can view them as part of the message XML in Integration Broker Monitor. You can also access some of them using equivalent PeopleCode Message class properties, as indicated later in this chapter. The message header fields are:

- OPERATIONNAME: This field contains the name of the operation in the PeopleSoft Pure Internet Architecture.
- PUBLISHER: This field contains the user ID that is in effect when the service operation is published, that is, the ID of the user who is signed in to the publishing database.
- PUBPROC: This field refers to the PeopleSoft process that publishes the service operation.

  It is generated when the service operation is published, and it can be the name of a component, an Application Engine program, or an iScript program.

These header fields are always available in the queue's partitioning field list. To designate a field as a partitioning key, select the Include check box next to its name.

Message XML Fields

The message XML fields comprise all the fields that exist anywhere in a message, including PeopleSoft common application message attributes (PSCAMA) record fields. Such fields may not be visible in the queues's partitioning field list, but they are valid for partitioning. Message XML fields can have aliases, allowing for support of mixed-case names.

To designate a message XML field as a partitioning key:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Queue Definitions page (PeopleTools, Integration Broker, Integration Setup, Queues.)
2. Click the Add Field button.
3. Enter the tag name of the XML field, or click the Lookup button to search for one.

   The value does not have to be in the database.

   All names are uppercase by default. You can then add an alias, which can be mixed-case for partitioning.

At runtime, the integration engine searches each message for the first instance of that field tag and uses the value that is associated with it for partitioning. Therefore, if you have common fields in the PSCAMA record that are specific to a batch publish set, you can use those fields.

Renaming and Deleting Queues

You can rename and delete queue definitions using the Queue page (IB_HOME_PAGE3) in the Service Administration component (IB_HOME_PAGE).

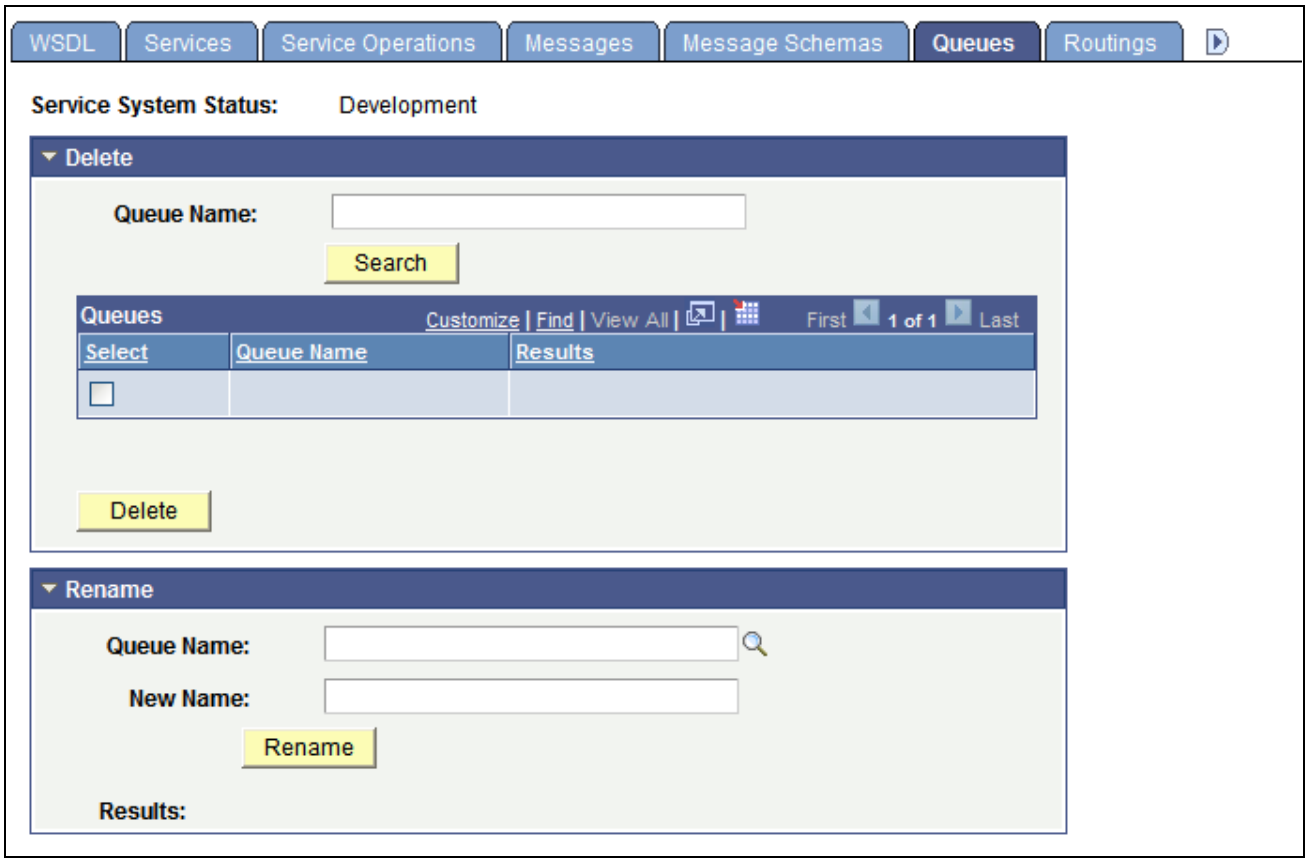
The Queues page contains two sections: a Delete section that enables you to delete a queue definition and a Rename section that enables you to rename a queue definition.

To access the page, select PeopleTools, Integration Broker, Service Utilities, Services Administration and select the Queues tab.

When you first access the page, both sections are collapsed. Click the section-header arrow buttons to expand and collapse each section.

The following example shows the Queues page with the Delete and Rename sections expanded:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Service Administration — Queues page with Delete and Rename sections expanded.

The top of the page displays a Service System Status field with the current setting, as defined on the Services Configuration page. This setting affects the ability to rename and delete queues.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services."

Renaming Queue Definitions

To rename a queue definition:

1. Access the Services Administration - Queues page (PeopleTools, Integration Broker, Service Administration, click the Queues tab).
2. Click the arrow next to the Rename section header to expand the section.
3. In the Queue Name field, enter the queue definition to rename, or click the Lookup button to search for and select the queue to rename.
4. In the New Name field, enter the new name for the queue definition.
5. Click the Rename button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Deleting Queue Definitions

To delete a message definition:

1. Access the Services Administration - Queues page PeopleTools, Integration Broker, Service Administration, click the Queues tab).
2. Click the arrow next to the Delete section header to expand the section.
3. In the Queue Name field, enter the name of the queue definition to delete, and click the Search button.

   Search results appear in the results grid.
4. In the results grid, select the check box next to the queue or queues to delete.
5. Click the Delete button.

Deleting Queues During Upgrade

To delete a queue definition in an application upgrade project, first ensure that no live instances of messages are assigned to that queue. Archive or delete any such messages in both the source and the target database. Otherwise, an error message appears during the copy process indicating that the message is in use.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 14

Enabling Runtime Message Schema Validation

This chapter discusses how to:

- Select service operations.
- View defined message schemas.
- Enable runtime message schema validation.

Understanding Message Schema Validation

PeopleSoft Integration Broker enables you to validate, at runtime, the messages defined in service operations against message schemas.

Message Schema Validation

Validating messages against message schemas can ensure that during integration development no changes or deletions were inadvertently made to a message. You can use schema validation on outbound and inbound messages.

During runtime schema validation, PeopleSoft Integration Broker checks messages to ensure that the XML structure is valid according to the specified message schema. If Integration Broker encounters a service operation with a message structure that does not adhere to the specified message schema, message delivery fails and Integration Broker generates an error message.

Schema validation is based on the message schema for the message.version defined on the default service operation.

You can view any schema validation error details on the source application server, using the Asynchronous Details page or Synchronous Details page in the Service Operations Monitor.

When schema validation is enabled the structure of a message cannot be changed.

Message Schema Validation and Transformations

In instances where an inbound transaction has a transformation applied, the input message.version is used for schema validation. If an input message.version is not defined, then validation takes place using the request message.version of the default service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

For the case where a point-to-point routing and an any-to-local routing are both defined, the system uses the point-to-point routing to check for a transform.

Message Schema Validation and Part Messages

PeopleSoft Integration Broker does not perform parser validation on part messages in container messages.

URLs defined in container messages do not contain absolute paths. As a result, the parser cannot import these schemas to perform validation.

PeopleSoft internal validation is performed on the data and structure of part messages within contain messages. However, these validation results may be different than results achieved using parser validation.

Prerequisites for Validating Message Schemas

Before you can enable schema validation, you must build or import a message schema for the message and message version.

If no message schema is present for service operations that contain regular nonrowset-based messages or container messages, it is not possible to enable validation.

For service operations that contain rowset-based messages, the system will attempt to generate a schema if one is missing. If the system is able to successfully generate a message schema for a rowset-based message, validation is enabled.

See Chapter 8, "Building Message Schemas," page 213.

Selecting Service Operations

This section discusses how to select service operations that contain messages to validate against message schemas at runtime.

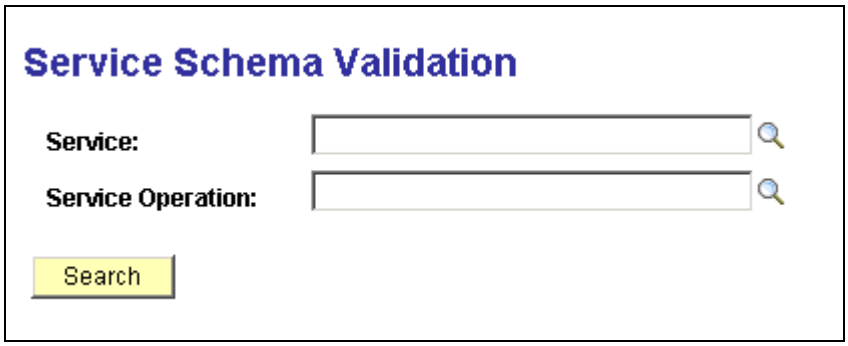
Selecting a Service Operation

The first step to enabling runtime message schema validation is to select the service operations that contain the messages to validate.

When you access the Service Schema Builder component (IB_SERVICEVAL), the Service Schema Validation page (IB_SERVICEVAL) appears and displays a search engine that you use to search for service operations.

To access the Service Schema Validation page select PeopleTools, Integration Broker, Service Utilities, Service Schema Validation.

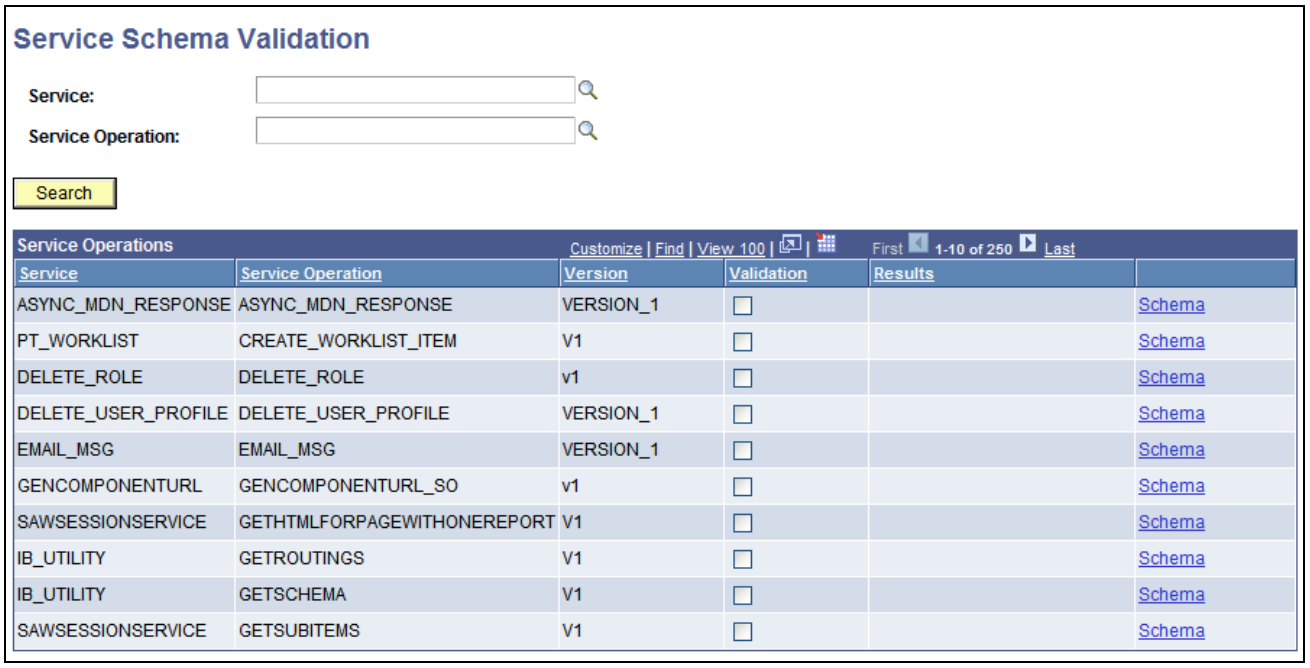
 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Service Schema Validation page

To search for a service operation, enter the service or service operation with which to work and click the Search button. A list of results displays in the Service Operations grid. If you do not enter any search criteria and click the Search button, the system returns all services and service operations in the database.

When you search for service schema validation data, the system returns the results in the Service Operations grid shown in the following example:



Service	Service Operation	Version	Validation	Results	
ASYNC_MDN_RESPONSE	ASYNC_MDN_RESPONSE	VERSION_1	☐		Schema
PT_WORKLIST	CREATE_WORKLIST_ITEM	V1	☐		Schema
DELETE_ROLE	DELETE_ROLE	v1	☐		Schema
DELETE_USER_PROFILE	DELETE_USER_PROFILE	VERSION_1	☐		Schema
EMAIL_MSG	EMAIL_MSG	VERSION_1	☐		Schema
GENCOMPONENTURL	GENCOMPONENTURL_SO	v1	☐		Schema
SAWSESSIONSERVICE	GETHTMLFORPAGEWITHONEREPORT	V1	☐		Schema
IB_UTILITY	GETROUTINGS	V1	☐		Schema
IB_UTILITY	GETSCHEMA	V1	☐		Schema
SAWSESSIONSERVICE	GETSUBITEMS	V1	☐		Schema

Schema search results appearing in the Service Operations grid

Service Indicates the name of the service.

Service Operation Indicates the name of the service operation.

Version Indicates the version of the service operation.

Validation Indicates if runtime schema validation is enabled.

When the check box is checked, schema validation is enabled.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Results Displays validation results.

The valid values are:

- *Error generating schema. Unable to turn on validation.*

  This message appears if one or more of the messages in the service operation is nonrowset-based and schemas do not exist for the nonrowset-based messages.

- *Service operation saved.*

  This message appears when you have successfully enabled runtime schema validation and saved the changes.

- *Error saving service operation.*

Schema Click the link to access message schemas for all messages defined on the service operation.

See Chapter 14, "Enabling Runtime Message Schema Validation," Viewing Defined Message Schemas, page 332.

Viewing Defined Message Schemas

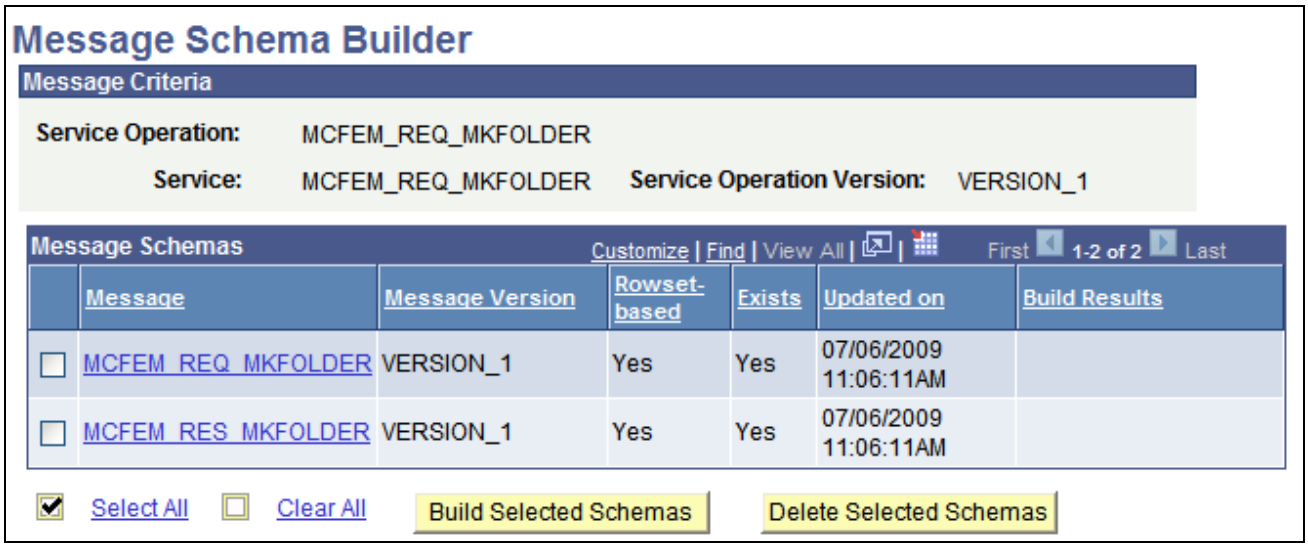
This section discusses how to view defined XML schemas for messages.

Viewing XML Schemas Defined for Messages

To view defined message schemas for all messages contained in a service operation, in the Service Operations grid, locate a service operation with which to work and click the Schema link. The systems displays the service operation in the Schema Builder page (IB_SCHEMABUILD).

The following example shows the Schema Builder page displaying two messages for the *MCFEM_REQ_MKFOLDER* service operation.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

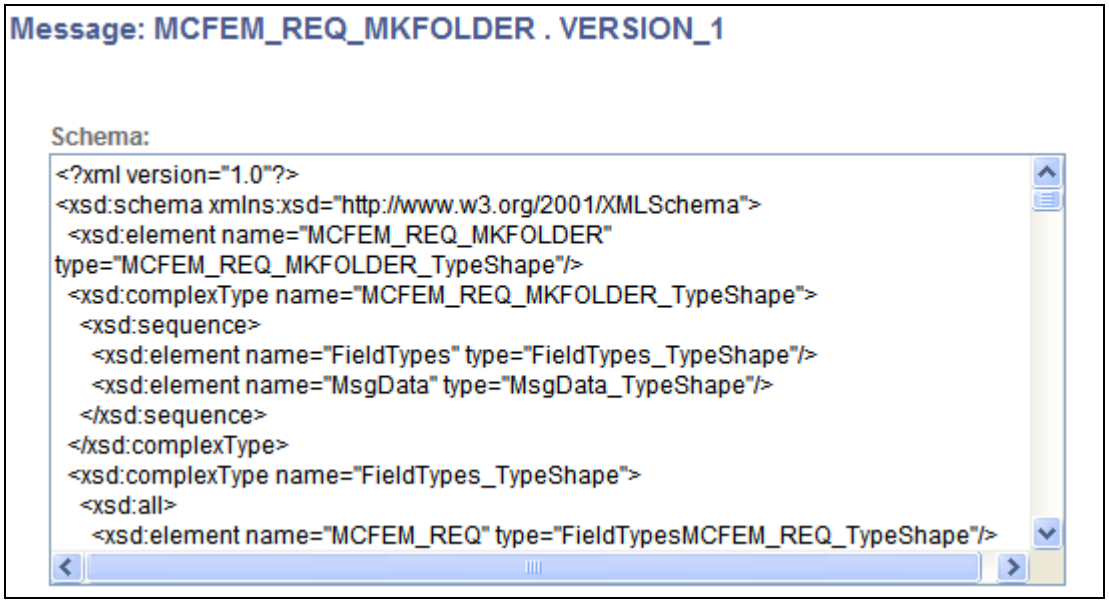


The Schema Builder page displaying messages for the MCFEM_REQ_MKFOLDER service operation.

The Exists field displays a value of *Yes* for both messages and indicates schemas have been built for both messages.

If schemas are not built for a message or messages, you can build them directly from this page by selecting the check boxes next to each message name and clicking the Build Selected Schemas button.

To view the XML schema for a message, click the message name link. The following example shows the XML schema for the *MCFEM_REQ_MKFOLDER* message.



XML schema for the MCFEM_REQ_MKFOLDER message.

To return to the Schema Builder page, click the Return button.

Using the Schema Builder page is documented in detail elsewhere in this PeopleBook.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 8, "Building Message Schemas," page 213

Enabling Runtime Message Schema Validation

This section discusses how to enable runtime schema validation for all messages defined in a service operation:

You can enable runtime message schema validation from the Service Schema Validation page or from the Service Operations page.

Using the Service Schema Validation Page to Enable Runtime Message Schema Validation

To enable runtime schema validation using the Service Schema Validation page:

1. Access the Service Schema Validation page (PeopleTools, Integration Broker, Service Utilities, Service Schema Validation).
2. Select a service operation that contains messages against which you want to validate message schemas.

   See Chapter 14, "Enabling Runtime Message Schema Validation," Selecting Service Operations, page 330.
3. Check the Validation check box.
4. Click the Save button.

Using the Service Operations page to Enable Runtime Message Schema Validation

To enable runtime schema validation using the Service Operations page:

1. Access the Service Operations page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. In the Default Service Operation Version section of the page, check the Runtime Schema Validation check box.
3. Click the Save button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 15

Creating Component Interface-Based Services

This chapter discusses how to:

- Select component interfaces to expose as services.
- Select component interface methods to include in service operations.
- Generate component interface-based services and service operations.
- View component interface-based service definitions.

Understanding Creating Component Interface-Based Services

PeopleSoft Integration Broker enables you to take an existing component interface and create a service that can be used to invoke the component interface. Further, it creates service operations, including request messages and response messages (if appropriate). The system creates an inbound any-to-local routing for the service operation version, as well as handlers for each method you choose to include in the service.

All service operations you generate from component interfaces are synchronous service operations.

Naming Conventions Integration Metadata Created

This section highlight the naming conventions that the PeopleSoft system uses when it creates services and services-related data based on component interfaces.

When it creates a web service from a component interface, the PeopleSoft system adds a *CI_* prefix to the component interface name. So if the component interface name is *MYCI*, the service name the system creates is *CI_MYCI*

The following table highlights naming conventions the PeopleSoft systems applies to other services data based on the method with which you are working:

Component Interface Method	***Service Operation Name***	***Message Name***	***Component Interface Handler***	***Request Message Shape***	***Response Message Shape***
Create	<service_name>_C	Mxxxxxx	ONREQUESTHDLR	CI Buffer	Key

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Component Interface Method	*Service Operation Name*	*Message Name*	*Component Interface Handler*	*Request Message Shape*	*Response Message Shape*
Find	<service_name>_F	Mxxxxxx	ONREQUESTHDLR	Get Keys	CI Buffer
Get	<service_name>_G	Mxxxxxx	ONREQUESTHDLR	Find Keys	Key Collection
Update	<service_name>_UP	Mxxxxxx	ONREQUESTHDLR	CI Buffer	Notification
Updatedata	<service_name>_UD	Mxxxxxx	ONREQUESTHDLR	CI Buffer	Notification
User-defined	<service_name>_<method_name>	Mxxxxxx	ONREQUESTHDLR	CI Buffer	Method Return Type

The naming convention used for message names, *Mxxxxxx*, is the letter M followed by a random six-digit number, as denoted by the x's. An example of a message name is *M548902.*

Note. The maximum number of characters for a service operation name is 30. If using a user-defined method name yields a greater result, the name is used is <service_name>_Mxxx, where xxx is a three-digit random number. An example of such a name is *CI_USERCI_M101023.*

User-Defined Method Restrictions

This section discusses restrictions related to using user-defined methods when exposing component interfaces as services.

User-Defined Method Actions

User-defined methods which are executed with the Create action should not be exposed as web services.

Specifying Method Parameters and Parameter Names

All user defined methods that will be accessed via the service interface must have a doc string.

The doc string contains:

1. The keyword *GET.* .
2. An ordered list of parameter names corresponding to the method's parameters.

The following example shows the format of the doc string:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Function simpleFunction(&sParm As string, &dParm As date,
                        &nParm As number, &bParm As boolean)
                        Returns string
   Doc "GET, StringParm, DateParm, NumParm , BoolParm"

   Local string &aString = &sParm;
   &aString = &aString | " -- " | &dParm;
   &aString = &aString | " -- " | &nParm;
   &aString = &aString | " -- " | &bParm;
   Return &aString;

End-Function;
```

The contents of the doc string are used when the function is invoked. For example, if GET is specified, then the GET keys are set, and GET is called on the component interface before the used-defined method is invoked.

The list of parameters are used at runtime to match the data in the input message with the method's parameters. This is an ordered list; if the parameter list in the doc string and the method parameters don't match, then the method may not work correctly. The names in the doc parameter list will be the names visible in any WSDL created for the service.

Method parameters and return values must be of a primitive type, such as a string, date or number. Object parameters or return values are not supported.

Impact of Changing Component Interfaces

If a user modifies or deletes a component interface in PeopleSoft Application Designer, when the user saves the changes, PeopleSoft Integration Broker checks if a service exists for the component interface. If a service exists, and the component structure/properties have changed in the component interface, a warning message appears stating that a service exists for the component interface.

If the component interface structure changes, a status of *Does not match* appears in the CI-Based Services Review Status grid, and the operation appears as active in the Service Operations component.

Prerequisites

Prior to creating any component interface-based web services and service operations, you must define the schema namespace, service namespace, and target location in the Service Configuration page

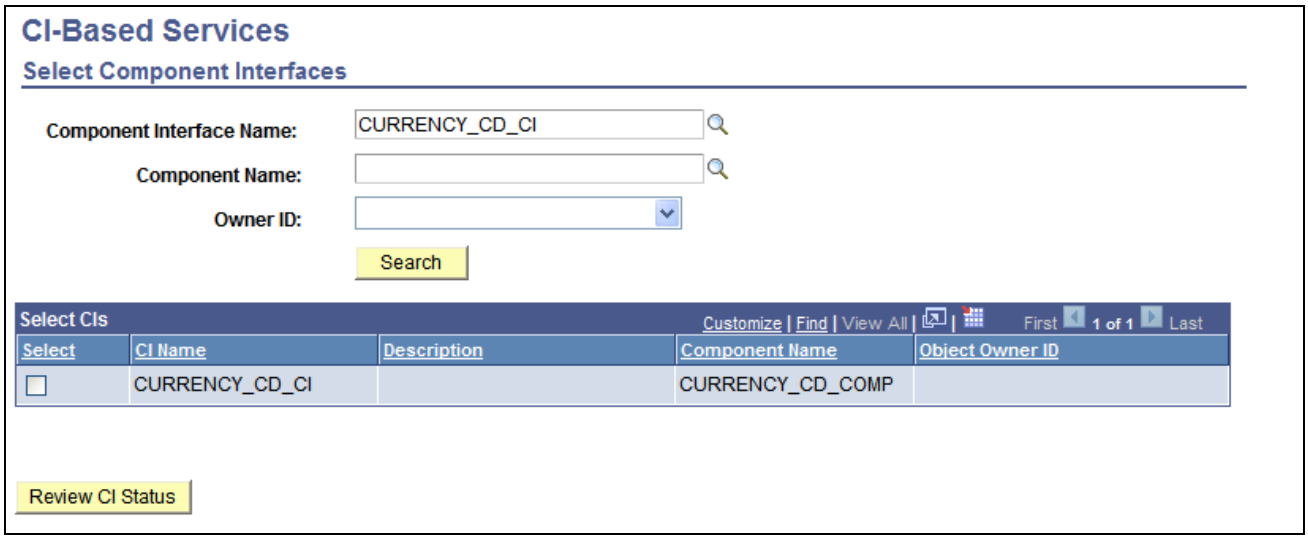
See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services."

Selecting Component Interfaces to Expose as Services

This section discusses how to select component interfaces to expose as web services.

The first step to creating component interface-based services is to choose the component interface on which to base a service. To do so, use the CI-Based Services component (IB_CISERVICES) and the Select Component Interface page (IB_CISERVICES). The following graphic shows this page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Select Component Interfaces page

When you search for component interfaces to select, the system returns results only for those component interfaces to which you have permissions.

To select a component interface:

1. Access the Select Component Interfaces page (PeopleTools, Integration Broker, Web Services, CI-Based Services).
2. Search for a component interface:

   Note. You can search only for those component interfaces to which you have permissions.

   - Click the Search button to search from all component interfaces in the database, or
   - Select one or more of the following criteria to narrow your search and then click the Search button.

     Component Interface Name Enter part or all of the name of the component interface to use, or click the Lookup button to search for one.

     Component Name Enter part or all of the component name to which a component interface belongs.

     Owner ID Owner ID drop-down list box, select the person or group that owns the component interface.

   The Select CIs grid displays are component interfaces that match your search.
3. Check the Select box next to one or more component interfaces.
4. Click the Review CI Status button.

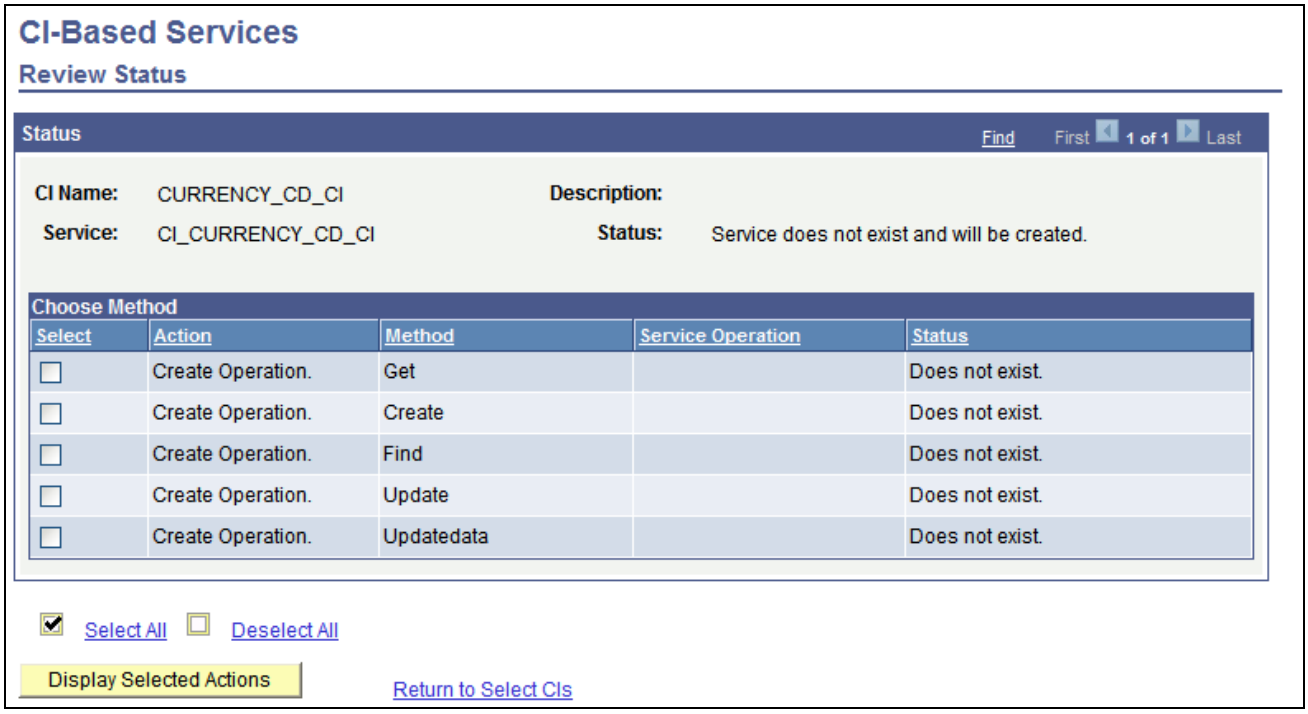
The Review CI Status page appears where you can review details about the select component interface and select the methods to include as operations in the service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Selecting Component Interface Methods to Include as Service Operations

This section discusses selecting methods to include web services as service operations.

The following example shows the Review Status page:



Review Status page

The Review Status page shows the following information about the component interface you select to expose as a service:

CI Name Name of the component interface.

Service Name of the service created based on the component interface.

Note that the service name is the component interface name with *CI_* added as a prefix.

Description A description of the component interface. This information is taken from the component interface record and displays if it exists.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Status (Service)

Displays the status of the service. The valid values are:

- *Service does not exist and will be created.* Indicates that a services does not exist for the service and the system will create one.
- *Service Exists.* Indicates that a service has already been created for the component interface.

Select

Check the box to include a method as an operation for the service.

If the box is disabled, the method has already been included in the operation and the Service Operation field displays the name of the operation created.

You can check one or more methods at a time.

Action

Displays the action available to perform on the method. The valid values are:

- *Create Operation.* This action displays if no service operation exists for the method.
- *Create New Version.*. This action displays if the current request or response shape do not match the shapes previously generated. A new service operation version is generated.
- *None.* If an operation exists, it compares the component interface and the service operation. If they are in sync, no action is required.

Method

Name of the component interface method.

The system displays user-defined and standard methods. All user-defined methods appear in lowercase.

You can create service operations based on the following standard component interface methods:

- *Create*
- *Get*
- *Find*
- *Update*
- *Updatedata*
- User-defined methods.

Update and Updatedata appear if both Get and Save have been enabled in the component interface.

Note. All user-defined methods are lowercase. If you have a user-defined method called *update,* it is a different method than the Update method used here.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operation Name of the service operation, if one exists for a method.

The name the system give the service operation depends on the service name as well as the component interface method.

Status (Service Operation) The status corresponds to the value in the Action field. The valid values are:

- *Does not exist.* No service operation exists for the method.
- *Does not match.* The service operation does not match the existing component interface.
- *OK.* The service operation matches the existing component interface.
- *Operation created.* The system created the operation.

Display Selected Actions Click the button to display a summary of the actions requested and then generate services and service operations.

Generating Component Interface-Based Services

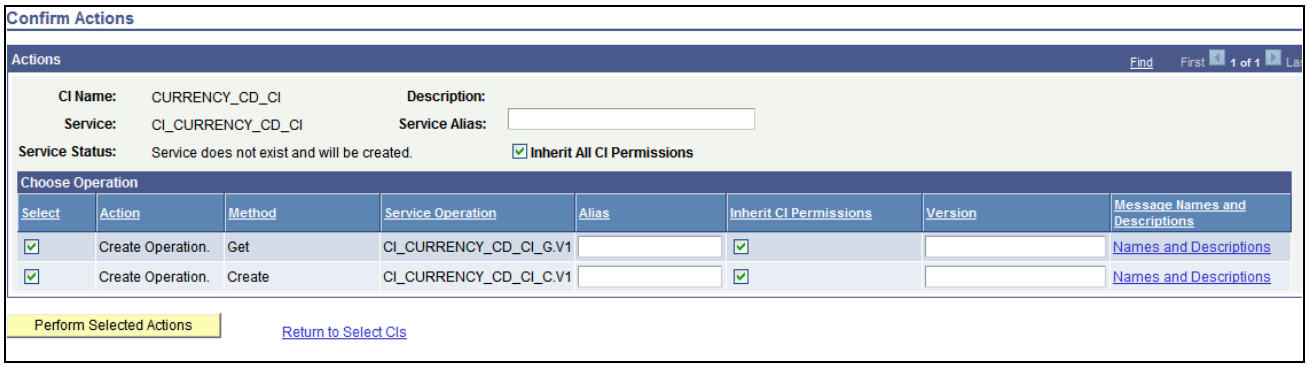
This section discusses how to:

- Generate services and service operations from component interface methods.
- Inherit component interface security permission lists.
- Add message names and descriptions to generated service operations.

Generating Services and Service Operations from Component Interface Methods

The following example shows the Confirm Actions page (IB_CISERVICES_SEC):

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Confirm Actions page

You can work with the following page elements:

Service Alias (Optional.) Enter an alias name for the service. The name you enter can be lower or mixed-case. If specified, this is the service name that appears in any WSDL documents you generate.

Inherit All CI Permissions Select the box so that all service operations you create inherit the security permissions as defined for the component interface after which they are based. By default the box is selected.

To inherit security permissions for specific service operations only, use the Inherit CI Permission box in the Choose Operation grid.

See Chapter 15, "Creating Component Interface-Based Services," Inheriting Component Interface Security Permission Lists, page 343.

Select (Optional.) Clear the box to omit creating a service operation for a method.

Alias (Service Operation) (Optional.) Enter an alias name for the service operation. The name you enter can be lower or mixed-case. If specified, this is the service operation name that appears in any WSDL documents you generate.

Inherit CI Permissions Select the box for a service operation to inherit the security permissions as defined for the component interface after which it is based. By default the box is selected for all service operations based off of a component interface.

Clear the box for each service operation you do not want to inherit the permissions defined for the component interface after which it is based.

See Chapter 15, "Creating Component Interface-Based Services," Inheriting Component Interface Security Permission Lists, page 343.

Version (Optional.) Service operations created default version *V1*.

You may enter a different value to use as the version when the service operation is created.

This field is a text field, so you may enter numeric or text values.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Message Names and Descriptions (Optional.) Click the link to specify names and descriptions for messages in the service operation. The message names and descriptions will appear in the generated XML message schema.

If you do not enter message names or descriptions, PeopleSoft Integration Broker assigns system-generated names to the messages. The following example shows a sample system-generated message name: *M702221.V1*

Perform Selected Action Click the button to generate services and service operations for the component interface and selected methods.

After the service and service operations are created, the Review Status page displays and you can review the actions performed as well as access the service definition created.

Return to Select CIs Click the link to return to the previous page, the Review Status page.

Inheriting Component Interface Security Permission Lists

You can specify that all or select component interface-based service operations inherit the security permission list(s) of the component interface from which they are derived. This feature allows you to bypass the process of manually assigning permission lists to the service operations after you generate them.

The Confirm Actions page (IB_CISERVICES_SEC) shown in the previous section contains an Inherit All CI Permissions box located on the upper half of the page. The Choose Operation grid of the page features an Inherit CI Permissions box for each service operation to be generated.

Important! Permission lists granted to a component interface after the service operation is created are not propagated to the service operations and you must manually add them.

Inherit All CI Permissions Select the box for all service operations to inherit the permission lists of the component interface from which they are based. By default the box is selected.

Clear the box to choose select service operations to inherit the permission list(s) of the base component interface or to manually define permission lists for the service operations in the PeopleTools Security component.

Inherit CI Permissions Select the box for each service operation that is to inherit the permission list(s) of the base component interface.

Clear the box for each service operation this is not to inherit the permission list(s) of the base component.

The system generates service operations for the Update and UpdateData methods based the existence of the Get method. Therefore, the same permissions granted to a component interface's Get method are inherited by service operations based on the Update and UpdateData methods

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

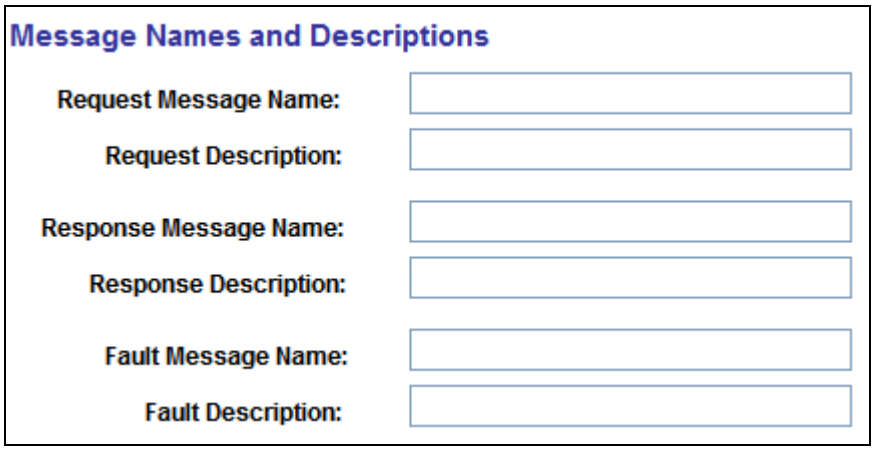
Chapter 15, "Creating Component Interface-Based Services," Generating Services and Service Operations from Component Interface Methods, page 341

Chapter 10, "Managing Service Operations," Setting Permissions to Service Operations, page 260

Adding Message Names and Descriptions to Generated Service Operations

The Confirm Actions page features a Message Names and Descriptions link for each service operation. Click the link to open the Message Names and Descriptions page (IBCISERVICES2_SEC). On the page you can enter names for each message in the service operations. You can also enter a description for each message name. The message names and description will appear in the generated XML message schema for the service operation.

The following example shows the Message Names and Descriptions page:



Message Names and Descriptions page

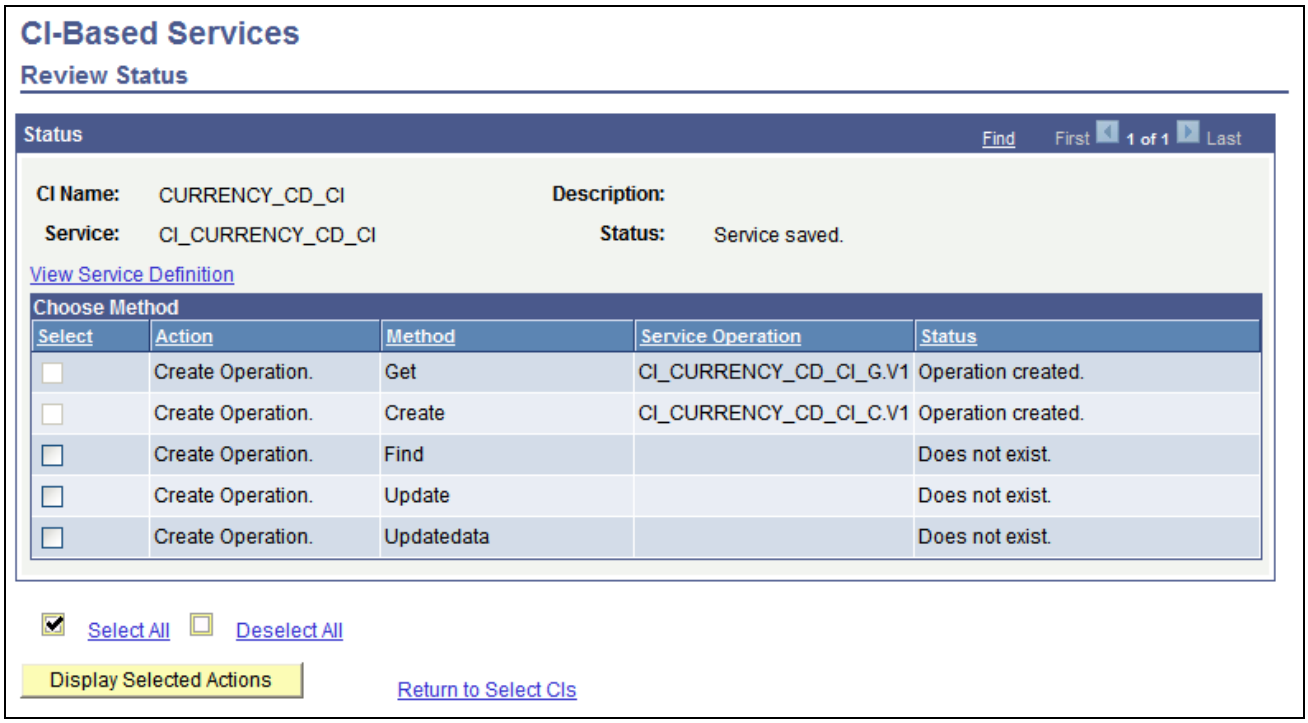
In the message name fields, enter a message name. You do not need to append the message name with a version number. The system automatically creates versions the message as version one, and appends the message name with *.V1*.

When you have completed entering message names and descriptions, click the OK button to return to the Confirm Actions page.

Viewing Component Interface-Based Service Definitions

After you generate services and service operations, the Review Status page (IB_SERVICES2) appears as shown in the following example:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Review Status page for generated service and service operations

Use this page to review the actions performed. For example, the previous graphic shows the names of the two service operations created as well as the service operation status of *Operation created.*

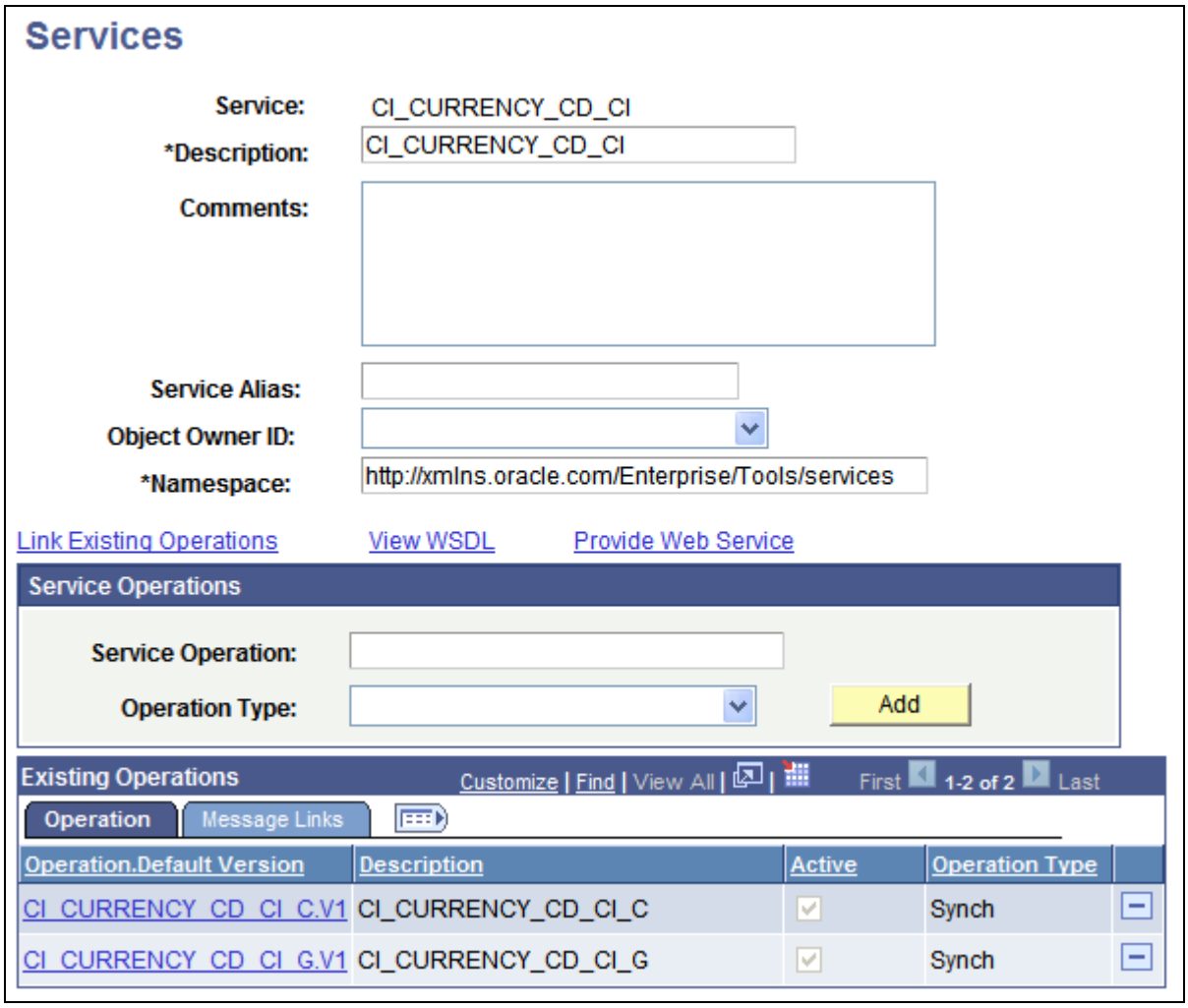
From this page you can continue to create additional service operations using the remaining available methods for the component interface.

Or, you can click the Return to Select CI link to return to the Select Component Interface page to select new component interfaces for which to generate services and service operations.

You can also click the View Service Definition link to view the service definition for the service created.

When you click the View Service Definition link the service you created, *CI_CURRENCY_CD_CI* appears in the Services page as shown in the following graphic:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Services page for the CI_CURRENCY_CD_CI service

From this page you can perform actions as you would on any other service, including:

- Click the Provide Web Service link to generate WSDL for the service.
- Use the Service Operations box to add additional service operations to the service.
- Click either of the operations that display in the Existing Operations box to generate routing definitions, view the response, request or fault messages, view handler details, and more.
- And so on.

See Also

Chapter 9, "Managing Services," page 223

Chapter 10, "Managing Service Operations," page 247

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 16

Managing Service Operation Handlers

This chapter provides an overview of service operation handlers and handler implementation methods, and discusses how to:

- Add handlers to service operations.
- Implement handlers using application classes.
- Implement handlers using application engine programs.
- Implement handlers using component interfaces.
- Implement handlers using the bulk load handler.
- Implement handlers using the deprecated PeopleCode handler.

Understanding Service Operation Handlers

Messaging handlers, or *handlers* are associated with a service operation on the Handlers tab of the Service Operations page.

Handlers define additional programming to be used with processing the message associated with the service operation.

Service Operation Handler Types

The following are the different types of handlers:

- On Notify
- On Pre-notify
- OnPost-notify
- On Receive
- On Request
- On Response
- On Route
- On Send

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Handler Types and Messaging Types

The availability of each handler type depends on the type of service operation you are using. The following tables list the message types used for each service operation type and the handler types available for use.

Table 1: Handler Types and Messaging Types

Service Operation Type	*On Notify Handler Type*	*On Pre-Notify Handler Type*	*On Post-Notify Handler Type*	*On Receive Handler Type*
Asynchronous one-way	Request message	Request message	Request message	Request message
Asynchronous request / response	Request message	NA	NA	Request message
Asynchronous to synchronous	Request message	NA	NA	NA
Synchronous	NA	NA	NA	NA

Table 2: Handler Types and Messaging Types (cont.)

Service Operation Type	*On Request Handler Type*	*On Response Handler Type*	*On Route Handler Type*	*On Send Handler Type*
Asynchronous one-way	NA	NA	Request message	Request message
Asynchronous request / response	NA	Response message	*Request message *Response message	Request message
Asynchronous to synchronous	Request message	Response message	*Request message *Response message	Request message
Synchronous	Request message	NA	Request message	Request message

Note. For On Route with On Send, the message structure is a request message. For On Route with On Receive, the message structure is a response message.

The On Response handler type is used to identify the type of On Notify event to be fired. For example, assume there are four On Notify handlers that are to be fired—three are general On Notify events, that is, the message is processed as part of the request, and the fourth is a response to the original asynchronous request. The fourth one is specified with a handler type of On Response, and the application class selected is the base class OnNotify.

Table 3: Handler Types and Messaging Types (cont.)

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

REST Provider HTTP Method	*On Request Handler Type*	*On Response Handler Type*	*On Route Handler Type*	*On Send Handler Type*
Delete	NA	NA	NA	NA
Get	NA	NA	NA	NA
Head	NA	NA	NA	NA
Post	Request message	NA	NA	NA
Put	Request message	NA	NA	NA

There are no handlers for REST consumer service operations. You may add connector properties to consumer service operations on the routing definition using the Routings-Connector Properties page or add them dynamically by using PeopleCode.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining and Overriding Gateway and Connector Properties, page 393 and Chapter 7, "Sending and Receiving Messages," Using PeopleCode to Manage REST Service Operations, page 205.

See Also

Chapter 10, "Managing Service Operations," Adding Handlers to Service Operations, page 259

Understanding Implementing Handlers

You can implement handlers using application classes, application engine programs, component interfaces, data mover scripts or pre-PeopleTools 8.48 integration PeopleCode constructs.

The following tables list the handlers you can implement using application classes, application engine programs, component interfaces and bulk-load processing.

Note. For REST provider services you may only implement the OnRequest handler using an application class.

Table 1:

Implementation	*On Notify Handler*	*On Pre-notify*	*On Post-notify*	*On Receive Handler*
Application class	Y	Y	Y	Y
Application Engine	Y	N	N	N
Component interface	Y	N	N	N
Bulk load	Y	N	N	N

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Table 2:

Implementation	*On Route Handler*	*On Send Handler*	*On Request Handler*	*On Response Handler*
Application class	Y	Y	Y	Y
Application Engine	N	N	N	N
Component interface	N	N	Y	Y
Bulk load	N	N	N	N

In addition to the implementation methods listed in the table, there is also a deprecated PeopleCode handler (DPC handler) that the system automatically creates for any integration PeopleCode that cannot be migrated to the PeopleTools 8.48 or higher systems.

The handler implementation methods listed in the tables are discussed in greater detail in subsequent sections of this chapter.

See Also

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Application Classes, page 353

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Application Engine Programs, page 356

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Component Interfaces, page 360

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Bulk Load Processing, page 361

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Deprecated PeopleCode Handlers, page 367

Adding Handlers to Service Operations

This section describes how to:

- Add a handler to a service operation.
- Set handler processing sequence.

Understanding Adding Handler Definitions to Service Operations

When you create a handler definition you supply basic information about the handler, then define information specific to the implementation method for the handler.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

You can create and use multiple handlers for a service operation. In addition you can set the sequence in which the system performs handler processing.

Handler Processing Sequence

The Sequence field enables you to select the order in which PeopleSoft Integration Broker performs handler processing. A handler with a sequence value of *1* is processed first, followed by that with a value of *2*, and so on.

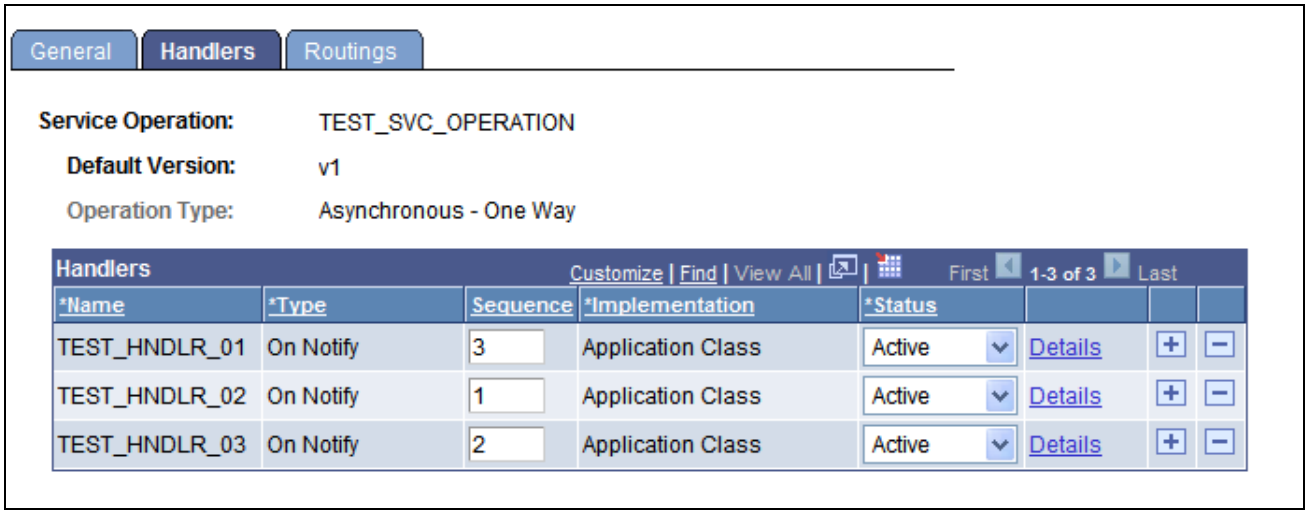
If multiple notification handlers exist and are active, but no sequence is set, the system processes handlers alphabetically.

If an error occurs in a notification, all subsequent handler processing stops until the error is corrected, the service operation is resubmitted, and the handler runs successfully.

In an error situation, the Service Operation Monitor shows a status of *Error* for the handler with the error, and all remaining handlers to be processed are in a status of *New*.

Adding a Handler to a Service Operation

The following example shows the Service Operations-Handlers page (IB_SERVICEHDLR) that you use to create handler definitions:



Creating handlers using the Service Operations — Handlers page.

To create a handler definition:

1. Access the Service Operations–Handlers page (PeopleTools, Integration Broker, Integration Setup, Service Operations. Click the Handlers tab.)
2. In the Handlers section, enter a handler name in theName field.

   Note that for OnRequest, and OnRoute handlers, you need not enter a name. The system adds a handler name after you provide the handler details.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. From the *Type* drop-down list box, select the handler type.

   The service operation type determines the handler types that are available to choose.

   See Chapter 16, "Managing Service Operation Handlers," Handler Types and Messaging Types, page 348.

   The valid values are:

4. In the Sequence field, enter a number to specify the order of processing for the handler.

   The Sequence field appears only when there are multiple notification handlers defined and active for the service operation.

5. From the Implementation drop-down list box, select the method to use to implement the handler.

   The handler type determines the available implementation methods.

   See Chapter 16, "Managing Service Operation Handlers," Understanding Implementing Handlers, page 349.

6. From the Status drop-down list box, select a status for the handler.

   Values are:

   - *Active.* (Default.) Select to make the handler active.
   - *Inactive.* Select to make the handler inactive.

7. Click the Save button.

Repeat these steps to add additional handlers to the service operation.

Specifying General Handler Details

The details you enter for a handler depend on the implementation method of the handler.

This section describes general handler detail information you must specify for implementing handlers using application classes and component interfaces.

There are additional details you must define for these implementation methods, as well as for application engine and data mover script implementation methods. See the sections for each implementation method for additional information you must specify.

Note. You do not enter handler details for handler implementations using a deprecated PeopleCode handler.

To add general handler details:

1. On the Handlers tab of the Service Operations – Handlers tab, in the Handlers section, click the Details link.

   The Action Details page appears.

2. In the Description field, enter a description for the handler. By default, the system populates this field with the name of the service operation associated with the handler.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. (Optional.) In the Comments field, enter comments about the handler.
4. (Optional.) In the Handler Owner field, enter the name of the person or group that owns or maintains the handler.

See Also

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Application Classes, page 353

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Application Engine Programs, page 356

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Component Interfaces, page 360

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Bulk Load Processing, page 361

Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Deprecated PeopleCode Handlers, page 367

Implementing Handlers Using Application Classes

This section provides an overview of implementing handlers using application classes and discusses how to:

- Develop an application class for implementing a handler.
- Specifying application class implementation details.

Understanding Implementing Handlers Using Application Classes

You can specify an application class as a handler for a service operation. This is the most typical implementation of a handler.

The process for implementing a handler using an application class is:

1. Develop an application class for implementing the handler.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Developing Application Classes for Implementing Handlers, page 354.

2. Add a handler to the service operation.

   This process is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Adding a Handler to a Service Operation, page 351.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Specify general handler details.

   This process is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Specifying General Handler Details, page 352.

4. Specify application class details for the handler.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Specifying Application Class Implementation Details, page 355.

Developing Application Classes for Implementing Handlers

This section discusses how to develop application classes for implementing service operation handlers.

After you create the application package, you must specify the application package, class and method in the service operation definition as part of the handler details.

Understanding Interfaces and Deriving Application Classes

For each handler type, the following table lists the interface from which to derive the application class. The interfaces are located in the PS_PT application package:

Handler Type	*Application Class Interface*
On Notify	INotificationHandler
On Post Notify	IPrePostNotification
On Pre Notify	IPrePostNotification
On Response	INotificationHandler
On Receive	IReceiver
On Request	IRequestHandler
On Send	ISend
On Route	IRouter

Understanding Input and Output Parameters

For application class handlers, the names that populate the drop-down used for selecting the appropriate method must have the exact signature required for the method.

Handler Type	*Input Parameter to Method*	*Method Output Parameter*
On Notify	Message	Void (none)

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Handler Type	*Input Parameter to Method*	*Method Output Parameter*
On Response	Message	Void (none)
On Receive	Message	Int
On Request	Message	Message
On Send	Message	Message
On Route*	Message	Integer or Boolean

For the On Route handler type, if you select a method that returns as integer, the handler type is On Route Send. If you select a method that returns as Boolean, the handler type is On Route Receive.

Developing an Application Class for Implementing a Handler

To develop an application class for implementing a handler:

1. Select the Integration Broker method that you want to implement based on the type of service operation you are creating.
2. Create a new application class, and import the appropriate Integration Broker application class. For example:

   ```
   import PS_PT:Integration:INotificationHandler;
   ```

3. Define a class that implements the Integration Broker application class.
4. Define the method that implements the Integration Broker method, with the appropriate signature. In the following example, the OnNotify method would be available as a handler method.

   ```
   class RESPONSE_NOTIFICATION implements PS_PT:Integration:INotificationHandler

   method RESPONSE_NOTIFICATION();
   method OnNotify(&MSG As Message);

   end-class;
   ```

5. In the definition of the class, create the program-specific code to be used for this handler.

Specifying Application Class Implementation Details

Before specifying application class implementation details for a handler, you must first create the application class to implement the handler, add a handler to the service operation definition, and specify general handler details.

To specify application class implementation details:

1. On the Service Operations – Handlers tab, in the Handlers section, click the Details link.

   The Handler Details page appears.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Complete the following fields:

Package Name	Enter the package name that contains the class that you want to specify, or use the Lookup button to search for and select one.
Path	Enter the name or names of any subpackages that contain the application class that you want to specify, or use the Lookup button to search for and select one.
Class ID	Enter the name of the application class that contains the method that you want to specify, or use the Lookup button to search for and select one. Only application classes that implement an appropriate base class are shown. The base class is dependent on the handler type.
Method	From the Method drop-down list box, select the method from the selected application class that you want to specify. Only methods with the correct signature for the current handler type are shown.

3. Click the OK button.
4. Click the Save button.

Implementing Handlers Using Application Engine Programs

This section provides and overview of implementing handlers using application engine programs and discusses how to:

- Specify application engine implementation details.
- Retrieve service operation content from application engine programs.
- View subscription contract status.

Understanding Implementing Handlers Using Application Engine Programs

Application engine handlers are available for notification events (OnNotify) and schedule an application engine program to run within PeopleSoft Process Scheduler to process service operation content. This handler is a good alternative for long-running programs or service operations that have large amounts of data to process, since processing is performed by PeopleSoft Process Scheduler and not in the PeopleSoft Integration Broker runtime environment.

After the service operation is invoked, you can track the process by viewing the subscription contract in the Service Operations Monitor.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When you implement a handler using an application engine program, you must retrieve the transaction ID that you use to get message data from within the application engine program. The run control record PSIBRUNCNTL is populated with the transaction ID prior to scheduling the actual application engine program. Note that it is not required to retrieve the transaction ID, therefore any application engine program can be scheduled that does not require initialization parameters via a run control table.

The process for implementing a handler using and application engine program is:

1. Develop the application engine program.

   Developing application engine programs is discussed elsewhere in PeopleBooks.

   See *PeopleTools 8.52: Application Engine*, "Creating Application Engine Programs."

2. Add a handler to the service operation.

   This process is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Adding a Handler to a Service Operation, page 351.

3. Specify general handler details.

   This process is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Specifying General Handler Details, page 352.

4. Specify application engine handler implementation details.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Specifying Application Engine Handler Implementation Details, page 357.

5. Retrieve service operation content from the application engine program.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Retrieving Service Operation Content from Application Engine Programs, page 358.

6. View subscription contract status.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Viewing Subscription Contract Status, page 359.

Specifying Application Engine Handler Implementation Details

Before specifying application engine handler implementation details, you must first create the application engine program to process the service operation, add a handler to the service operation definition, and specify general handler details.

To specify application class implementation details:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. On the Service Operations – Handlers tab, in the Handlers section, click the Details link.

   The Action Details page appears.

2. In the Program Name field, click the Lookup button to locate the name of the application engine program to use.

3. Click the OK button.

4. Click the Save button.

Retrieving Service Operation Content from Application Engine Programs

This section discusses how to:

- Retrieve a transaction ID from an application engine program.
- Instantiate a Message object using a transaction ID.

Understanding Retrieving Service Operation Content from Application Engine Programs

Within the application engine program, you can retrieve the transaction ID and thereby get an instantiated Message object to retrieve the content data respectively.

Retrieving Transaction IDs from Application Engine Programs

Use the GetIBTransactionIDforAE method on the %IntBroker object to get the transaction ID or simply write the SQL to retrieve the ID.

The following examples show how to get the transaction ID from an application engine program and instantiate a Message object.

Method from %IntBroker Object:

```
StrTransaction = %IntBroker.GetIBTransactionIDforAE
(string OperID, string Run Control ID);

&MSG = %IntBroker.GetMessage(StrTransaction, %IntBroker_SUB);
```

Or, you can write code within a SQL step similar to the following:

```
%SelectInit(IBTRANSACTIONID)
 SELECT IBTRANSACTIONID
  FROM PSIBRUNCNTL
 WHERE OPRID = %Bind(OPRID)
AND RUN_CNTL_ID = %Bind(RUN_CNTL_ID)
```

Or, you can perform a simple SQLExec as follows:

```
SQLExec("select ibtransactionid from psibruncntl where oprid = :1 and
 run_cntl_id = :2",

j_book_aet.oprid, j_book_aet.run_cntl_id, &ibtransid);
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Instantiating Message Objects Using Transaction IDs

The following code example shows how to instantiate a Message object using the transaction ID from an application engine program:

```
&MSG = %IntBroker.GetMessage(J_BOOK_AET.IBTRANSACTIONID, %IntBroker_SUB);
```

Viewing Subscription Contract Status

The following table lists the possible contract statuses that may display in the Service Operations Monitor when implementing a handler using an application engine program:

Service Operations Monitor Status	*Description*
Working	At runtime, the Service Operations Monitor shows a status of *Working* for the subscription contract when it is picked up by the subscription contract handler.
Submit	After the application engine program is scheduled successfully, the state of contracts changes to *Submit* in the Service Operations Monitor. This status indicates that the application engine program has been successfully submitted and a process instance has been returned.
Error	• If the program does not get scheduled successfully, the status is set to *Error* • A Process Monitor status of *Error* will change the subscription contract status in the Service Operations Monitor to Error. You must read the error details from the Process Monitor Details page, since the Service Operations Monitor does not have access to the Process Monitor error tables.
Done	When the application engine program has been processed successfully, the status of the contract in the Service Operations Monitor is *Done.*
Cancel	• For application engine programs that have Process Monitor status of *Cancel* or *Delete,* the status of the subscription contract in the Service Operations Monitor is changed to *Cancel.* • If the instance is deleted from the process scheduler table then the status of the subscription contract is *Cancel.* This will clear out the contract in the partitioned queue so that the next contract in sequence can be processed.

Note. In the Service Operations Monitor the *Submit* status has the same behavior as the *Working* status, thus proper queue partitioning behavior is maintained. Therefore, intelligent queue partitioning should be taken into account for service operations using the application engine handler.

A Subscription Contract processing an application engine handler cannot be cancelled or resubmitted from the Service Operations Monitor, but rather must be done in the Process Monitor. Once you resubmit or cancel the program in the Process Monitor, the status is updated for the subscription contract. Refresh the Service Operations Monitor page to view the updated status. Note that there could be a delay from when the time you cancel or resubmit the application engine program in the Process Monitor and when the update is seen in the Service Operations Monitor. In addition, at least one Pub/Sub domain must be active for this status update to happen.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Implementing Handlers Using Component Interfaces

You can implement a component interface as a handler to access extant business rules and processes to be used with the service operation data.

Understanding Implementing Handlers Using Component Interfaces

This section provides an overview of implementing handlers using component interfaces.

You can only use a component interface for On Notify, On Request and On Response handler types.

Component interfaces can be used as handlers for asynchronous and synchronous service operations. Asynchronous service operations should only be used with component interface methods that do not require a response, such as Update or Insert.

You must define a component interface as a service before you can implement any of the component interface methods as handler actions.

See Chapter 15, "Creating Component Interface-Based Services," page 335.

The standard methods for a component interface (such as Find, Get, Save, Update, and so on) are automatically available for a handler. However, if you want to use a user-defined method, you must include the keyword **Doc** as part of the signature.

The process for implementing a handler using a component interface is:

1. Define the component interface as a service.

   This procedure is discussed elsewhere in this chapter.

   See Chapter 15, "Creating Component Interface-Based Services," page 335.

2. Add a handler to the service operation.

   This procedure is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Adding a Handler to a Service Operation, page 351.

3. Specify general handler details.

   This procedure is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Specifying General Handler Details, page 352.

4. Specify component interface handler implementation details.

   This process is discussed in this section.

   See Chapter 16, "Managing Service Operation Handlers," Specifying Component Interface Handler Implementation Details, page 361.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Specifying Component Interface Handler Implementation Details

Before specifying component interface handler implementation details, you must first create a service from the component interface, add a handler to the service operation definition, and specify general handler details.

To specify component interface implementation details:

1. On the Service Operations – Handlers tab, in the Handlers section, click the Details link.

   The Action Details page appears.

2. In the Name field enter the component interface name, or use the Lookup button to search for and select one.

3. From the Action drop-down list box, select an action.

4. Click the OK button.

5. Click the Save button.

Implementing Handlers Using Bulk Load Processing

This section provides and overview of implementing handlers using the bulk load processing and discusses how to:

- Enable transactional rollback.
- Enable table truncation.
- Specify XML attributes for nonrowset-based messages.
- Add data structures to nonrowset-based messages.

Understanding Implementing Handlers Using the Bulk Load Handler

You can bulk load large amounts of data (large messages) into a local table using the bulk load handler.

Note. Bulk load handlers are allowed only on asynchronous service operations and only one bulk load handler is allowed per service operation.

When you implement a handler using a data mover script use the OnNotify event. When the OnNotify event is fired, the message data is inserted into the tables. Bulk load handlers do not perform any data validation.

Bulk Load Handler Message Types

You can use bulk load handlers for service operations that contain the following types of asynchronous messages:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Rowset-based messages.

  For service operations that contain rowset-based messages, the message structure determines processing.

- Nonrowset-based messages.

  For service operations that contain nonrowset-based messages, you define the structure for processing by mapping data to PeopleSoft records and providing attribute names and values.

- Container messages that contain rowset-based messages parts.

  For service operations with container messages that contain rowset-based message parts, the bulk load handler uses the message structure of the message parts to determine processing.

Transactional Rollback

Bulk load handler processing is *destructive load processing,* meaning that the handler deletes existing data from the table and then bulk inserts the new data.

PeopleSoft Integration Broker provides an optional transaction rollback feature that enables you to rollback the database to the pre-bulk insert state if needed.

Transactional rollback is dependent on the bulk size of the data to be updated. Therefore, when you select the rollback option, a Rollback Size field appears where you specify a rollback value. The default value is *32* kilobytes.

Table Truncation

By default, the bulk load handler truncates table data before performing any processing.

PeopleSoft Integration Broker enables you to disable table truncation.

Nonrowset-Based Messages and the Bulk Load Handler

If a service operation contains a nonrowset-based message, you must add a data structure to define the message content, thereby allowing non-PeopleSoft data to map to the records and fields in the database.

In addition, you can provide an XML attribute name/value pair to distinguish record and field nodes on incoming XML content.

Process Overview for Implementing Handlers Using Bulk Load Handlers

The process for implementing a bulk load handler is:

1. Add a bulk load handler to the service operation on the target system.

   This procedure is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Adding a Handler to a Service Operation, page 351.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Specify general handler details.

   This procedure is discussed elsewhere in this chapter.

   See Chapter 16, "Managing Service Operation Handlers," Specifying General Handler Details, page 352.

3. (Optional.) Enable transactional rollback and the rollback size.

   This procedure is discussed in this section.

4. (Optional.) Specify XML record attributes.

   This procedure is discussed in this section.

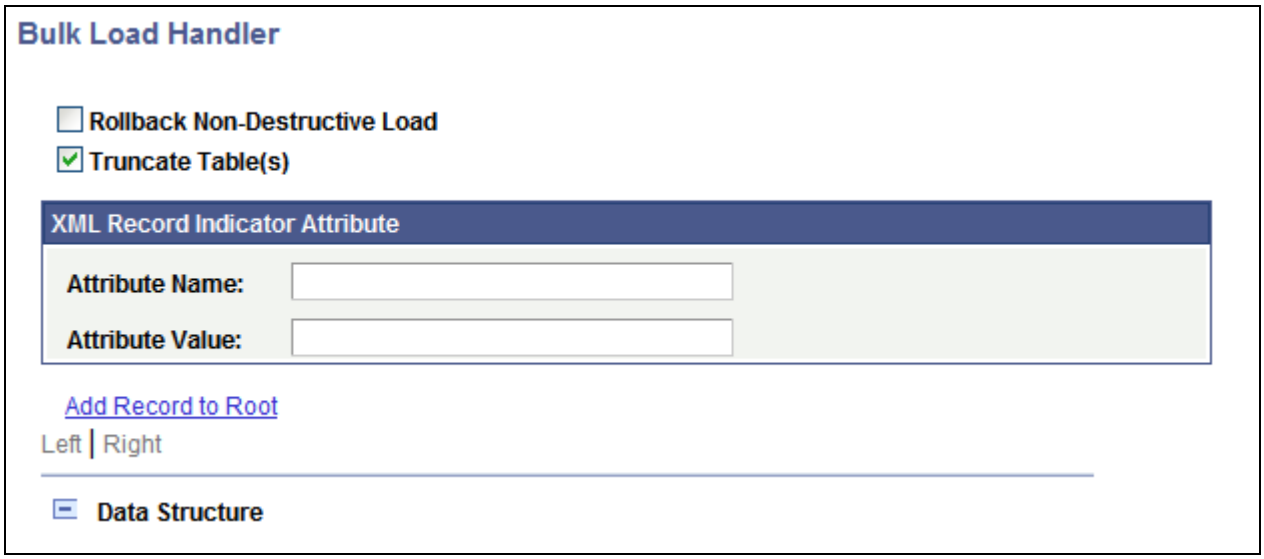
5. Add the data structure for the message contained in the service operation.

   The step is required only when a nonrowset-based message is contained in the service operation.

   This procedure is discussed in this section.

Enabling Transactional Rollback

The Bulk Load Handler page (IB_SERVICEHDL5_SEC) features a Rollback Non-Destructive Load option that you can set to enable transaction rollback. The Bulk Load Handler page is shown in the following example:

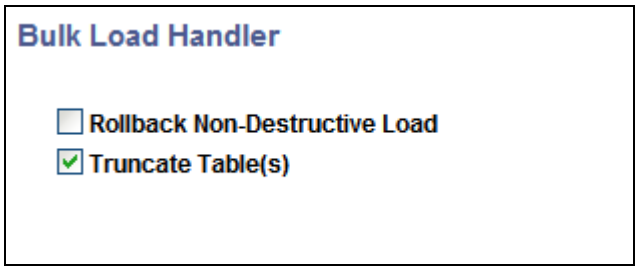


Bulk Load Handler page when the service operation contains a nonrowset-based message.

The previous example shows the Bulk Load Handler page that displays when the service operation contains a nonrowset-based message.

If the service operation contains a rowset-based message, the page does not display the XML attribute fields or the data tree structure controls. The following example shows the Bulk Load Handler page when the service operation contains a rowset-based message:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Bulk Load Handler page when the service operation contains a rowset-based message.

Note. Transactional rollback is not supported if message segments are used. Transactional rollback will automatically be disabled if the message being processed contains more then one message segment.

To enable rollback processing:

1. On the Service Operations – Handlers tab, in the Handlers section, click the Details link.

   The Bulk Load Handler page appears.

2. Check the Rollback Non-Destructive Load box.

   The Rollback Size field appears.

3. In the Rollback Size field, enter a rollback size value, in kilobytes.

   The amount of data that you can rollback is depends on your database.

   The default is *32*.

   Warning! Before you specify a rollback size value, verify the rollback size configured for your database. If the rollback size value specified in the PeopleSoft Pure Internet Architecture is greater then the transactional rollback size configured on the database, then transactional rollback will be disabled for that particular transaction, thereby causing destructive load processing to take place. To avoid destructive load processing, modify the rollback size configured for your database so that it is greater than the value you specify on the Bulk Load Handler page.

   ---

4. Click the Save button.

Enabling Table Truncation

The Bulk Load Handler page features a Truncate Table(s) option that you can set so that the system truncates the database table(s) prior to handler processing. The option is located under the Non-Destructive Load option on the Bulk Load Handler page.

By default, the Truncate Table(s) option is enabled and the PeopleSoft system performs table truncation. However, if you do not wish to perform table truncation you can clear the option to disable table truncation.

To enable/disable table truncation:

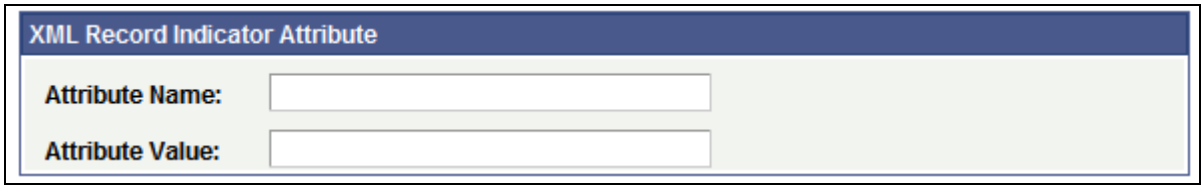
1. Access the Bulk Load Handler page (PeopleTools, Integration Broker, Integration Setup, Service Operations. Click the Handlers tab. Click the Details link for the bulk load handler.)

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. On the Truncate Table(s) box perform one of the following options:
   - To enable table truncation, check the box.
   - To disable table truncation, clear the box.
3. Click the OK button.

Specifying XML Record Attribute Values

Use the XML Record Indicator Attribute box on the Bulk Load Handler page, shown in the following example, to distinguish and specify records and fields on incoming XML content of nonrowset-based messages.



Fields used to specify an attribute name and value for nonrowset-based messages.

When you specify an attribute name and value, the system recognizes content with that name and value as a record. Content placed in the message after that attribute name/value pair is considered child data.

When the service operation contains a nonrowset-based message, you must specify an attribute name and attribute value. If you do not supply values for these fields, an error message appears. These fields are optional when the service operation contains a rowset-based message.

To specify XML record attribute values:

1. Access the Bulk Load Handler page (PeopleTools, Integration Broker, Integration Setup, Service Operations. Click the Handlers tab. Click the Details link for the bulk load handler.)
2. In the Attribute Name field, enter the attribute name that the system should recognize as the record.

   For example, *Class.*
3. In the Attribute Value field, enter the value for the attribute name.

   For example, *R.*
4. Click the Save button.

Adding Data Structures for Nonrowset-Based Messages

This section discusses how to:

- Build data structures for nonrowset-based data structures.
- Insert nonrowset-based messages in project definitions.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Building Nonrowset-Based Data Structures

The Bulk Load Handler details page provides a data tree, similar to that found in the Messages component, that you use to build the structure of the nonrowset-based message.

You must define a minimum of one record to save the handler definition.

For building the data structure you use the same graphical user interface and procedures that you would for managing rowset-based messages in the Messages component.

By clicking the Add Record to Root link you can begin to create the data structure by adding a root record. As with the Messages component, you can select to include or exclude fields and specify field aliases. One unique difference here, however, is that you can build structures with multiple level 0 records. At save time, the system builds a message for each level 0 record found in the structure.

Note that message references are not allowed.

Message structures you create on the Bulk Load Handler details page are not accessible in the Messages component and appear only as part of the bulk load handler.

The message names start with *~IB_M_DMS~*.

See the documentation elsewhere in this PeopleBook for managing rowset-based messages for creating the data structure. Again, even though here you are building the structure of a nonrowset-based message, you will use the same graphical user interface and procedures as those for working with rowset-based messages.

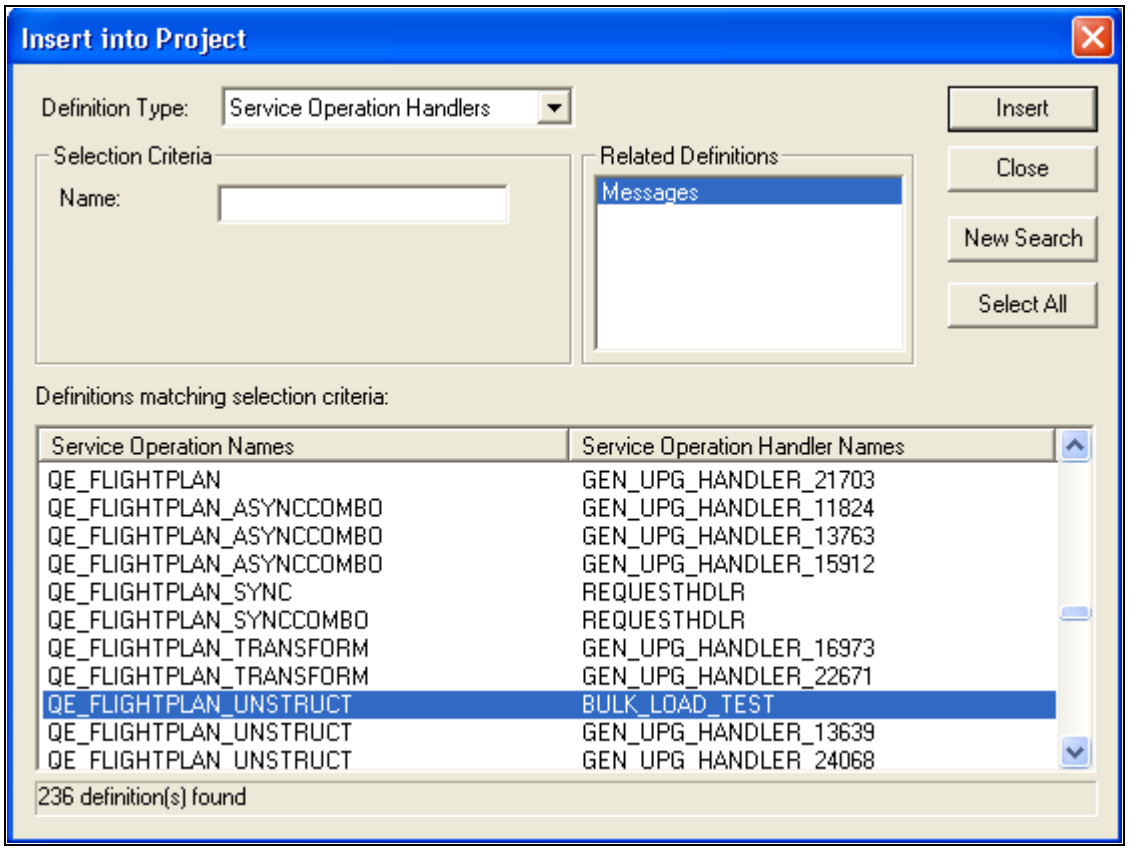
See Chapter 6, "Managing Messages," Managing Rowset-Based Messages, page 87.

Inserting Nonrowset–Based Messages in Project Definitions

You must assign nonrowset-based messages you create using the Bulk Load Handler page to a bulk load handler. To accomplish this task, in PeopleSoft Application Designer when you insert a service operation handler definition into a project, you must include the message(s) as a related definition.

The following example shows the Insert into Project page used to insert a service operation handler definition and related message definitions into a project:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Inserting a bulk load service operation handler and related message definitions into a project.

To insert a service operation handler definition and related message definitions into a project:

1. In PeopleSoft Application Designer, from the Insert menu, select Definitions info Project.

   The Insert into Project page appears.
2. From the Definition Type drop-down list box, select Service Operation Handlers.
3. Click Insert to view all service operation definitions and handlers in the system.
4. Click the name of the service operation and handler to insert into the project.
5. In the Related Definition section, click Messages.
6. Click the Insert button to insert the service operation, handler, and related messages into the project.

Implementing Handlers Using Deprecated PeopleCode Handlers

This section discusses the deprecated PeopleCode handler and using the Service Administration – Deprecated PeopleCode page to delete deprecated PeopleCode handlers.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

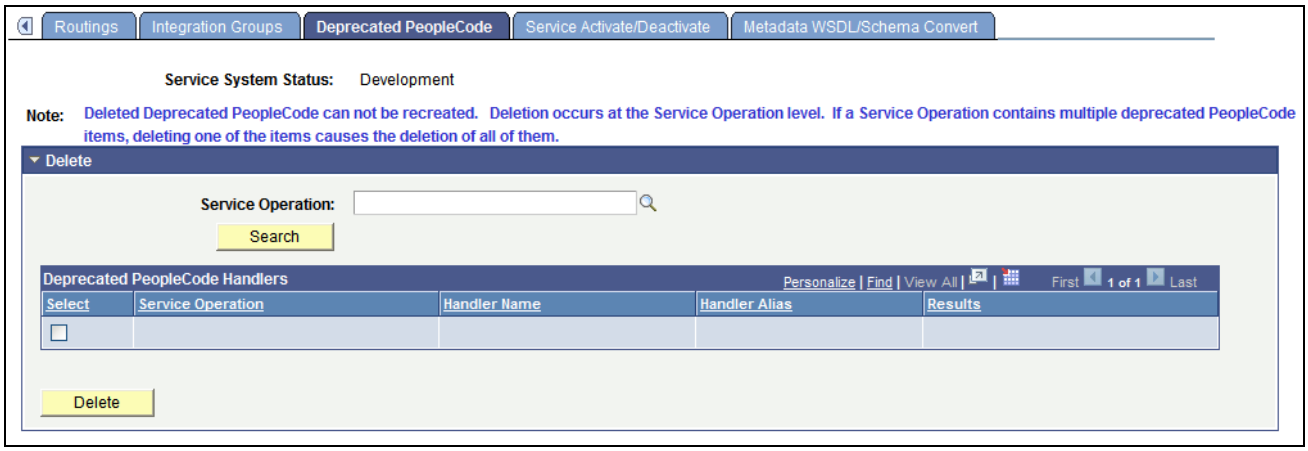
Understanding the Deprecated PeopleCode Handler

During the upgrade process, the PeopleSoft system creates a deprecated PeopleCode handler (DPC handler) for any integration PeopleCode that cannot be migrated from a PeopleTools 8.47 or earlier system to a PeopleTools 8.52 system. Deprecated handlers enable you to run PeopleTools 8.47 and earlier PeopleCode (subscription or OnRequest). However, PeopleSoft recommends that you correct the PeopleCode and migrate the code into an Application Class type handler for use in PeopleTools 8.52 systems.

Deleting Deprecated PeopleCode Handlers

Use the Service Administration – Deprecated PeopleCode page (IB_HOME_PAGE8) to delete deprecated PeopleCode and its associated handler(s).

To access the page select PeopleTools, Integration Broker, Service Utilities, Service Administration and click the Deprecated PeopleCode tab. The following example shows the page:



Service Administration – Deprecated PeopleCode page

The page enables you to delete the PeopleCode that was assigned to the Pre-PeopleTools 8.48 subscription, along with the (deprecated) handler that was created to invoke the PeopleCode.

Warning! Deleted deprecated PeopleCode cannot be recreated.

To delete a deprecated PeopleCode handler:

1. Access the Service Administration – Deprecated PeopleCode page (PeopleTools, Integration Broker, Service Utilities, Service Administration and click the Deprecated PeopleCode tab).
2. In the Service Operation field, enter the name of the service operation that contains the deprecated PeopleCode handler, or click the Lookup button to search for one.
3. Click the Search button.

   The search results appear in the Deprecated PeopleCode Handler grid.
4. Check the Select box next to the handler to delete.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Click the Delete button.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 17

Managing Service Operation Routing Definitions

This chapter discusses how to:

- Manage system-generated routing definitions.
- Add routing definitions.
- Configure routing definitions.
- Use introspection to create routing definitions.
- Activate and inactivate routing definitions.
- View routing definitions in graphical format.
- View integration status and activate integration metadata.
- Retrieve routing properties programmatically.
- Search for duplicate routing aliases.
- Rename and delete routing definitions.
- Delete duplicate routing definitions.

Understanding Routing Definitions

This section provides overview information about routing definitions.

Routing Definitions

A routing definition defines the sending and receiving nodes for a transaction, specifies any inbound and outbound transformations to invoke and defines external aliases. It also defines overrides that the default integration gateway and the default target connector that the local node use to communicate with an integration endpoint.

Routing Types

There are four routing types:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Any-to-local	An any-to-local routing enables the local node to receive transactions from any node. This routing type is for inbound transactions only and the "any" node is always the sending node. You can use this routing type for all service operation types, except for asynchronous-to-synchronous service operations.
Local-to-local	A local-to-local routing is a routing in which transactions are sent and received within the local database. When working with synchronous service operations, you have the option to generate transactional routings, whereby the OnRequest event will run under the existing transaction boundary of the component.
Local-to-Atom	A local-to-Atom routing is used exclusively with feeds functionality. See *PeopleTools 8.52: Feeds Publishing Framework*
Point-to-point	A point-to-point routing requires that specific nodes that you define send and receive a transaction.

Defining Routing Definitions

A routing definition must exist on each system that is participating in a particular transaction.

For example, let's say that System A is going to send a service operation to System B and a point-to-point routing needs to be created between the two systems. Further, the local node on System A is called *Node A,* and the local node on System B is called *Node B.*

System A and System B need to have the identical service operation defined on each of their systems.

In addition, System A needs to have an outbound routing definition defined on its system that specifies its local node, *Node A,* as the sending node. The routing definition must specify System B's local node, *Node B,* as the receiving node.

System B needs to have an inbound routing definition defined on its system that specifies its local node, *Node B,* as the receiving node. The routing definition on System B also needs to specify System A's local node, *Node A,* as the sending node.

The exception to this is when a receiving system generates an any-to-local routing for a service operation. If the receiving system has an any-to-local routing definition defined for a particular service operation, it will accept requests from any node without the need of a specific point-to-point routing definition.

So using the examples of System A being the sending system and System B being the receiving system, this is what happens when an any-to-local routing is defined. System A still needs to have a routing definition defined on its system where its local node, *Node A,* is defined as the sending node, and System's B local node, *Node B,* is defined as the receiving node. On System B, when the any-to-local routing was generated, the PeopleSoft system automatically populated System B's local node, *Node B,* as the receiving node and listed the value of *~Any~* as the sending node to designate that the system will accept the service operation specified on the routing from any node.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Methods for Generating and Defining Routing Definitions

This section discusses methods of generating and defining routing definitions.

System-Generated Routing Definitions During Upgrade

If upgrading from a PeopleTools 8.47 or earlier release, during the upgrade process the PeopleSoft system creates routing definitions from node transaction and relationship data defined in your earlier PeopleTools 8.4x release.

System-Generated Routing Definitions During Consuming Services

When you consume a service using the Consume Web Service wizard, the system creates PeopleSoft integration metadata for the imported service, including routing definitions.

System-Generated Routing Definitions at Runtime

The PeopleSoft system can generate any-to-local and local-to-local routing definition for you. The system takes integration metadata from the service operation, including service operation name, service operation version, service operation type, local node information, and other data, and generates a routing definition.

If you make any subsequent modifications to the service operation, you can regenerate the routing definition to reflect the changes. In addition, at any point you can open the definition and manually modify it to include transformations, as well as override default integration gateway and target connector settings.

You use the Service Operations-General page to generate any-to-local and local-to-local routing definitions.

You may also manually create local-to-local routing definitions. However, you must always system-generate any-to-local routing definition.

See Chapter 17, "Managing Service Operation Routing Definitions," Managing System-Generated Routing Definitions, page 376.

User-Defined Routing Definitions

When you require a point-to-point routing, you may manually create the routing definition or generate it using introspection.

You can also manually create local-to-local routing definitions. However, you must always system generate any-to-local routing definitions.

To create user-defined routing definitions, use the Routings component or the Service Operations-Routings page.

See Chapter 17, "Managing Service Operation Routing Definitions," Adding Routing Definitions, page 379.

Introspecting Nodes To Generate Routing Definitions

PeopleSoft provides the ability to introspect other PeopleTools 8.52 nodes (PeopleSoft and external nodes) to generate point-to-point routing definitions.

You use the Deployment and Validation component to introspect nodes.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 17, "Managing Service Operation Routing Definitions," Using Introspection to Create Routing Definitions, page 396.

Summary of Methods for Creating and Generating Routing Definitions

The following table summarizes the method for generating and defining routing definitions:

Routing Type	*System-Generated*	*User-Defined*	*Introspection*
Any-to-local	Yes	No	No
Local-to-local	Yes	Yes	No
Point-to-point	No	Yes	Yes

Routing Definition Naming Conventions

The following table lists the naming conventions for routing definitions:

Method for Generating and Creating Routing Definitions	*Convention*	*Description*
System-generated during upgrade.	~GEN_UPG~<unique number> For example: *~GEN_UPG~10062*	Routing definitions generated during the upgrade process. These may be any-to-local, local-to-local, or point-to-point routing definitions.
Routing definitions generated or created by: • System-generated at runtime • Node introspection.	~GENERATED~<unique number> For example: *~GENERATED~15312*	Routing definitions generated by the PeopleSoft system from the Service Operations-General tab or from the Deployment Validation component. When generated from the Service Operations-General tab, these routing definitions are any-to-local or local-to-local. When generated from the Deployment Validation component, routing definitions are usually point-to-point definition, but may also be local-to-local routing definitions.
System generated by the Consume Web Service wizard.	~IMPORTED~<unique number> For example:*~IMPORTED~14857*	Routing definitions generated using the Consume Web Service wizard.
User-defined.	Up to 30 characters, no spaces. For example: *QE_ROUTING*. No special characters, such as dots (".") and slashes ("/" or "\"), are permitted.	Manually created point-to-point routing definitions and local-to-local routings. You can also rename system-generated routing definitions or introspected routing definitions using the Service Administration component.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Routing Definition External Aliases

When working with routing definitions you have the option of creating a routing alias. This alias is used as a SOAPAction attribute in the WSDL binding to identify the service operation in the Integration Broker metadata.

The routing external alias defaults to *<ServiceOperationAlias>.<Version>,* if present. Otherwise it defaults to *<ServiceOperation>.<Version>*.

In an asynchronous request/response any-to-local routing, the outbound routing alias format is *<Alias Name>_CALLBACK.<Version>*.

For inbound transactions you can fire multiple service operations for one invocation when external aliases on the routing definition are the same for each service operation. This is called service operation mapping.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Searching for Duplicate External Routing Aliases, page 410

Chapter 17, "Managing Service Operation Routing Definitions," Service Operation Mapping, page 375

Service Operation Mapping

You can map inbound asynchronous transactions to one or more service operations by specifying the name of the inbound transaction as the external alias on the routing for each service operation that you want to invoke.

Note. Service operation mapping is supported for inbound asynchronous transactions only.

For example, there is an inbound asynchronous transaction from SAP called *Customer_SAP*. However, the service operation contained in that transaction maps to two service operations on the PeopleSoft system, *Customer_Get* and *Customer_Update.* To invoke both service operations, change the external alias name on the inbound routing definition for the *Customer_Get* and *Customer_Update* service operations to *Customer_SAP*. When the routings are determined at runtime for this external service operation name, PeopleSoft Integration Broker will find both service operations *(Customer_Get* and *Customer_Update)* and process them accordingly.

Graphical Routings View

The Routing Definitions page provides a Graphical View link that enables you to view routing definitions in graphical format. The graphical routings view is intended to provide you with a view of the flow of data between integration partners.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Viewing Routing Definitions in Graphical Format, page 404

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Integration Status

When you use the Graphical View link to view a routing definition in graphical format, an Integration Not Active link displays if any metadata associated with the integration is inactive. Inactive metadata might include the routing definition, the service operation, a service operation handler, and so on.

If you click the Integration Not Active link an Integration Status page appears and you can view activate the metadata.

Managing System-Generated Routing Definitions

This section discusses how to:

- View system-generated routing definition status.
- Initiate system-generated routing definitions.
- Regenerate system-generated routing definitions.

Understanding Managing System-Generated Routing Definitions

The PeopleSoft system can automatically generate any-to-local and local-to-local routing definitions when you save a service operation definition.

After the system generates the routing definition, you can view and fine-tune the definition as needed using the pages in the Routings component.

In addition, if you make any changes to a service operation after the system has generated a routing definition for it, you can have the system regenerate a routing definition. However, any modifications made to the routing definition are lost when you regenerate it.

Viewing System-Generated Routing Definition Status

The Service Operations-General page (IB_SERVICE) features a Routing Status box that you can use to verify if any system-generated routing definitions exist for a service operation. To access the page select PeopleTools, Integration Broker, Integration Setup, Service Operations.

The following example shows the Routing Status box:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operations-General page Routing Status box

When an any-to-local, local-to-local, or local-to Atom routing definition exists for the service operation, the corresponding field displays a status of *Exists.* When no routing definition exists, the corresponding field displays *Does not exist.*

Note. The system generates a Local-to-Atom routing definition when you publish a service operation as a feed. Using and managing feeds is described elsewhere in PeopleBooks.

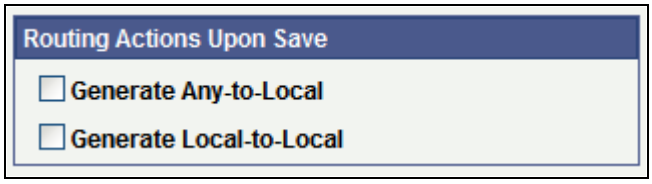
See *PeopleTools 8.52: Feeds Publishing Framework PeopleBook*

Initiating System-Generated Routing Definitions

The Default Service Operation Version section of the Service Operations-General page features a Routing Actions Upon Save box where you can choose the type of routing to generate, any-to-local or local-to-local.

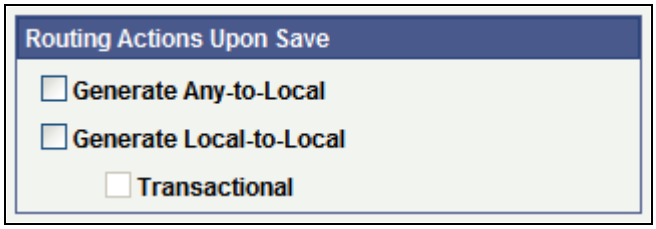
To access the Service Operations-General page select PeopleTools, Integration Broker, Integration Setup, Service Operations.

The following example shows the Routing Actions Upon Save box that appears when working with asynchronous service operation types:



Routing Actions Upon Save box for asynchronous service operations.

The following example shows the Routing Actions Upon Save box that appears when working with a synchronous service operation type:



The Routing Actions Upon Save box for synchronous service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When you select Generate Local-to-Local, the Transactional box becomes available for selection. By choosing the Transactional check box, the system routes the service operations using the OnRequest event on a single transaction.

When you initiate system-generated any-to-local or local-to-local routings, PeopleSoft Integration Broker checks to see if the routing you are initiating is already in the system. This situation can arise when any-to-local and local-to-local routings are created in another database and are imported into the current database. If the routing already exists in the current database, a message appears indicating so and no new routing is generated. You must remove the routing before generating a new one.

See Chapter 17, "Managing Service Operation Routing Definitions," Deleting Duplicate Routing Definitions, page 413.

Note. Any-to-Local routing definitions are read-only, with the exception of the external alias name and transform information. You may need to change the external alias name for WSDL generation.

To initiate a system-generated routing definition:

1. From the Service Operations-General page, locate the Default Service Operation Version section.
2. In the Routing Actions Upon Save group box select one of the following options:
   - *Generate Any-to-Local.* Generates an any-to-local routing definition when you save the service operation record.
   - *Generate Local-to-Local.* Generates a local-to-local routing definition when you save the service operation.

     If working with a synchronous service operation, click the Transactional box if you want to generate a transactional local-to-local routing.
3. Click the Save button.

When you save the service operation the system generates the routing definition that you selected.

After you save the service operation definition the Routing Status group box displays a status of *Exists* for the routing definition generated.

To view the routing definition , click the Service Operations-Routings tab and click the name of the routing. The Routing Definitions page appears and you can view and modify routing definition details.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Routing Definition Naming Conventions, page 374

Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382

Chapter 17, "Managing Service Operation Routing Definitions," Defining Routing Parameters, page 388

Chapter 17, "Managing Service Operation Routing Definitions," Defining and Overriding Gateway and Connector Properties, page 393

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Regenerating System-Generated Routing Definitions

If a system-generated routing exists for a service operation and you change some aspect of the service operation, you can have the system regenerate the routing definition. However, any modifications made to the routing definition are lost when you regenerate it.

To initiate the regeneration of a routing definition, use the Routing Actions Upon Save box on the Service Operations-General page to regenerate the routing. To access this page select PeopleTools, Integration Broker, Integration Setup, Service Operations.

To regenerate a system-generated routing definition:

1. Access the Service Operations-General page (PeopleTools, Integration Broker, Integration Setup, Service Operations).
2. On the page, locate the Default Service Operation Version section.
3. In the Routing Actions Upon Save group box select one of the following options:
    - *Regenerate Any-to-Local.* Regenerates an any-to-local routing definition when you save the service operation definition.
    - *Regenerate Local-to-Local.* Regenerates a local-to-local routing definition when you save the service operation.
4. Click the Save button.

When you save the service operation the system regenerates the routing definition that you selected.

After you save the service operation record the Routing Status group box displays a status of *Exists* for the routing definition generated.

Adding Routing Definitions

This section discusses how to:

- Add a routing definition.
- Define general routing definition information.
- Define routing parameters for requests and responses.
- Define routing properties.
- Define and override gateway and connector properties.

Understanding Adding Routing Definitions

The following table describes the location within the PeopleSoft system where you can add routing definitions to the system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

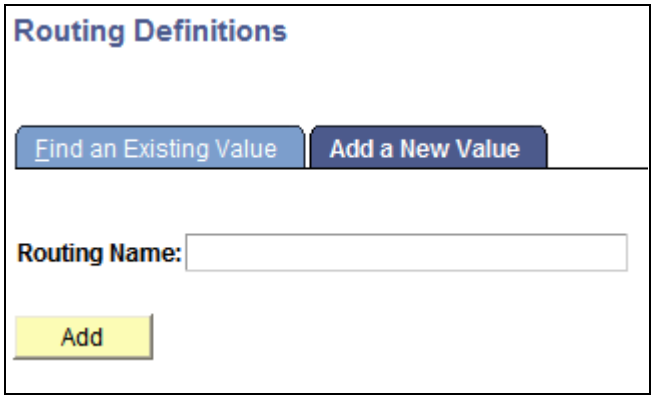
Page	*Page Object ID*	*Component*	*Component Object ID*	*Navigation*
Routings Definition	IB_ROUTINGDEFN	Routings	IB_ROUTINGDEFN	PeopleTools, Integration Broker, Integration Setup, Routings
Service Operations - Routings	IB_SERVICERTNGS	Service Operations	IB_SERVICE	PeopleTools, Integration Broker, Integration Setup, Service Operations. Select the Routings tab.
Node - Routings	IB_NODEROUTINGS	Nodes	IB_NODE	PeopleTools, Integration broker, Integration Setup, Nodes. Select the Routings tab.

Note. When using the RSS feeds functionality you may need to create a routing definition for a non-default service operation version. The only location that you can add a routing definition for a non-default service operation version is in the Routings component.

Note. You cannot add routings to the system for REST-based services. You can only use system generated routings (any-to-local or local-to-local) for REST-based services.

Adding Routing Definitions Using the Routings Component

The following graphic shows the page used to add a routing record when using the Routings component:



Adding a routing record from the Routings component

To add a routing record using the Routings component:

1. Select PeopleTools, Integration Broker, Integration Setup, Routings.
2. Click the Add a Value tab.
3. In the Routing Name field, enter a name for the routing definition.
4. Click the Add button.

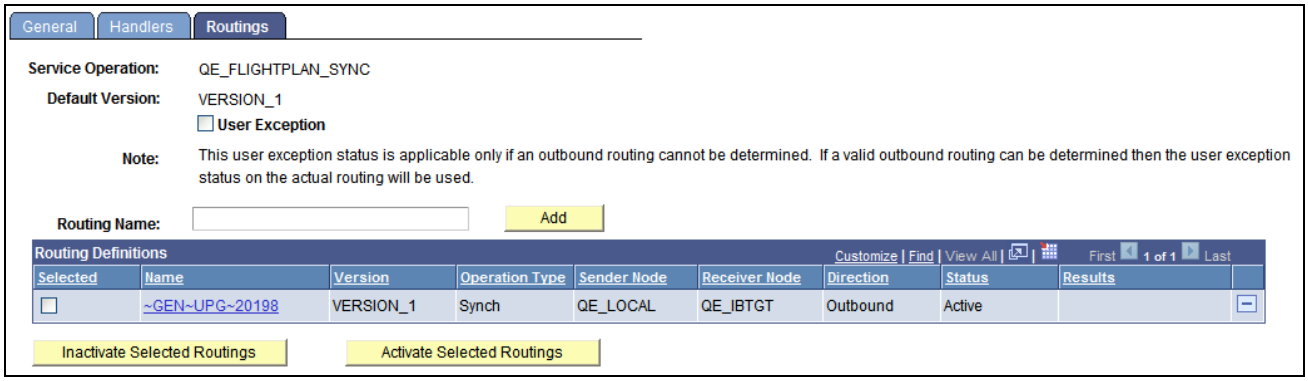
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The routing definition is added to the system and the Routing Definitions page appears where you can define the routing details.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382.

Adding Routing Definitions From Service Operation Definitions

When working in a service operation definition, you can use the Service Operations-Routing page (IB_SERVICERTNGS) to add a routing definition to the service operation. The following example shows the Service Operations-Routings page.



Adding a routing definition from the Service Operations-Routings tab

When you are adding a routing definition for a synchronous service operation, the system displays a User Exception box at the top of the page. Select the box to capture any exceptions that occur before the system validates an outbound routing for the service operation. This option enables you to capture errors such as inactive routing definitions, multiple routing definitions, any errors that result from running the OnRouteSend PeopleCode event, and other initialization errors. When the User Exception option is enabled and if an exception does occur before a routing can be validated, the system returns a response message to the integration partner that contains the exception.

To add a routing definition from a service operation definition:

1. Access the Service Operations - Routings page (PeopleTools, Integration Broker, Integration Setup, Service Operations. Click the Routings tab.)
2. (Optional.) Check the User Exception box to capture exceptions before routing validation.
3. In the Routing Name field, enter a name for the routing definition.
4. Click the Add button.

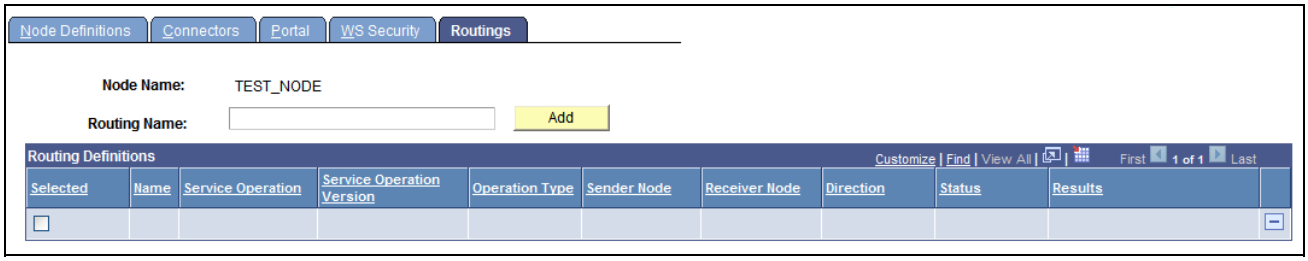
The routing definition is added to the system and the Routing Definitions page appears where you can define the routing details.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Adding Routing Definitions Using the Nodes Component

The following graphic shows the page used when adding a routing definition using the Nodes component.



Adding a routing definition from the Nodes-Routings page.

To add a routing definition from the Nodes-Routings page:

1. Access the Nodes-Routings page (PeopleTools, Integration Broker, Integration Setup, Nodes. Click the Routings tab.)
2. In the Routing Name field, enter a name for the routing definition.
3. Click the Add button.

The routing definition is added to the system and the Routing Definitions page appears where you can define the routing details.

See Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382.

Configuring Routing Definitions

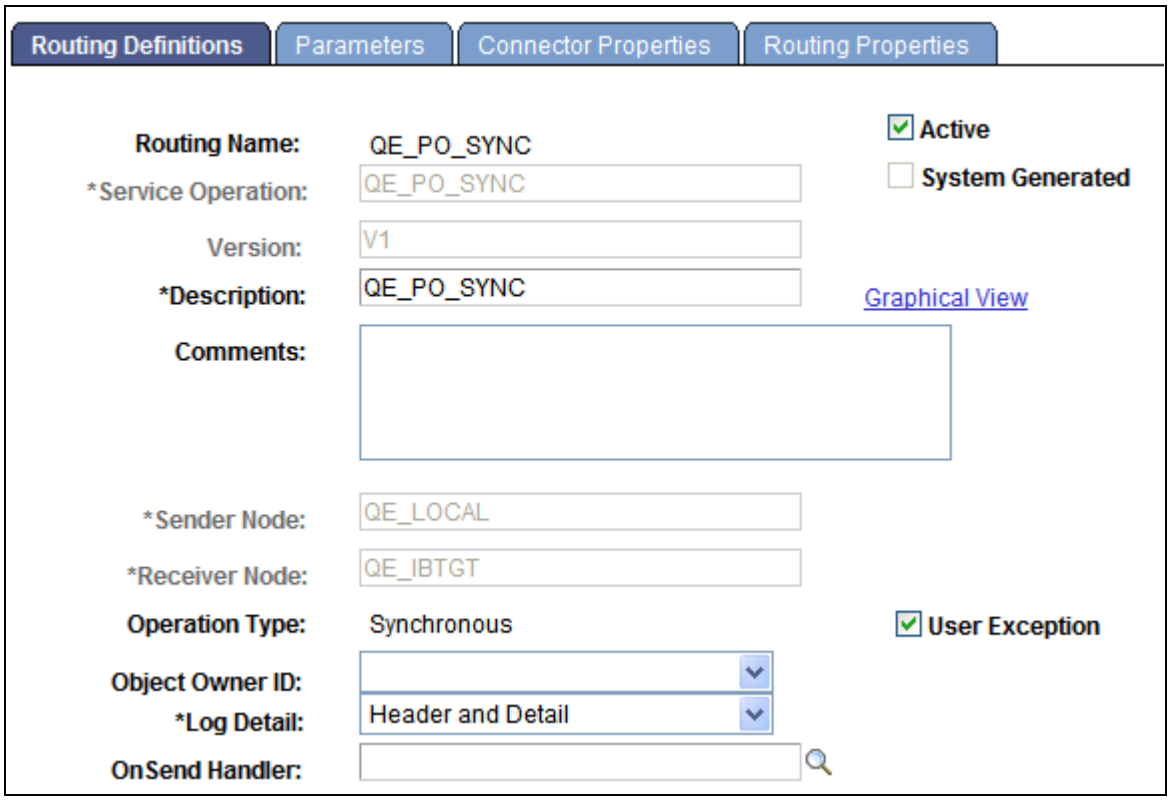
This section discusses how to:

- Define general routing information.
- Define routing parameters.
- Define and override gateway and connector properties.
- Define routing properties.

Defining General Routing Information

After you add a routing definition to the system use the pages of the Routing component to define the routing details. The following graphic shows the Routing Definitions page that you use to define general routing information, as accessed from the Service Operations-Routing Definitions page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



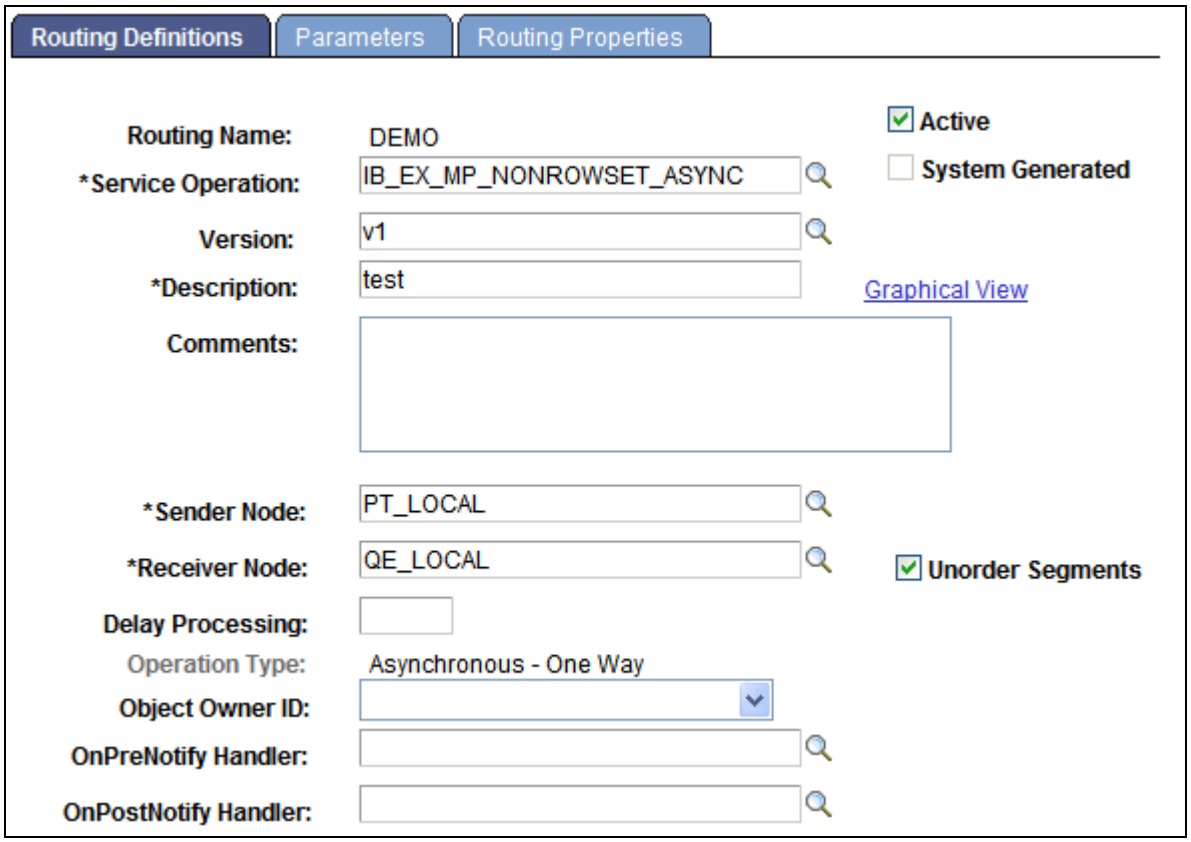
Defining general routing information for a synchronous service operation on the Routing – Routing Definitions page.

The fields that appear on the Routing–Routing Definitions page vary based on if you are defining a routing for a synchronous service operation or an asynchronous service operations.

The previous example show the Routing–Routings Definition page for a synchronous service operations. Fields such as Log Detail, User Exception, and OnSend Handler only appear on the page when you are working with a synchronous service operation.

The following example shows the same page for an asynchronous service operations:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Defining general routing information for an asynchronous service operation on the Routing – Routing Definitions page.

The various ways to access this page are discussed in the previous section.

See Chapter 17, "Managing Service Operation Routing Definitions," Understanding Adding Routing Definitions, page 379.

When you add a routing definition from a service operation record, the PeopleSoft system automatically populates some of the data on this page based on the data in the service operations record from which you created the routing. Automatically populated data includes the service operation name, version, and routing type.

Routing Name Indicates the name of the routing definition. This name is specified when you add a routing definition to the system.

Service Operation Enter the name of the service operation that will use the routing.

If you access the Routings component from the Service Operations-Routing tab, PeopleSoft Integration Broker automatically populates this information.

Active (Optional.) Check the box to activate the routing.

By default, new routing definition are active.

If any of the referenced nodes are inactive, you cannot activate the routing.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

System Generated When selected, indicates that the PeopleSoft system generated the routing definition.

Version Indicates the version of the service operation selected.

Description Description of the routing definition. If you do not enter a description, this value defaults to the name of the service operation associated with the routing definition upon save.

Graphical View Click this link to view saved routing definitions in graphical format.

See Chapter 17, "Managing Service Operation Routing Definitions," Viewing Routing Definitions in Graphical Format, page 404.

Comments (Optional.) Enter comments about the routing definition.

Sender Node Enter the name of the sending node.

Receiving Node Enter the name of the receiving node.

Unordered Segments (Optional.) Check the box to indicate that system should process the service operation messages in parallel.

The Unordered Segments check box appears only under the following conditions:

- The local node is the receiving node.
- After you save the initial routing definition

This feature is frequently used in conjunction with the OnPreNotify Handler and OnPostNotify Handler fields to perform pre- or post-processing on segmented messages.

Operation Type Indicates the service operation type. PeopleSoft Integration Broker automatically populates this information when you select the service operation.

User Exception The User Exception check box displays only for synchronous service operations.

(Optional.) Check the box to enable exception handling using PeopleCode.

When enabled and an error occurs you can handle any errors in the calling PeopleCode.

If not enabled any errors that occur cause the program to stop.

Object Owner ID (Optional.) From the drop-down list box, select the owner of the definition.

The owner ID helps to determine the application team that last made a change to the definition. The values in the drop-down list box are translate table values that you can define in the OBJECTOWNERID field record.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnPreNotify Handler (Optional.) From the drop-down list box, select an OnPreNotify handler to perform pre-processing on segmented messages being processed in parallel by the system.

This page control appears only when the Unordered Segments check box is selected.

OnPostNotify Handler (Optional.) From the drop-down list box, select an OnPostNotify handler to perform post-processing on segmented messages being processed in parallel by the system.

This page control appears only when the Unordered Segments check box is selected.

Accept Compression (Optional.) The Accept Compression check box displays only for inbound synchronous non-REST service operations.

Setting compression for REST service operations is discussed elsewhere in this PeopleBook.

See Chapter 12, "Managing REST Service Operations," Setting Compression for REST Service Operations, page 315.

Check the box for the system to send the response to the consumer compressed.

You must compress the response before sending, using PeopleCode or by setting the Min Message Size Compression parameter in PSAdmin.

You can override the compression of the response using PeopleCode by setting the CompressionOverride property on IBInfo. The following example shows sample PeopleCode to override compression of the response message:

```
&MSG.IBInfo.CompressionOverride = %IntBroker_Compress;
Set/Get property for Compression override.  Valid parms : ⇒
%IntBroker_Compress, %IntBroker_UnCompress, %IntBroker_⇒
Compress_Reset
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Log Detail

The Log Detail drop-down list box displays only for synchronous service operations.

This option enables you to set the level of information logging for synchronous messages that is viewable in the Service Operations Monitor.

The valid values are:

- *Header.*

  Log header information only. With this option, you can view synchronous message header information in the Service Operations Monitor.

- *Header and Detail.* Log header and message detail information. With this option, you can view synchronous message header information and XML message content on in the Service Operations Monitor.

  When you select this option for a consumer REST service operation routing, a Service Operation Security link appears and you can add permissions to the service operation. Doing so enables access to service operation details in the Service Operations Monitor.

- *No Logging.* (Default.) Turn off all logging. No information is available to view in the Service Operations Monitor.

Service Operation Security

A Service Operation Security link appears when the selected routing definition is for an outbound REST-based service operation

Click the link to access the Web Service Access page and to add permissions to the view details for the service operation in the Service Operations Monitor.

OnSend Handler

This field displays when an OnSend handler is defined for the service operation and the sending node is the local node. It also displays when you the system is serving as a hub, and neither the sender nor receiver are local.

Select a handler from the list. This handler runs when a message is sent or received to perform processing logic.

Schema Validation Details

This field displays only when working with any-to-local routing definitions.

(Optional.) When this check box is selected and a schema validation error occurs the raw schema parser errors are returned to the consumer within the default message CDATA tag. If the check box is not selected (Default) then if a schema validation error occurs the systems uses the standard message set/ ID framework to generate the error.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

OnReceive Handler This field displays when an OnReceive handler is defined for the service operation and one of the following conditions exists:

- The sending node is the local node, or
- The service operation type is asynchronous request/response where the sender is not local and the receiver is local, or
- The system is serving as a hub, and neither the sender (initiator) nor final receiver are local.

Select a handler from the list. This handler runs when a message is sent or received to perform processing logic.

Use Secure Target Location This check box appears only when you are working with an outbound asynchronous request/response service operation.

Check the box to use the secure target location as the endpoint for the response. If the box is not checked, the (unsecured) target location is used.

Defining Routing Parameters

This section provides an overview of defining transformations for any-to-local routing definitions and discusses how to define general routing parameters.

Understanding Routing Parameters for Requests and Responses

A routing definition contains routing parameters for each inbound request, inbound response, outbound request and outbound response associated with a service operation. The routing parameters that you can define include for each request or response include, routing alias, message names before and after transformation, and transformation program names.

You define routing parameters using the Routings-Parameters page in the Routings component.

The following tables list the number of routing parameters a routing definition contains based on the service operation type and whether the sending and receiving nodes are local.

Note. Parameters for routing definitions for REST service operations are not included in the following tables, since they are system generated.

Asynchronous service operations have the following routing parameters:

Service Operation Type	*Sender is Local*	*Receiver is Local*	*Inbound Request Routing*	*Outbound Request Routing*	*Inbound Response Routing*	*Outbound Response Routing*
Asynchronous One- Way	Y	Y	N	Y	N	N
Asynchronous One- Way	N	N	N	Y	N	N

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operation Type	Sender is Local	Receiver is Local	Inbound Request Routing	Outbound Request Routing	Inbound Response Routing	Outbound Response Routing
Asynchronous One- Way	N	Y	Y	N	N	N
Asynchronous One- Way	N	N	Y	Y	N	N

Synchronous service operations have the following routing parameters:

Service Operation Type	Sender is Local	Receiver is Local	Inbound Request Routing	Outbound Request Routing	Inbound Response Routing	Outbound Response Routing
Synchronous	Y	Y	Y	Y	Y	Y
Synchronous	Y	N	N	Y	Y	N
Synchronous	N	Y	Y	N	N	Y
Synchronous	N	N	Y	Y	Y	Y

Asynchronous-to-synchronous service operations may have the following routing parameters:

Service Operation Type	Sender is Local	Receiver is Local	Inbound Request Routing	Outbound Request Routing	Inbound Response Routing	Outbound Response Routing
Asynchronous-to-synchronous	Y	Y	Y	N	N	Y
Asynchronous-to-synchronous	Y	N	N	Y	Y	N
Asynchronous-to-synchronous	N	Y	Y	N	N	Y
Asynchronous-to-synchronous	N	N	Y	Y	Y	Y

Asynchronous request/response service operations may have the following routing parameters:

Service Operation Type	Sender is Local	Receiver is Local	Inbound Request Routing	Outbound Request Routing	Inbound Response Routing	Outbound Response Routing
Asynchronous Request / Response	Y	Y	N	Y	Y	N

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Service Operation Type	*Sender is Local*	*Receiver is Local*	*Inbound Request Routing*	*Outbound Request Routing*	*Inbound Response Routing*	*Outbound Response Routing*
Asynchronous Request / Response	Y	N	N	Y	Y	N
Asynchronous Request / Response	N	Y	Y	N	N	Y
Asynchronous Request / Response	N	N	Y	Y	Y	Y

Understanding Transformations on Any-to-Local Routing Definitions

If you define a transformation on an any-to-local routing definition, the system uses the input message.version on the transform for the inbound request for WSDL. If a transform is defined on the outbound response, then the system uses the message.version on the output of the transformation for WSDL.

In cases where the input message.version or output message.version are not defined on the transform, the system uses the request or response message.version defined on the service operation for WSDL.

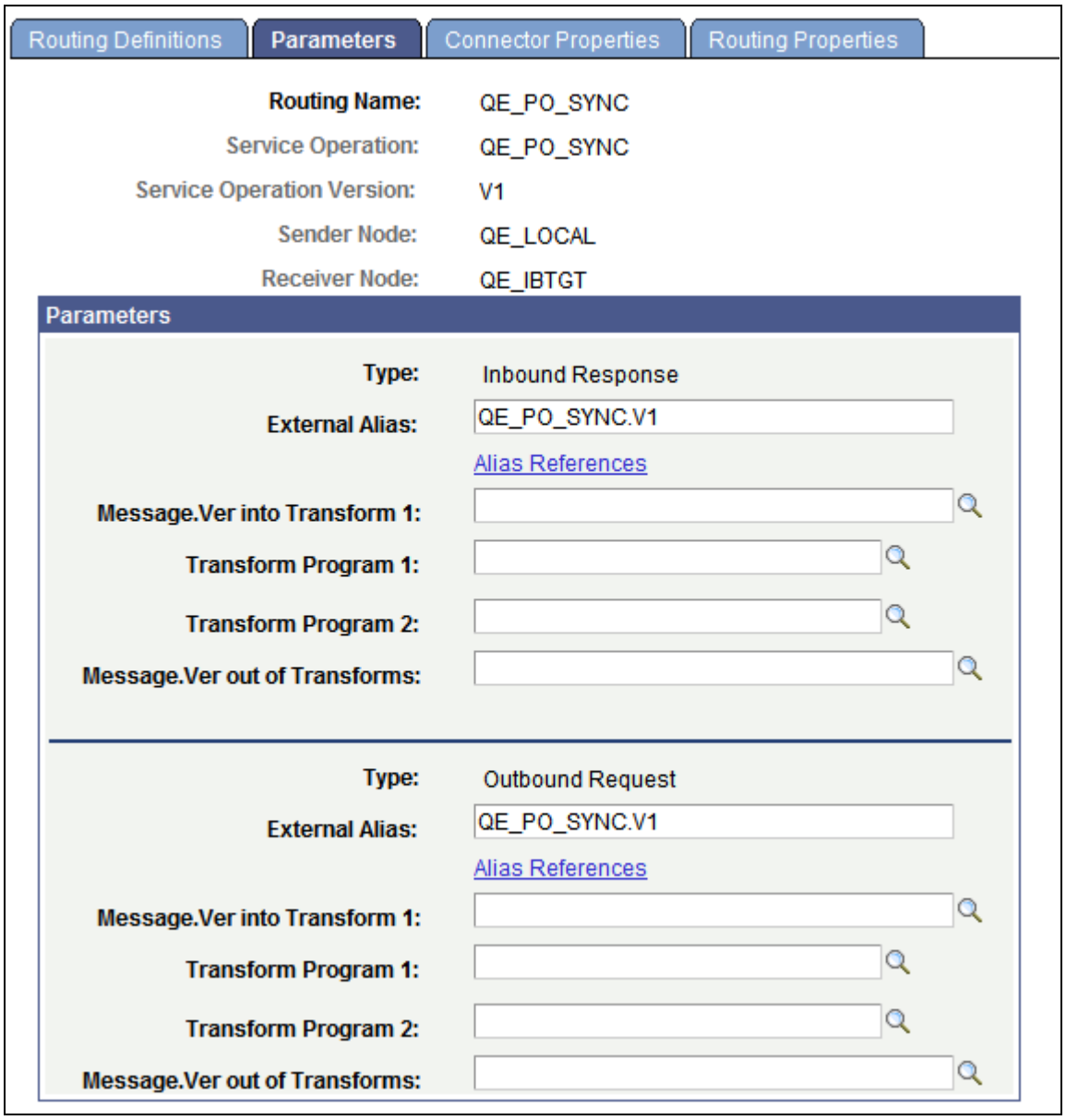
Note that any-to-local routing definitions are read-only when WSDL has been exported. As a result, you cannot change the in/out message transformation, aliases, and so on.

Defining Routing Parameters for Requests and Responses

Use the Routings-Parameters page (IB_ROUTINGDEFNDOC) to view and define parameters for requests and responses associated with a service operation. Information you define includes, routing external aliases for inbound and outbound requests and responses, as well as any inbound or outbound transformations to invoke.

To access the Routings - Parameters page, select PeopleTools, Integration Broker, Integration Setup, Routings and click the Parameters tab. The following graphic shows the Routings-Parameter page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Routings – Parameters page

The following page elements display on the Routings-Parameters page:

Type Specifies the routing direction and the type of message (request or response) associated with the service operation. This information is automatically populated from the service operation definition.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

External Alias

This alias is used as a SOAPAction attribute in the WSDL binding to identify the service operation in the Integration Broker metadata.

The routing external alias defaults to *<ServiceOperationAlias>.<Version>*, if present. Otherwise it defaults to *<ServiceOperation>.<Version>*.

In an asynchronous request/response any-to-local routing, the outbound routing alias format is *<Alias Name>_CALLBACK.<Version>*.

For inbound transactions you can fire multiple service operations for one invocation when external aliases on the routing definition are the same for each service operation. This is called service operation mapping.

See Chapter 17, "Managing Service Operation Routing Definitions," Service Operation Mapping, page 375.

Duplicate external aliases are not allowed for synchronous operations.

See Chapter 17, "Managing Service Operation Routing Definitions," Searching for Duplicate External Routing Aliases, page 410.

Alias References

Click the link to view other routing definitions with the same external alias.

See Chapter 17, "Managing Service Operation Routing Definitions," Searching for Duplicate External Routing Aliases, page 410.

WS Security

This link appears on the Parameters page when you are working with:

- Non-REST service operations and are integrating with external nodes.

  Click the link to override WS Security settings configured at the node level for the outbound request or response.

  See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Overriding Node-Level WS-Security Settings on Routing Definitions.

- Outbound consumer REST service operations.

  Click the link to set the security level of the outbound request as required by the REST provider.

  See Chapter 12, "Managing REST Service Operations," Securing Consumer REST Service Operations, page 312.

Message.Ver into Transform 1

Displays the name of the request or response message associated with the service operation before any transformations are applied.

For inbound transactions, this is the message name and version as it arrives from the integration partner system, before any transformations are applied.

For outbound transactions, this is the message name and version directly from the PeopleSoft system, before any transformations are applied.

Transform Program 1

(Optional.) Enter the name of the transform program to invoke on the message listed in the Message.Ver info Transform 1 field.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Transform Program 2	(Optional.) Enter the name of the transform program to invoke after the transform program in the Transform Program 1 field has completed processing. When you invoke two transform programs, the output from the first transform program (Transform Program 1) is used as the input into the second transform program (Transform Program 2).
Message.Ver out of Transforms	(Optional.) Enter the name of the message after all transform program have completed processing. For inbound messages, this is the message name and version that the PeopleSoft system is expecting. For outbound messages, this is the message name and version that the integration partner system is expecting.

When the Routings-Parameters page first displays values for the Message.Ver into Transform 1 and Message.Ver out of Transforms fields display values to assist you in choosing transform programs. After you save the page, values do not appear in these fields unless the transform programs have an input/output messages associated with them.

Note. Based on the transform selected, the message.version of the inbound request or response and the message.version of the outbound request or response that the system populated on the page can be different then those specified on the component. Should this occur a warning message displays and you can accept or reject the message.version information populated on the Routings - Parameters page. If you reject the message.version information populated on the page, you can modify the fields with the appropriate message.version information, or change the information that is specified on the component.

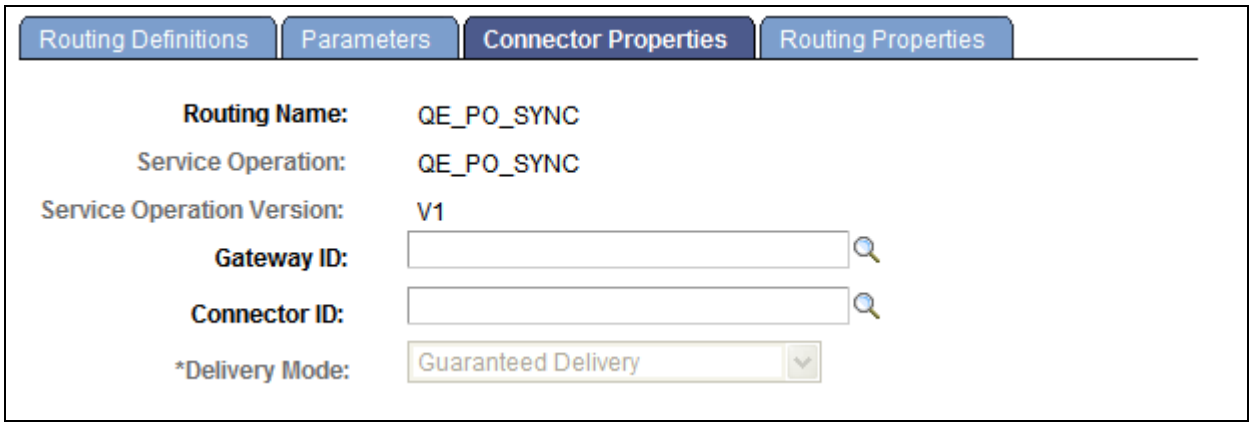
Defining and Overriding Gateway and Connector Properties

The Routings-Connector Properties page (IB_ROUTINGDEFCON) enables you to define and override the default integration gateway and target connector that the local node uses to communicate with an endpoint for a specific routing definition.

Note. The Routings-Connector Properties page displays in the Routings component only if the receiving node is not the local node.

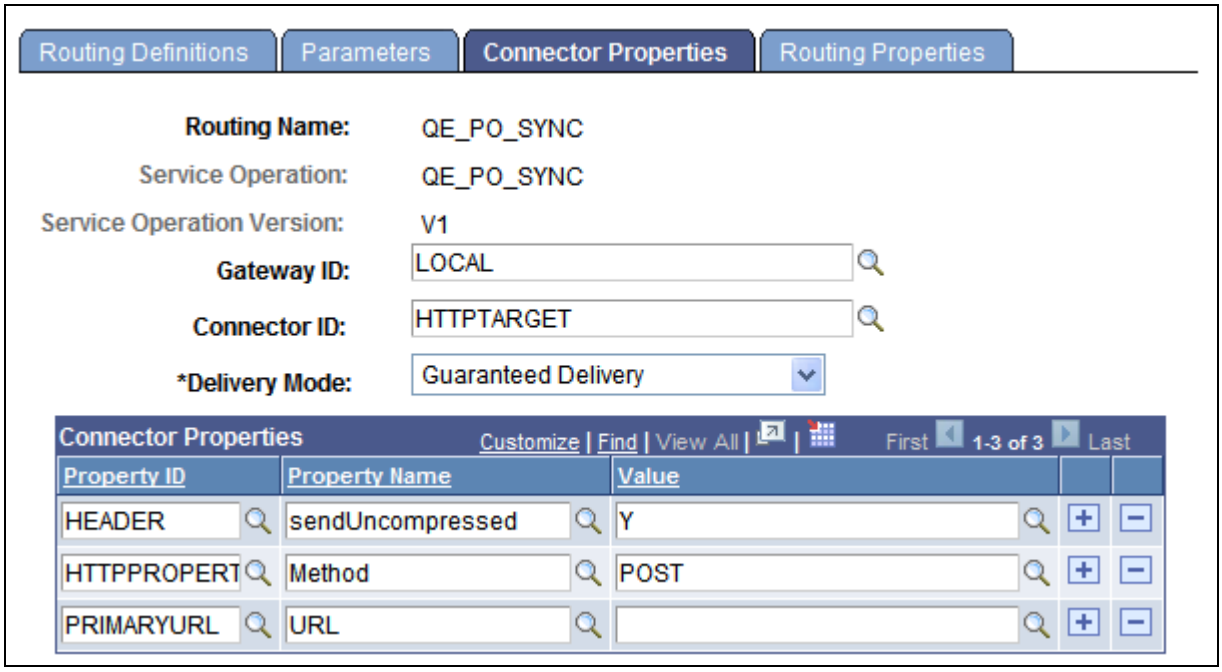
The following graphic shows the Routings-Connector Properties pages used to define connector properties for a routing definition:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Routings-Connector Properties page

After you select an integration gateway and target connector with which to work, the system displays the required properties for the connector that you can set and override. To set or override additional properties add them to the properties list with the desired values.



Properties for the HTTP target connector

The Routings-Connector Properties tab features the following controls:

Gateway ID Click the Lookup button to select an integration gateway.

Connector ID Click the Lookup button to select a target connector that resides on the gateway entered in the Gateway ID field.

After you select a target connector, its required properties appear.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Delivery Mode Select one of the following options:

- Guaranteed. (Default.)
- Best Effort.

These options are discussed in detail elsewhere in PeopleBooks.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Using Listening Connectors and Target Connectors," Setting Target Connector Delivery Modes.

Save Click the Save button to save your changes.

Overriding Default Integration Gateways

To override the default integration gateway, use the Gateway ID field Lookup button to select a gateway. You must also select a target connector on the new gateway to use and define properties for that connector.

Overriding Default Target Connectors

To override the default target connector on the default integration gateway, use the Gateway ID field Lookup button to select the default gateway. Use the Connector ID field Lookup button to select a different target connector that resides on the gateway and then define properties for the new connector.

For REST-based services the connector ID default to *HTTPTARGET* and cannot be changed.

Overriding Default Connector Properties

To override the default target properties for the default target connector, use the Gateway ID field Lookup button to select the default gateway. Use the Connector ID field Lookup button to select the default target connector and then adjust the gateway properties as appropriate.

HTTP header properties, those with the property ID of *HEADER*, are the only properties you can add for routings for outbound REST-based service operations.

Defining Routing Properties

This section provides an overview of routing properties and discusses how to add them in the form of name/value pairs to routing definitions.

Understanding Routing Properties

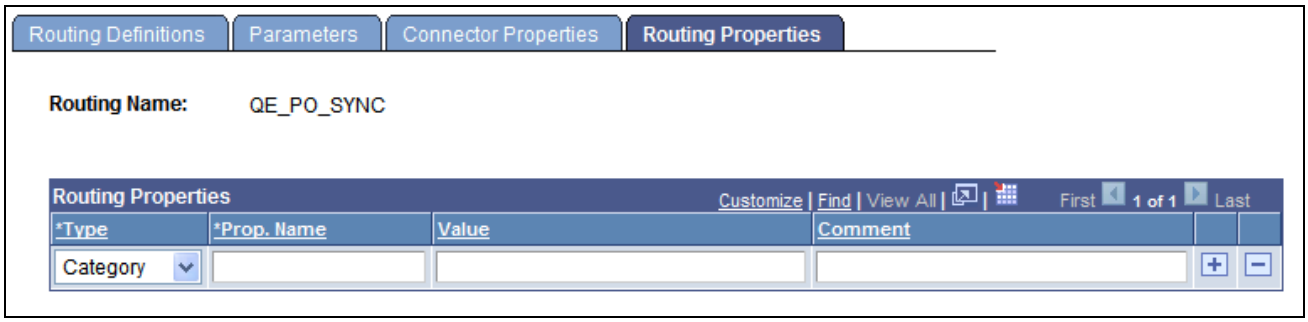
Routing properties are user-defined name/value pairs that denote data contained in transformations defined for a routing definition.

Once they know the name/value pairs in a transformation, developers can extract the data from transformations using application classes. Developers might use the name/value pairs data to add to XML, perform SELECT actions in tables, and so on.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Defining Routing Property Name/Value Pairs

Use the Routings-Routing Properties page (IB_ROUTINGDEFNPROP) to add routing properties to a routing definition. The following example shows the Routings-Routing Properties page:



Routing Properties page

To add routing property name/value pairs:

1. Access the Routings-Routing Properties page (PeopleTools, Integration Broker, Integration Setup, Routings and click the Routing Properties tab).
2. From the Type drop-down list box, select a property type. The options are:
   - *Category.*
   - *Ident.*
   - *Search.*
3. In the Prop. Name field, enter a name for the property.
4. (Optional.) In the Value field, enter a value for the property.
5. (Optional.) In the Comment field, enter a comment or description for the name/value pair.
6. Click the plus button (+) to add additional rows and property name/value pairs.
7. Click the Save button.

Using Introspection to Create Routing Definitions

This section discusses how to:

- Select service operations for which to create routing definitions.
- Select the node to introspect.
- Select routing definitions to generate.
- View introspection results.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Using Introspection to Create Routing Definitions

You can introspect PeopleTools 8.52 nodes to create inbound or outbound point-to-point routing definitions on nodes that have matching service operations and versions for the local node. You can introspect *PIA* and *External* nodes types.

The following table lists the actions you can perform using introspection:

Routing Definition On Local Node	***Routing Definition on Introspected Node***	***Introspection Option***
Inbound point-to-point routing.	None.	Delete routing on local node.
Outbound point-to-point routing.	None.	Delete routing on local node.
None.	Inbound point-to-point routing.	Create outbound point-to-point routing.
None.	Outbound point-to-point routing.	Create inbound point-to-point routing.
None.	Any-to-local routing. (Inbound.)	Create outbound point-to-point routing.

Prerequisites for Using Introspection to Create Routing Definitions

The following prerequisites exist for using introspection to create routing definitions:

- Nodes to be introspected must be active. To verify that a node is active, open the node definition for the node and make sure that the Active box is checked on the definition.
- External nodes to be introspected must have a WSIL URL defined on the node definition.

See Also

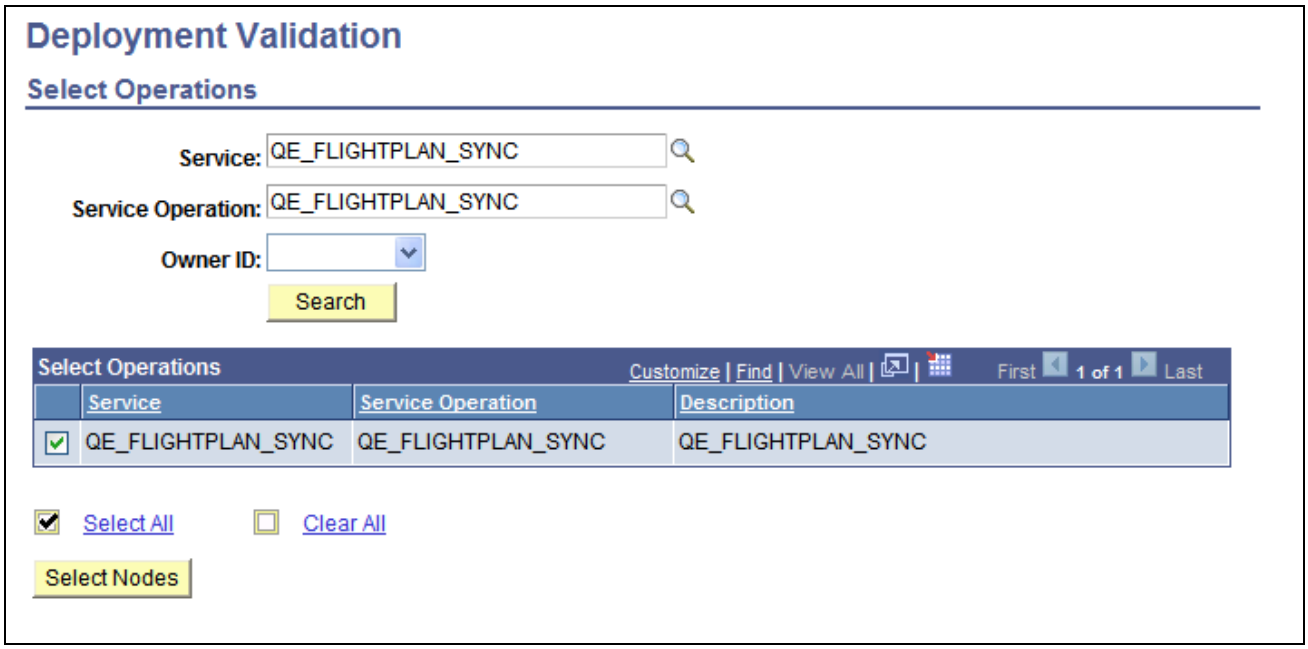
PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes," Configuring Nodes

Selecting Service Operations for Which to Create Routing Definitions

To begin using introspection to generate routing definitions, select the service operations for which to create routing definitions.

Use the Deployment Validation-Select Operations page (PTIB_INTROSPECT_0) in the Deployment Validation component (PTIB_INTROSPECTION) for this task. To access the page, select PeopleTools, Integration Broker, Web Services, Deployment Validation. The following example shows the Deployment Validation - Select Service Operations page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Deployment Validation-Select Operations page

You can search by service name and service operation. You can also search by object owner ID, if one is defined for the service. You can enter one or more of these criteria when performing your search. If you select no search options, the system searches for and returns all service operations in the database.

After you enter the search criteria and click the Search button, the results display in the Select Operations grid and you can select the service operations for which to generate routing definitions.

You can select one or more services operations for which to generate routing definitions.

To select services operations for which to generate routing definitions:

1. Select PeopleTools, Integration Broker, Web Services, Deployment Validation. The Deployment Validation-Select Operations page appears.
2. Enter search criteria for the services operations for which to generate routing definition. Provide one or more of the search criteria to narrow your search. Select no search criteria to retrieve a list of all service operations in the database.
   - In the Service field, enter a service name.
   - In the Service Operation field, enter a service operation name.
   - From the Object Owner ID drop-down list box, select the object owner of the service to provide.
3. Click the Search button.

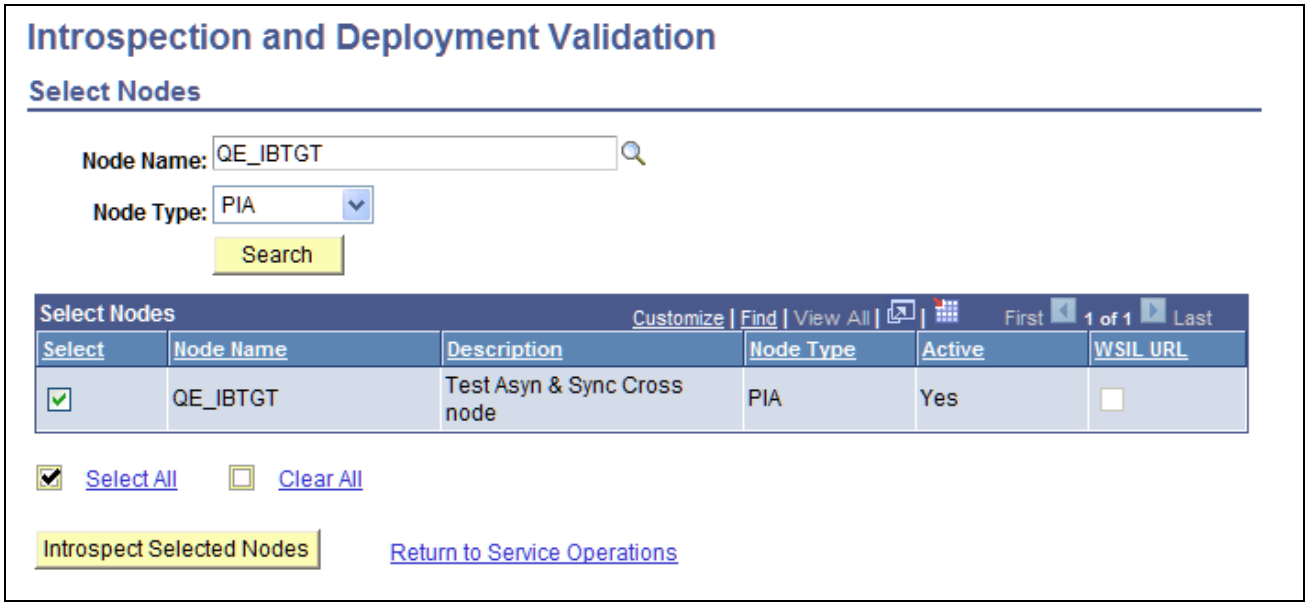
   A Select Operations grid appears that contains the search results.
4. Check the box next to each service operation for which to generate a routing definition.

   To clear a selection, check the box again.
5. Click the Select Nodes button to proceed to select nodes to introspect.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Selecting Nodes to Introspect

Use the Introspection and Deployment Validation-Select Nodes page (PTIB_INTROSPECT_1) to select the nodes to introspect. The following example shows this page:



Select Nodes page with the node QE_IBGT selected for introspection.

To select nodes to introspect:

1. Enter one or more of the following search criteria to search for a node:
   - In the Node Name field, enter a node name.
   - From the Node Type drop-down list box, select a node type. The options are:

     PIA Designates the node as a PeopleSoft database that uses PeopleSoft Integration Broker.

     External Designates the node as an entity that doesn't use PeopleSoft Integration Broker.

     Note. The ICType node type displays in the list, however you cannot introspect the ICType node type to create routing definitions.

   - Select no search criteria to retrieve a list of all nodes defined in the database.
2. Click the Search button.

   A Select Nodes grid appears that contains the search results.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Check the box next to the nodes to introspect.

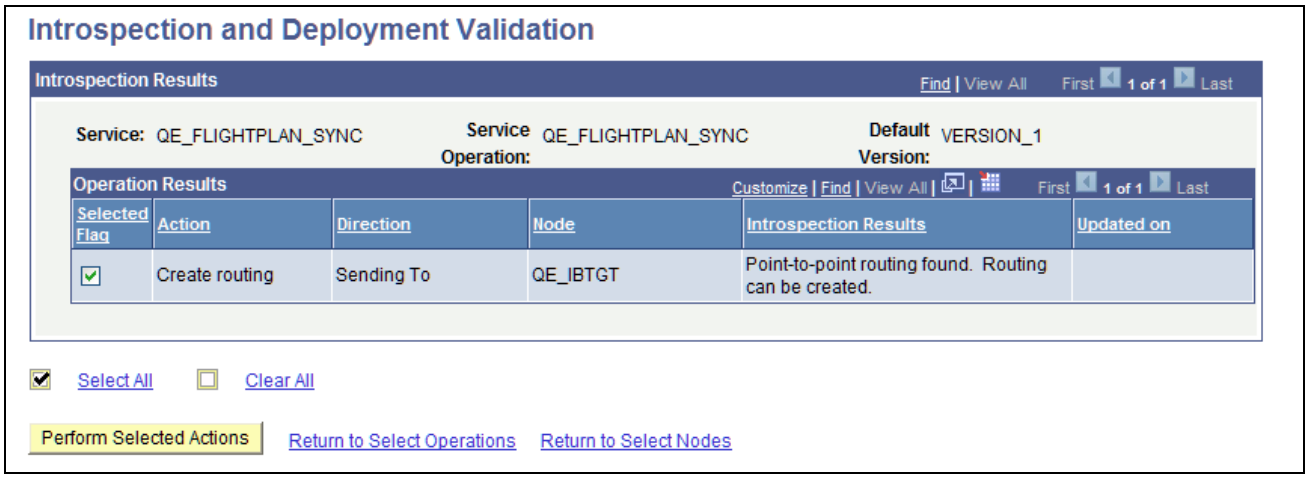
   If a node displays in the list, but isn't available for selection, the check box is grayed out. A node may not be available for selection due to not being active or in the case of external nodes, no WSIL URL is defined on the node definition.

4. Click the Introspect Selected Nodes button to introspect the node or nodes that you selected.

Click the Return to Service Operations link at any time to go back to the Deployment Validation-Select Operations page to modify the selection of service operations for which to generate routing definitions.

Selecting Routing Definitions to Generate

Use the Introspection and Deployment Validation-Introspection Results page (PTIB_INTROSPECT_2) to view the introspection results and select the routing definitions to generate.



Selection the action to create a point-to-point routing.

After you introspect one or more nodes an Operation Results box displays for each service operation for which you are generating routing definitions. Select the actions to perform and click the Perform Selected Actions button.

The following page elements appear on the Introspection and Deployment Validation-Introspection Results page:

Service The name of the service.

Service Operation The name of the service operation.

Default Version The default version of the service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Action Indicates the possible action to perform on the service operation. The valid values are:

- *None.* Displays when a valid routing already exists between nodes. Also displays if there are no matching routing definitions on sending and receiving nodes.
- *Delete Routing.* Displays when there is a routing on the source node, but no corresponding routing on the target node.
- *Create Routing..* Displays when matching routing definitions are present on the sending and receiving nodes.

Direction Indicates if the direction of the transaction is inbound or outbound. The valid values are:

- *Sending To.* Indicates an outbound routing.
- *Receiving From.* Indicates an inbound routing.
- *Blank.* No routing found.

Node Indicates the name of the node introspected.

Introspection Results Indicates the results of the introspection. The valid values are:

- *A Routing exists locally. No routing found on the node. Delete local routing?.* A routing definition exists on the local database, but there is no corresponding routing on the introspected node. You may delete the routing definition on the local node.
- *No Match Found.* Indicates that no matching routing was found on the introspected node.
- *Routing Created.*The PeopleSoft system found a matching routing definition on the introspected node and created the routing.
- *Routing Deleted.* The PeopleSoft system deleted the routing.
- *Any-to-Local routing found. Routing can be created.* The target system has an any-to-local routing defined, meaning that it will accept transactions from any node. A routing will be created.
- *Point-to-point routing found. Routing can be created.*
- *Valid Routings Found.* Routing definitions between systems already exist.

Updated On Indicates the date and time that a change or delete action was performed in the current session.

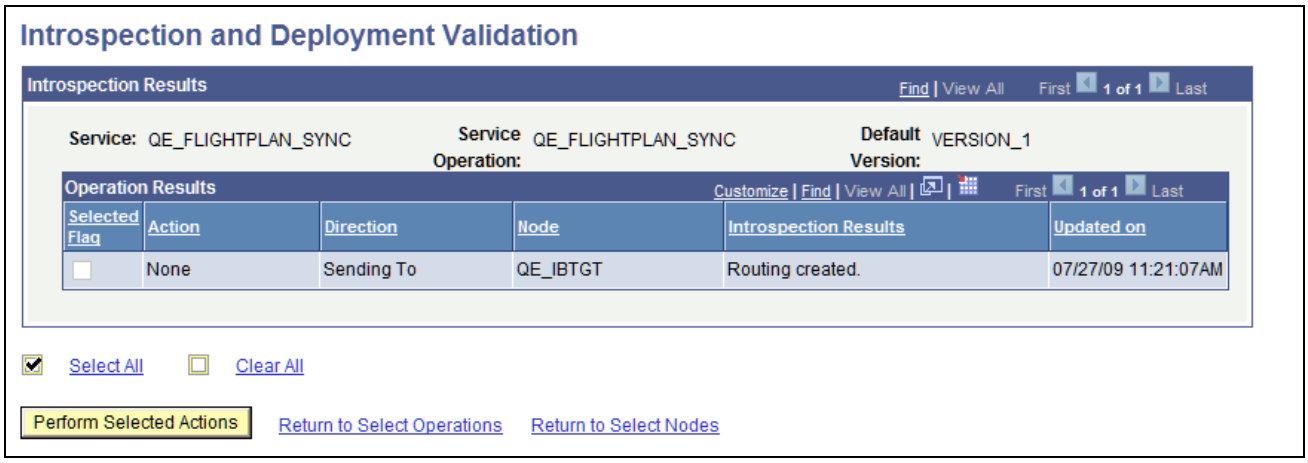
Select All Click the button to select to perform all actions listed in the Action field for each service operation displayed.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Clear All	Click the button to clear all selections.
Perform Selected Action	Click the button to perform the action described for selected Action fields in the Operation Results grid for each service operation.
Return to Select Operations	Click the link to go back to the Deployment Validation-Select Operations page to modify the selection of service operations for which to generate routing definitions. This option displays only when introspection is initiated from the Deployment Validation component.
Return to Service Operation	Click the link to go back to the Service Operations page. This option displays only when introspection is initiated from the Service Operations page.
Return to Select Nodes	Click the link to go back to the Introspection and Deployment Validation-Select Nodes page to modify the selection of nodes to introspect.

Viewing Introspection Results

After the system performs the selected actions, the following Introspection Results page (PTIB_INTROSPECT_2) appears:



Introspection Results page showing that the routing was created.

The Introspection Results field shows the status for each routing definition.

You can view routing definitions created using the Routing Definition page or the Service Operations-Routings page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Defining General Routing Information, page 382

Chapter 17, "Managing Service Operation Routing Definitions," Renaming Routing Definitions, page 412

Activating and Inactivating Routing Definitions

This section discusses how to:

- Use the Routings component to activate and inactivate routing definitions.
- Use the Service Operations component to activate and inactivate routing definitions.
- Use the Nodes component to activate and inactivate routing definitions.

Understanding Activating and Inactivating Routing Definitions

You can use the Routings component or the Service Operations component to activate and inactivate routing definitions.

Activating and Inactivating Routing Definitions in the Routing Component

You can use the Routings-Routing Definitions page to activate and inactivate a routing definition.

To activate or inactivate a routing definition:

1. Select PeopleTools, Integration Broker, Integration Setup, Routings.

   Select a routing definition with which to work.

2. Perform one of the following actions:

   - To activate the routing definition, check the Active check box.
   - To inactivate the routing definition, clear the Active check box.

3. Click the Save button.

Activating and Inactivating Routing Definitions in the Service Operations Component

You can also activate and inactivate routing definitions in the Service Operations component using the Service Operations-Routings page.

See Chapter 10, "Managing Service Operations," Activating and Inactivating Routing Definitions, page 260.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Activating and Inactivating Routing Definitions in the Nodes Component

You can use the Nodes-Routings page to access a routing definition and to activate or inactivate a routing.

To activate or inactivate a routing definition from the Nodes component:

1. Select PeopleTools, Integration Broker, Integration Setup, Nodes.
2. Click the Routings tab.
3. Locate the routing definition to activate or inactivate and click the Details link.

   The routing definition appears in the Routing Definitions page.
4. Perform one of the following actions:
   - To activate the routing definition, check the Active check box.
   - To inactivate the routing definition, clear the Active check box.
5. Click the Save button.

Viewing Routing Definitions in Graphical Format

This section discusses common elements used to view routing definitions in graphical format. It also discusses how to view routing definitions in graphical format.

Common Elements Used to View Routing Definitions in Graphical Format

This symbol denotes an integration participant.

Online and in color documents the light blue-colored symbol denotes the PeopleSoft system. A gray–colored symbol denotes the integration partner of the PeopleSoft system.

The symbol shown here is light blue and denotes the PeopleSoft system.

The left-facing arrow denotes an outbound flow of data from the PeopleSoft system.

This symbol denotes a transformation. This can occur on the inbound or outbound side of the integration.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

This symbol denotes the outbound flow of data from the PeopleSoft system out of the integration gateway.

This left-facing arrow denotes the inbound flow of data to the PeopleSoft system.

This symbol denotes the inbound flow of data into the integration gateway of the PeopleSoft system.

Alias Name of the routing alias.

Default Message Name of the default message as it leaves or enters the PeopleSoft system.

For outbound messages, this is the message name *before* any transformations are applied.

For inbound messages, this is the message name *after* any transformations are applied.

In Message The inbound message name before any transformations have been applied.

In Resp Transform Click the link to view information about transformations applied to the inbound response message.

Integration Not Active This link displays when one or more of the following items are inactive for an integration:

- Routing definition.
- Service operation version.
- Handler.
- Service operation permissions.

Click the link to view the inactive items.

See Chapter 17, "Managing Service Operation Routing Definitions," Viewing Integration Status and Activating Integration Metadata, page 407.

Out Message Name of the outbound message after any transformation have been applied.

Out Req Transform Click the link to view information about transformations applied to the outbound request message.

Receiver Name of the receiving node.

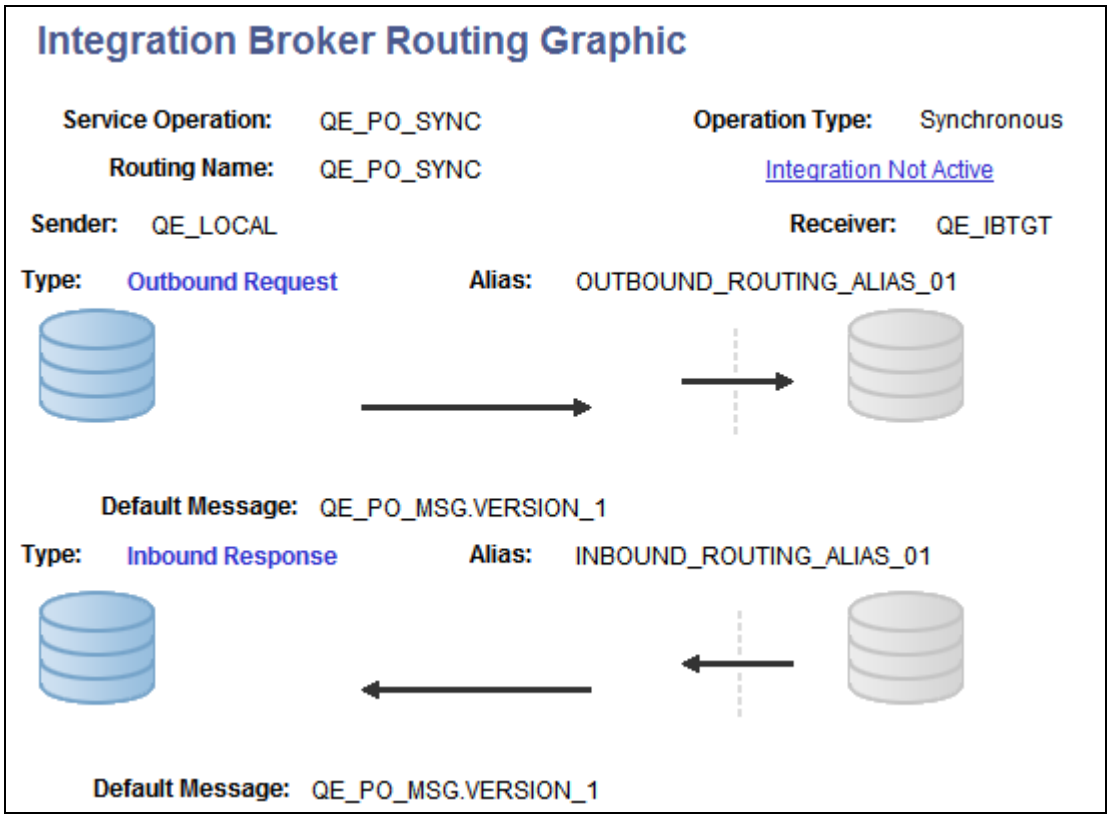
Return Click the button to return to the previous page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Routing Name	Name of the routing definition.
Operation Type	The service operation type.
Sender	Name of the sending node.
Service Operation	Service operation name and version.
Type	Type of request on the PeopleSoft system. The possible values are: • Outbound Request. • Inbound Request. • Inbound Response. • Outbound Response.

Viewing a Routing Definition in Graphical Format

The Integration Broker Routing Graphic page (IB_IMAGETEST2), shown in the following example, displays routing definition data in graphical format:



The routing definition QE_PO_SYNC in graphical format.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The example shows that the service operation *QE_PO_SYNC* is associated with the routing. It also shows that the routing type is *Synchronous.*

The sending node is *QE_LOCAL*, and the node is depicted graphically by the cylinders on the left side of the example. The receiving node is *QE_IBTGT* and is depicted graphically by the cylinders on the right side of the example.

The example shows the outbound request from the node *QE_LOCAL* to the node *QE_IBTGT* has a routing alias of *OUTBOUND_ROUTING_ALIAS.V1*. It also shows that the outbound default message associated with the routing is *QE_PO_MSG.VERSION_1*.

When the node *QE_IBTGT* sends back its response, it has a routing alias of *INBOUND_ROUTING_ALIAS.V1* and the message associated with the response is *QE_PO_MSG.VERSION_1*.

The dotted line depicts the integration gateway, and depending on the arrow direction shows the service operation leaving or entering the gateway.

Note in the example that an *Integration Not Active* link displays at the top right of the page. This indicates that one or more pieces of metadata associated with the integration is not active. Click the link to view and activate the data.

See Also

Chapter 17, "Managing Service Operation Routing Definitions," Viewing Integration Status and Activating Integration Metadata, page 407

Viewing Integration Status and Activating Integration Metadata

This section discusses how to:

- View inactive integration metadata.
- Activate integration metadata using the Integration Status page.

Understanding Viewing Integration Status and Activating Integration Metadata

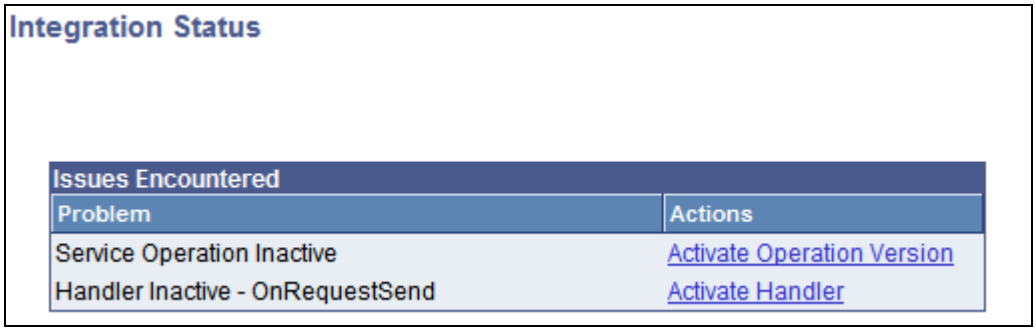
When you view a routing definition in graphical format, if any metadata associated with the integration is inactive, an Integration Not Active link displays on the Integration Broker Routing Graphic page. You can click the link to open an Integration Status page, where you can view the inactive metadata and activate the data.

Note. The Integration Not Active link displays only when an integrations contains metadata that is not set to *Active.*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Inactive Integration Metadata

To view inactive metadata for an integration, from the Integration Broker Routing Graphic page, click the Integration Not Active link. The Integration Status page (IB_IMAGETEST_SEC), shown in the following example, displays:



The Integration Status page lists the metadata that you need to activate.

The issues encountered box lists the metadata that is inactive and that you must activate before you can invoke the service operation.

The links in the Actions column enable you to activate the associated metadata directly from this page.

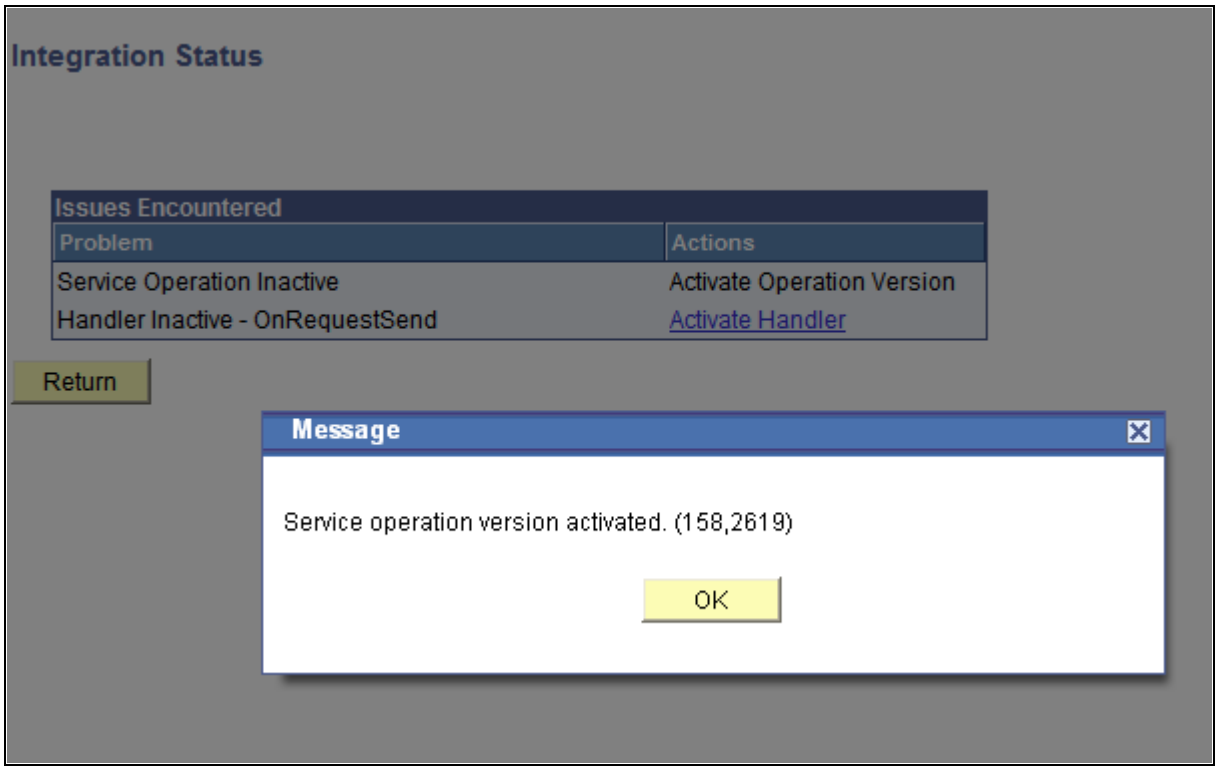
See Chapter 17, "Managing Service Operation Routing Definitions," Activating Integration Metadata Using the Integration Status Page, page 408.

Activating Integration Metadata Using the Integration Status Page

The system activates the associated metadata and displays a message when the data is successfully activated.

The following example shows the successful activation of a service operation from the Integration Status page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Successfully activating integration metadata from the Integration Status page.

To activate metadata for an integration using the Integration Status page:

1. From the Integration Broker Routing Graphic page, click the Integration Not Active link.

   The Integration Status page appears.

2. In the Actions column, click an action to complete.

   The system processes the action and displays a message that the metadata is activated.

3. Click the OK button.
4. Repeat the steps to activate additional metadata.
5. Click the Return button to go back to the Integration Broker Routing Graphic page.

Retrieving Routing Properties Programmatically

The %TransformData.routingDefnName property enables you to retrieve the routing definition name for a transaction, and allows you to retrieve the routing properties respectfully.

You use this property in a PeopleSoft Application Engine program as follows:

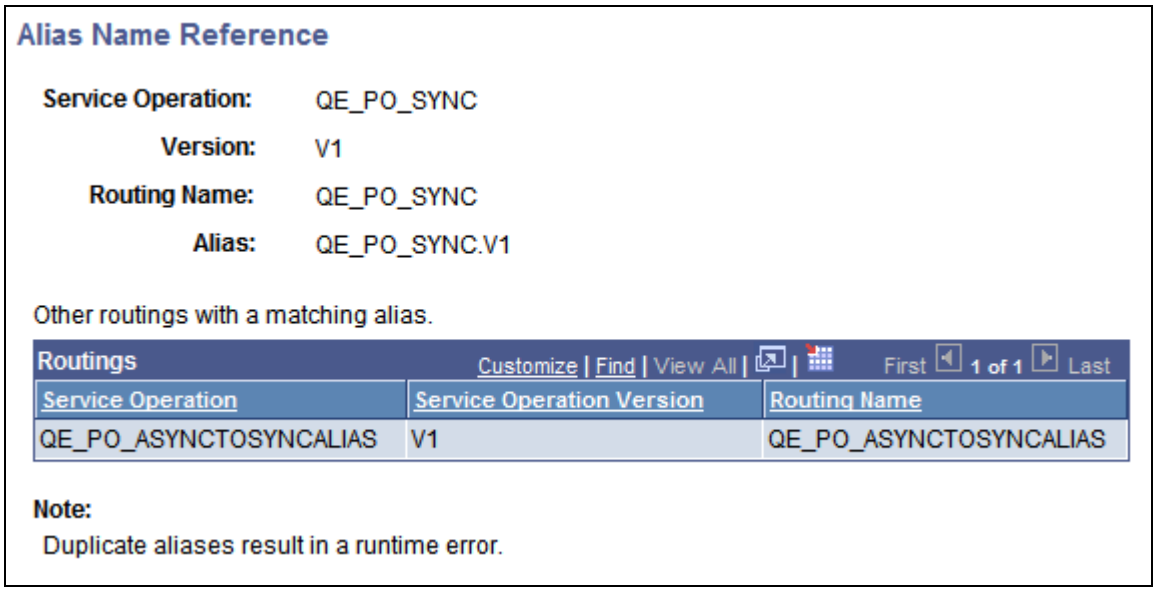
```
string rtgDefnName = %TransformData.routingDefnName
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Searching for Duplicate External Routing Aliases

The Alias Name Reference page (IB_ALIASXREF_SEC) displays duplicate external routing aliases that exist for a routing definition.

To access the Alias Name Reference page, select PeopleTools, Integration Setup, Routings. Click the Parameters tab and click the Alias Reference link for a request or response. The following graphic shows the Alias Name Reference page:



The Alias Name Reference page showing that the external routing alias QE_PO_SYNC.V1 is used for the current routing definition, QE_PO_SYNC, and for the routing definition QE_PO_ASYNCTOSYNCHALIAS.

The Alias Name Reference page appears only if the system finds that the external alias on the current definition also exists on one or more other routing definitions. If the system finds that the external alias is not used on any other routing definitions, it displays a message indicating so.

Note. The Alias Name Reference page appears only if a duplicate external alias exists on another routing definition.

Duplicate external routing alias names are never allowed for synchronous routing definitions and the system generates an error at runtime if it encounters any.

In most cases duplicate external routing alias names are not desired for asynchronous routing definitions. The exception is when you are using service operation mapping, and you want one routing definition to invoke multiple service operations.

To remedy the problem of duplicate external routing alias names, create unique alias names for each duplicate found.

To search for duplicate external routing alias names:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Select PeopleTools, Integration Broker, Integration Setup, Routings.

   The Routings Definition page appears.

2. Click the Parameters tab.

   The Parameters page appears.

3. Locate the request or response for which to check for duplicate external routing alias names.

   Click the Alias Reference link.

   If duplicate external routing alias names are found, the Alias Name Reference page appears and lists the routing definitions with which any duplicate aliases are associated. Otherwise, the system displays a message indicating that no duplicate aliases exist in the system.

Renaming and Deleting Routing Definitions

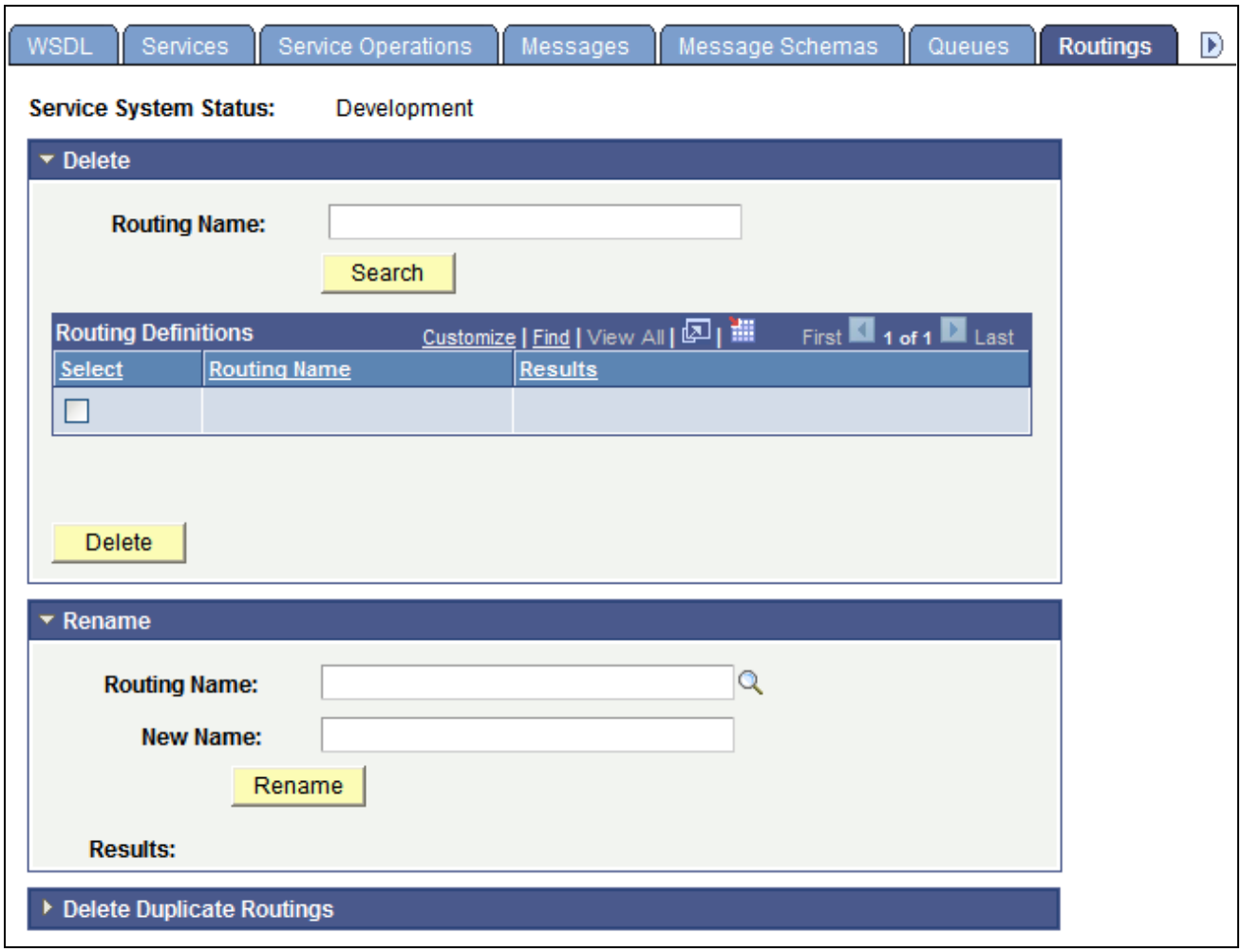
You can rename and delete routing definitions using the Routings page (IB_HOME_PAGE_4) in the Service Administration component (IB_HOME_PAGE).

The Routings tab contains three sections: a Delete section that enables you to delete routing definition, a Rename section that enables you to rename routing definitions, and a Delete Duplicate Routings section that enables you to view and delete duplicate routing definitions.

When you first access the Routings tab, the sections are collapsed. Click the section header arrow buttons to expand and collapse each section.

The following example shows the Routings tab with both Delete and Rename sections expanded:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Service Administration—Routings page

The service system status that you set on the Services Configuration page affects the ability to rename services.

This section discusses renaming and deleting routings. See the following section for information about deleting duplicate routings.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services; *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Using the Service Configuration Page to Set Service Configuration Properties and Chapter 17, "Managing Service Operation Routing Definitions," Deleting Duplicate Routing Definitions, page 413.

Renaming Routing Definitions

To rename a routing definition:

1. Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Select the Routings tab.

   The Routing page appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Click the arrow next to the Rename section header to expand the section.
3. In the Routing Name field, enter the routing definition to rename, or click the Lookup button to search for and select one.
4. In the New Name field, enter the new name for the routing definition.
5. Click the Rename button.

After you click the Rename button, the Results field displays a message that the action was successful or displays a warning or error message with a description of the problem.

Deleting Routing Definitions

To delete a routing definition:

1. Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Select the Routing tab.

   The Routing page appears.
2. Click the arrow next to the Delete section header to expand the section.
3. In the Routing Name field, enter the name of the routing definition to delete, and click the Search button.

   Search results display in the Routings grid.
4. Select the check box next to the routing definition or routing definitions to delete.
5. Click the Delete button.

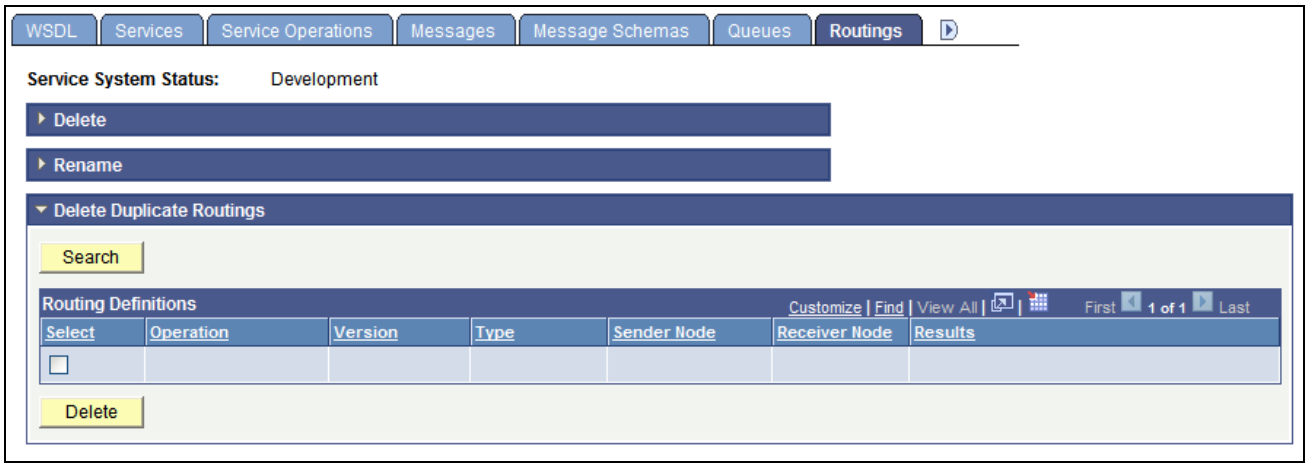
Deleting Duplicate Routing Definitions

Application upgrades and the PeopleSoft Application Designer project copy process can cause duplicate routings in the PeopleSoft system.

The Service Administration - Routings page (IB_HOME_PAGE_4) features a Delete Duplicate Routings section that enables you to search for duplicate routings in the system and delete them. When you first access the page, all sections on the page are collapsed. Click the arrow next to the Delete Duplicate Routings section title to expand the section.

The following graphic shows the Service Administration–Routings page with the Delete Duplicate Routings section expanded:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Delete Duplicate Routings section of the Service Administration–Routings page.

The page displays active duplicate routings only.

To delete duplicate routings:

1. Select PeopleTools, Integration Broker, Service Utilities, Service Administration. Click the Routings tab.

   The Routings page appears.

2. Click the arrow next to the Delete Duplicate Routings section title to expand the section.

3. Click the Search button to search the system for duplicate routing definitions.

   Duplicate routing definitions populate the Routing Definitions grid and all duplicates are selected for deletion.

4. Clear the Select box for any routing definitions you do not want to delete.

5. Click the Delete button.

The Routing Definitions grid displays up to 100 routing definitions at a time. The maximum number of rows returned at a time is 1000. Use the arrow buttons to move from page to page through the search results. If the maximum number of rows is reached, an information message appears that indicates the maximum has been reached. After you delete the routing definitions, click the Search button again to return more rows.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 18

Applying Filtering, Transformation and Translation

This chapter discusses how to:

- Define transform programs.
- Develop transforms programs using PeopleSoft Application Engine.
- Develop transforms programs using Oracle XSL Mapper.
- Invoke transform programs.
- Access transform message data.
- Rename or delete transform programs.
- Filter messages.
- Apply transformations.
- Perform data translation.
- Reject transformation programs.
- Terminate transformation programs.

Understanding Filtering, Transformation, and Translation

This chapter describes how to use filtering, transformations and translations for integrations. This chapter also features code examples to help illustrate how to filter, transform and translate integrations.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Filtering, transformation, and translation are all accomplished by applying an Application Engine *transform program* to an outbound or inbound message. You can use these programs to:

- *Filter* a message based on its content, to determine whether to pass it through to its destination or to subsequent steps of the transform program.
- Perform *transformation* on a message to make its structure comply with the receiving system's requirements.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Perform *data translation* on a message so that its data is represented according to the receiving system's conventions.

  Simple translation might be required if the two systems use different values to represent the same information for a given field.

  Complex translation might involve augmenting or replacing groups of fields with a completely different structure and encoding.

If your PeopleSoft application uses the PeopleCode XmlDoc or SoapDoc classes to generate or process a message, the message probably doesn't adhere to the PeopleSoft rowset-based message format.

Filtering, transformation, or translation can be necessary for messages sent between two PeopleSoft Integration Broker nodes, or between a PeopleSoft Integration Broker node and a third-party application.

See Also

Chapter 5, "Understanding Supported Message Structures," page 39

Understanding Transform Programs

This section provides overview information about transform programs.

Transform Programs

A transform program is a type of PeopleSoft Application Engine program. After you create a new transform application engine program, you add steps and actions to the program, and then add code to the steps and actions that performs data transformation, filtering or translation.

To develop a transform program, you must know the initial structure and possibly the content of the message with which you are working, as well as the structure (and content) of the result you want to achieve. Make sure that all participating nodes agree on a format or employ transformations to accommodate the variations from node to node.

The message data is made available to your transform program in a PeopleCode system variable after being extracted from the wrapper in which it was transmitted. The format of this wrapper depends on the transmission method, but is irrelevant to the transform program.

Any participating node with PeopleSoft Integration Broker installed — the source, the target, or a hub — can apply a transform program to a given message.

You specify which transform program to apply within a routing definition for a service operation.

Note. With PeopleSoft Integration Broker, the term *node* refers to a system or application participating in an integration, but in this chapter a *node* is also a structural element in an XML document. The context in which the term is used should make its meaning clear.

Transform programs cannot modify the following messaging features:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Transmission protocols.

  You handle a given protocol by selecting an appropriate target connector for the target node's local gateway, or by directing a third-party sender to the appropriate listening connector on the default local node's local gateway. You can select from the delivered connectors or develop new ones.

- Character set encoding.

  This is handled by the PeopleSoft globalization system.

Use a transformation when one node sends a request or response message with a data structure different from the structure required by the other node. One or both of the participating nodes can be PeopleSoft applications. At either end of the transaction, any of the following structure types may be required:

- The PeopleSoft rowset-based message format.
- An XML DOM-compliant non-rowset-based structure. This is generic XML data.
- A SOAP-compliant XML structure. This is also XML DOM-compliant.
- A non-XML structure. Third-party applications are more likely than PeopleSoft applications to require this type.

Your transformation can be between different structure types or between different structures of the same type.

Transformation Programming Languages

You can use PeopleCode or Extensible Stylesheet Language Transformation (XSLT) as programming languages for creating transformation logic.

XSLT is a well-recognized standard language perfectly suited to manipulating XML structures, so it's highly recommended for transformations. Because of its straightforward template-based approach to accessing the codeset repository, XSLT is also recommended for data translation.

Note. When programming using XSLT, you can hand-code the XSLT or use the Oracle XSL Mapper to graphically associate records and fields. The Oracle XSL Mapper then automatically generates the XSLT code.

However, you cannot use XSLT to filter messages based on content, so filtering must be implemented in PeopleCode.

Note. Filtering must be implemented using PeopleCode

You can use both XSLT and PeopleCode steps in a single transform program.

Each XSLT program must be enclosed in the following wrapper:

Note. When using Oracle XSL Mapper, the mapper automatically encloses the program in this wrapper.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   your_xslt_program
</xsl:stylesheet>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Third-party XSLT development tools may generate a wrapper that specifies a different URI. Make sure the URI in your program is exactly as shown here, or your program may not work as expected.

You can find more information about XSLT at the World Wide Web Consortium (W3C) web site.

Third-Party Considerations

When no transformation is applied, applications using PeopleSoft Integration Broker send, and expect to receive, messages containing data that conforms to a minimum XML message structure: the PeopleSoft rowset-based message format.

When exchanging messages with third-party applications, you can:

- Employ a transformation at the PeopleSoft end of each transaction to convert messages to or from the PeopleSoft rowset-based message format.
- Require third-party applications to send and receive messages that comply with the PeopleSoft rowset-based message format. Third-party applications must comply with the rowset-based message format if *both* of the following are true:
    - Your PeopleSoft application uses the PeopleCode Message and Rowset classes to generate and send, or receive and process messages with Integration Broker.
    - You don't want to employ PeopleSoft Integration Broker transformations to accommodate the third-party application.

Note. Third parties can submit messages to PeopleSoft Integration Broker systems using any listening connector registered with the local gateway. Regardless of the message data format, the third-party system is responsible for properly constructing the wrapper that contains the message data, and using the appropriate tools and protocols to address the connector.

Defining Transform Programs

This section discusses how to define transform programs.

Understanding Defining Transform Programs

This section contains information about defining transform programs

Transform Program Type

To create a transform program, in the Program Properties dialog box for the application engine program you must specify that the program type is *Transform Only.* After you select this option, input message name, output message name, input root element and output root element fields display.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Input and Output Message Names

When developing a transformation program, PeopleSoft Integration Broker enables you to specify the schema of the message that is going into the transform (input message and version), as well as the schema of the message that is the output of the program (output message and version).

Note. You must specify this information when using the Oracle XSL Mapper to develop transformation programs.

These fields are required when developing transformations using the Oracle XSL Mapper.

In all other cases, these fields are optional, since there may be occasions where a transformation is general in nature and used by many messages. For example, it might be a transform that changes certain fields regardless of the message shape. In cases such as these, you would not want to define a specific input or output shape, since the transform program is only changing fields.

Input and Output Root Elements

When working with nonrowset-based messages, there may be situations where the schemas for input and output messages have multiple root elements. However, Oracle XSL Mapper uses only one of the root elements on the input side as well as only one on the output side.

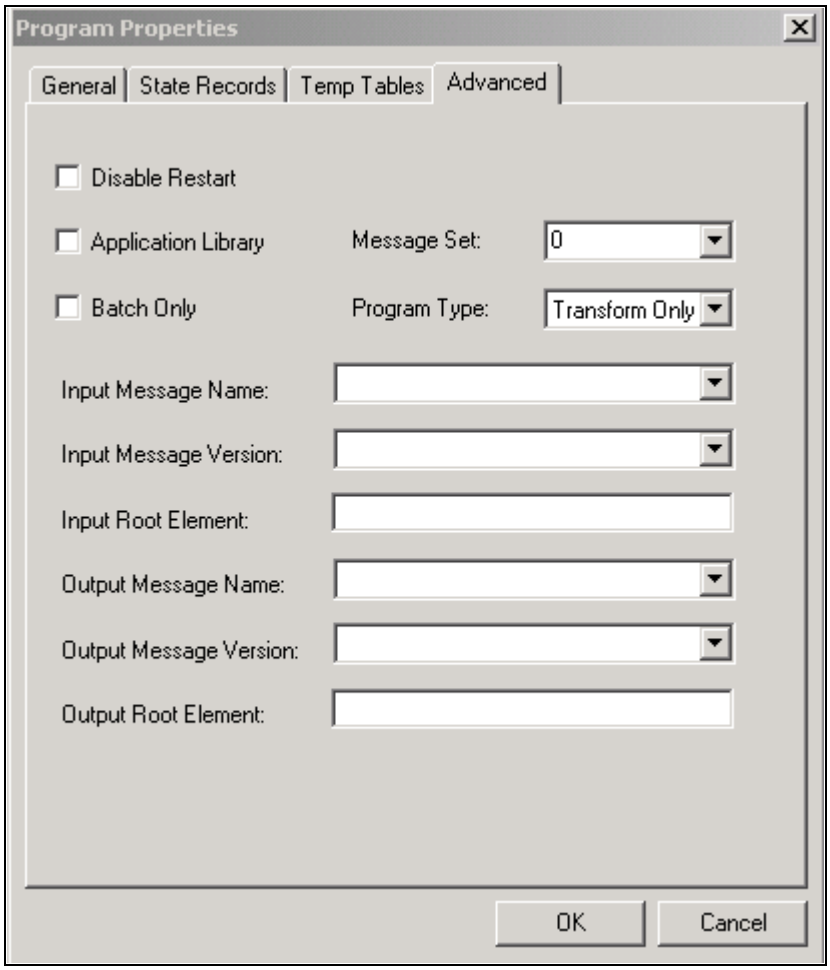
When using Oracle XSL Mapper to build XSLT transformations, you may specify the input and output root elements in the Program Properties dialog box. If you do not specify an input or output root element, Oracle XSL Mapper uses the first root element in the schema.

Defining a Transform Program

To define a transform program, create a new application engine object in PeopleSoft Application Designer. Then in the Program Properties dialog box you specify the program type as a transform program.

The following graphic shows the Program Properties dialog box for an application engine program. Note that the Program Type field displays *Transform Only*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Program Properties — Advanced tab

To define a transform program:

1. In PeopleSoft Application Designer, select File, New, App Engine Program and click the OK button.

   A new application engine program window appears.

2. On the toolbar, click the *Properties* button.

   The Program Properties dialog box appears.

3. Click the Advanced tab.

4. From the Program Type drop-down list box, select *Transform Only*.

   Additional fields relating to input messages, output messages and root elements appear.

5. Select an input message and version:

   a. From the Input Message Name drop-down list box, select the name of the message before transformation is applied.

   b. From the Input Message Version drop-down list box, select the version of the input message.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

6. In the Input Root Element field, enter the name of the input schema root element to use.

   Enter a value in this field if the input message has multiple root elements. If the input message has multiple root elements and you do not enter an input root element, the first root element in the message is used for transformation.

   This field is disabled when the input message is a rowset-based message.

7. Select an output message and version.

   a. From the Output Message Name drop-down list box, select the name of the message after transformation is applied.

   b. From the Output Message Version drop-down list box, select the version of the output message.

8. In the Output Root Element field, enter the name of the output schema root element to use.

   This field is disabled when working with rowset-based messages.

9. Click the OK button.

10. The Program Properties dialog box closes.

11. Select File, Save.

Developing Transform Programs Using PeopleSoft Application Engine

This section discusses how to:

- Insert steps and actions into transform programs.
- Work with transform programs.
- Access message data.
- Make working data available globally.
- Preserve record and field aliases.

Understanding Developing Transform Programs Using PeopleSoft Application Engine

Following are some points to keep in mind when working with transform programs:

- Each transform program step operates on the message content that results from the previous step, so you can break your transform program into a sequence of discrete steps.
- Multiple transform actions within a step can produce unwanted effects, so insert each XSLT or PeopleCode action in its own step.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- XSLT works only on XML DOM-compliant data, so PeopleSoft Integration Broker assures that both outbound and inbound messages are in XML DOM-compliant form when transform programs are applied to them.
- XSLT is not supported by PeopleSoft on the OS/390 or z/OS operating systems. Transformations must use PeopleCode on these platforms.

A transformation can modify an entire message until it no longer resembles the original. So can a data translation. In a transformation, you must hard-code what you want to accomplish, whereas the data translation relies on a repository of *codeset metadata* that you define. This means you can establish consistent rule-based translations and reuse the same translation rules without having to reenter them.

Although you can combine transformation and data translation in a single transform step, it's better to keep these processes in separate steps and produce a modular program that you can more easily maintain, with code you can reuse in other transform programs.

Inserting Steps and Actions into Transform Programs

This section describes how to insert steps and actions into transform programs.

Understanding Inserting Steps and Actions

After you define a transform program, you insert steps and actions as you would with any other application engine program to construct the transformation program.

The two types of actions you can add to steps when building a transform program are *XSLT* and *PeopleCode*.

Inserting XSLT Actions

When you select *XSLT* as the step action type you can develop the transform code using Oracle XSL Mapper or you can hand-code the program.

Installing, configuring and using Oracle XSL Mapper to develop transform programs is discussed elsewhere in this chapter.

See Chapter 18, "Applying Filtering, Transformation and Translation," Developing Transforms Using Oracle XSL Mapper, page 425.

Note. After selecting XSLT as the action type, you must save the program before you can choose to use Oracle XSL Mapper or hand-code the program.

To insert XSLT actions:

1. From the Action Type drop-down list, select *XSLT*.
2. (Optional.) In the XSLT Description field, enter a description for the XSLT action.
3. Save the transform program.

   A Graphical Mapper drop-down list appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Choose how to code the XSLT action:

   - To use Oracle XSL Mapper , click the XSLT action to highlight it. Then double-click the action to launch the mapper tool.
   - To hand-code the program, from the Graphic Mapper drop-down list box, select *No.* Right-click the action and select View XSLT.

5. Create code for the step/action.

   - If using Oracle XSL Mapper, beginning mapping records and fields as appropriate.
   - If hand-coding using XSLT, right-click the action and select View XSLT. The programming window appears and you can begin coding.

Inserting PeopleCode Actions

To insert PeopleCode actions:

1. From the Action Type drop-down list, select *PeopleCode.*
2. (Optional.) In the PeopleCode Description field, enter a description for the PeopleCode action.
3. Save the program.
4. Right-click and select View PeopleCode to open the programming window.
5. Enter PeopleCode appropriate for the step and action.

Making Working Storage Data Available Globally

XSLT transform steps can't access external data, but PeopleCode can. XSLT also has no global variables. However, the message itself is global, and can be used to pass working or external data to all steps in the transform program. During a PeopleCode step, you can add a special node to the message to contain the data, which is then available to subsequent transform steps.

Following is an example of a minimal input message:

```
<?xml version="1.0"?>
<Header>
  <LANGUAGE_CODE>en_us</LANGUAGE_CODE>
  <STATUS_CODE>1000</STATUS_CODE>
</Header>
```

The following PeopleCode inserts a node in the message to contain working data, by convention called *psft_workingstorage.* Then the PeopleCode inserts the current system date into that node:

```
/* Get the data from the AE Runtime */
Local TransformData &incomingData = %TransformData;
/* Set a temp object to contain the incoming document */
Local XmlDoc &inputDoc = &incomingData.XmlDoc;
/* Add a working storage node*/
Local XmlNode &wrkStorageNode =
 &inputDoc.DocumentElement.AddElement("psft_workingstorage");
/* Add the current system date to the working storage*/
Local XmlNode &sysDateNode = &wrkStorageNode.AddElement("sysdate");
&sysDateNode.NodeValue = String(%Date);
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Following is the resulting output message:

```
<?xml version="1.0"?>
<Header>
  <LANGUAGE_CODE>en_us</LANGUAGE_CODE>
  <STATUS_CODE>0</STATUS_CODE>
  <psft_workingstorage>
    <sysdate>2002-01-24</sysdate>
  </psft_workingstorage>
</Header>
```

Any subsequent transform step now has access to the current system date. Make sure the last step that uses the *psft_workingstorage* node removes it from the final output, as with this XSLT fragment:

```
<xsl:template match="psft_workingstorage">
   <!-- Do not copy this node -->
</xsl:template>
```

Preserving Record and Field Aliases

When you apply a transform program to a rowset-based message, Integration Broker submits the message to the program in its final XML DOM compliant form, with any aliases you defined in place of the corresponding original record and field names.

In a PeopleCode transformation, the message is initially available as an XmlDoc object. However, you may want to transform the message using the PeopleCode Rowset class. Because XmlDoc object structure is compatible with Rowset object structure, you can copy the XML data to a rowset using the XmlDoc class CopyToRowset method.

Because the rowset to which you copy the data must be based on a message object instantiated from your original message, the rowset uses the message's original record and field names. If you defined aliases for any of the message records or fields, you must ensure that the rowset uses those aliases instead, or the XmlDoc data won't copy successfully.

The following set of conditions summarizes this situation:

- The message definition includes at least one record or field alias.
- You're applying a transform program to the message.
- Your transform program includes a PeopleCode step.
- The PeopleCode step uses a Rowset object to hold the message data.

Using Optional CopyToRowset Parameters

To make sure the rowset object uses the record and field aliases that exist in the XML data, you must specify two optional string parameters in the CopyToRowset method, which convey the message name and version:

```
CopyToRowset(&Rowset, Message_Name, Version)
```

The integration engine uses any aliases it finds in the specified message definition to rename the appropriate records and fields in the rowset object before copying the data. Following is an example of a rowset-based transform step that preserves aliases:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
Local Message &TempMSG;
Local Rowset &TempRS;

/* Get the data from the AE Runtime */
Local TransformData &tempData = %TransformData;
/* Set a temp object to contain the incoming document */
Local XmlDoc &tempDoc = &tempData.XmlDoc;

/* Create a working rowset (no aliases used) */
&TempMSG = CreateMessage(Message.MyMsgName);
&TempRS = &TempMSG.GetRowset();

/* Copy message data to rowset (restoring aliases) */
&OK = &tempDoc.CopyToRowset(&TempRS, "MY_MSG_NAME", "MY_MSG_VERSION");

   /* . . .Transform rowset data. . . */

/* Copy transformed rowset back to XmlDoc object */
&OK = &tempDoc.CopyRowset(&TempRS, "MY_MSG_NAME", "MY_MSG_VERSION");
```

See Also

PeopleTools 8.52: PeopleCode API Reference, "XmlDoc Classes," XmlDoc Class Methods

Tracing Transform Programs

For debugging purposes, you can trigger a trace of your transform program by adding a specific value to the Application Engine *trace* parameter, in one of the following ways:

- Specify the TRACE switch on the Application Engine command line, with the value *8192* added, for example:

  ```
  -TRACE 8192
  ```

- Add the value *8192* to the TRACEAE parameter in the appropriate application server or Process Scheduler server configuration file, for example:

  ```
  TRACEAE=8192
  ```

See Also

PeopleTools 8.52: PeopleCode API Reference, "XmlDoc Classes"

PeopleTools 8.52: Application Engine, "Tracing Application Engine Programs"

PeopleTools 8.52: PeopleSoft Process Scheduler, "Using the PSADMIN Utility"

http://www.w3.org/Style/XSL/

Developing Transforms Using Oracle XSL Mapper

This section discusses how to develop transformations using Oracle XSL Mapper.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Oracle XSL Mapper

Oracle America has developed a tool that enables you to graphically map records and fields and that creates the underlying XSLT transformation code for you. You launch this tool directly from a PeopleSoft application engine transformation program. When you save the XSLT code in Oracle XSL Mapper, it automatically gets saved to the application database and your application engine transform program.

Note. You cannot use Oracle XSL Mapper to modify XSLT that you've hand-coded or created with any other XSLT editing tool.

Oracle XSL Mapper is an Oracle JDeveloper plug-in. Check the My Oracle Support website for information about supported versions.

See http://www.oracle.com.

Development Considerations

Note the following as you develop transformations using Oracle XSL Mapper:

- When you save XSL maps that you create in the mapper, the underlying XSL code is automatically saved to the application engine program.
- The mapper does not support codesets and working storage constructs. You must add these constructs manually into the XSL code using the Source view of the mapper.

Prerequisites for Developing Transforms Using Oracle XSL Mapper

To use Oracle XSL Mapper you must:

- Install Oracle JDeveloper.

  Oracle XSL Mapper is part of Oracle JDeveloper .

  See Chapter 18, "Applying Filtering, Transformation and Translation," Installing Oracle XSL Mapper, page 427.

- In PeopleSoft Configuration Manager, specify the path to the Oracle JDeveloper installation location.

  See Chapter 18, "Applying Filtering, Transformation and Translation," Specifying the Installation Path and Classpath for Oracle XSL Mapper, page 427.

- All messages used in the mapper must have schemas generated for them.

  For rowset-based messages, use the Message Definitions–Schema page to generate schemas. For nonrowset-based messages, use the Message Definitions–Schema page to add or import schemas for these types of messages.

  See Chapter 6, "Managing Messages," Managing XML Message Schemas for Rowset-Based Messages, page 97 and Chapter 6, "Managing Messages," Adding XML Message Schemas to Nonrowset-Based Messages, page 100.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- You must create a Transform Only application engine program and define the program properties described earlier in this chapter.

  See Chapter 18, "Applying Filtering, Transformation and Translation," Defining a Transform Program, page 419.

- If you are using Oracle JDeveloper 11g, the JDK folder and its contents must exist inside the JDeveloper home location. Therefore you must install JDK 1.6 or use an existing installation of JDK 1.6, and copy all the contents inside the <JDEV_HOME> directory. The following example shows the installation path: <JDEV11_HOME>/jdk/<jdk_copied_contents>

- If you are using Oracle 11gR2 or a later release, you must move the following folders from <Middleware_Home>\oracle_common\modules to <Middleware_Home>\jdeveloper\modules:

  - \oracle.xdk_11.1.0
  - \oracle.xmlef_11.1.1
  - \oracle.javatools_11.1.1

Installing Oracle XSL Mapper

Oracle XSL Mapper is a tool that provides the ability to rapidly create XSL-based transformation maps to support integration of PeopleSoft to third-party applications deployed through Integration Broker. The Oracle XSL Mapper is a component of Oracle BPEL Designer and is included in Oracle JDeveloper. Access to this feature requires the download of Oracle JDeveloper at the location below.

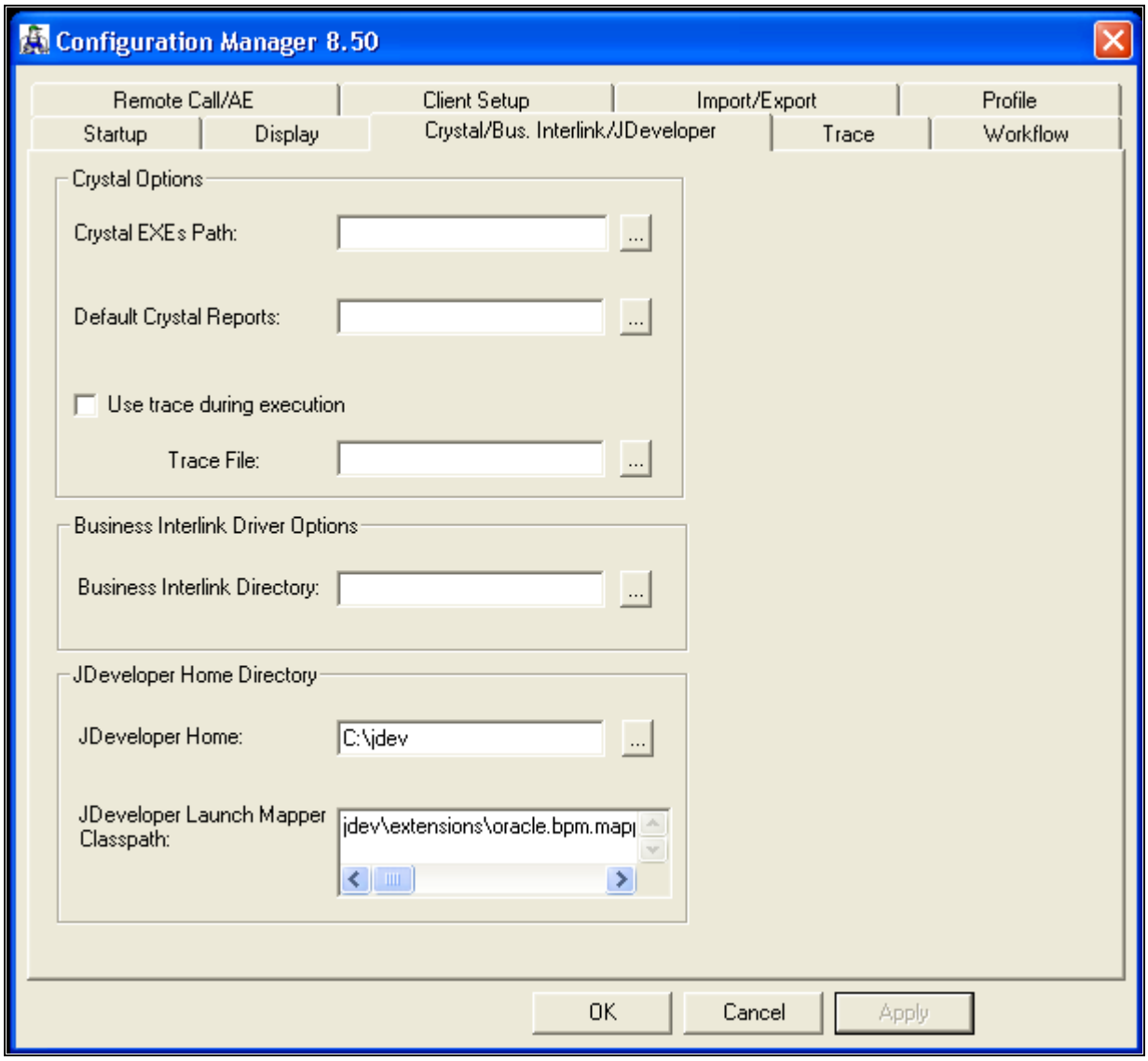
See http://www.oracle.com/technology/index.html.

Note. After you install Oracle XSL Mapper do not move or delete the installed jar files from the default installation location. The PeopleSoft system reads this information from the default location for proper functioning of the feature.

Specifying the Installation Path and Classpath for Oracle XSL Mapper

For Oracle XSL mapper to function, the JDeveloper installation location and classpath must be specified in the PeopleSoft system in the PeopleSoft Configuration Manager. You enter this information on the Crystal/Bus. Interlinks/JDeveloper tab shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



The JDeveloper <Home> location and classpath defined in PeopleSoft Configuration Manager

To specify the path to the Oracle XSL Mapper installation location and classpath:

1. Open PeopleSoft Configuration Manager (pscfg.exe).
2. Click the Crystal/Bus. Interlink/JDeveloper tab.
3. Locate the JDeveloper Home Directory section at the bottom of the page.
4. In the JDeveloper Home field, enter the path or browse to the location where JDeveloper is installed.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Click the Apply button.

   If you are using JDeveloper version 10.1.3 or 10.1.2, Configuration Manager populates the JDeveloper Launch Mapper Classpath field with the appropriate variables for your version of JDeveloper.

   If you are using JDeveloper version 11.1.1 enter the following classpath in the JDeveloper Launch Mapper Classpath field:

   ```
   jdev\extensions\oracle.bpm.mapper.jar;integration\lib\bpm-ide-common.jar;
   ide\lib\javatools.jar;jdev\lib\xmleditor.jar;modules\oracle.xdk_11.1.0\
   xmlparserv2.jar;modules\oracle.xmlef_11.1.1\xmlef.jar;modules\
   oracle.javatools_11.1.1\javatools-nodeps.jar
   ```

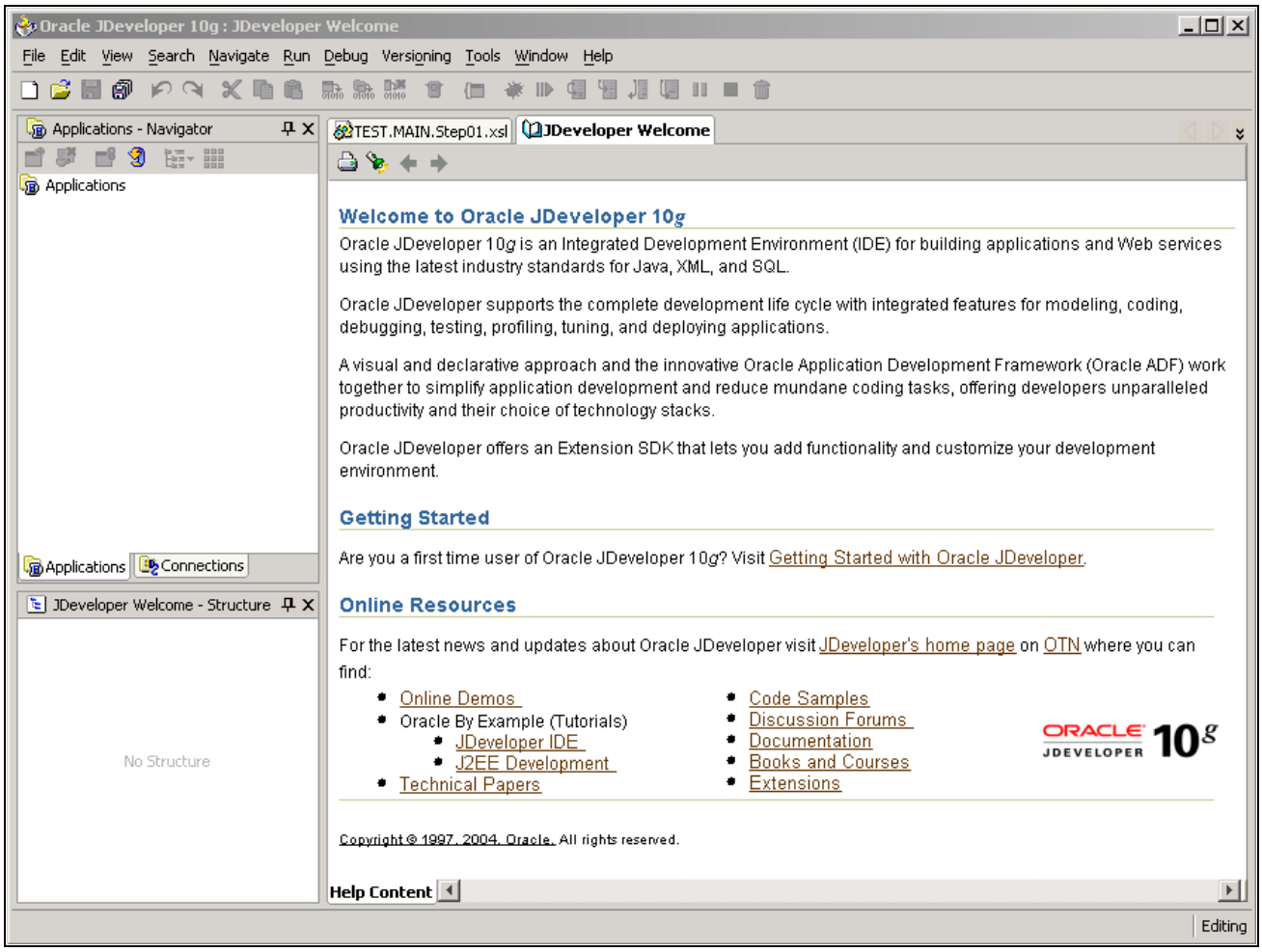
6. Click the OK button.

Launching Oracle XSL Mapper

You launch Oracle XSL Mapper from within an application engine transform program.

The first time you launch the mapper, the Oracle JDeveloper Welcome window appears.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

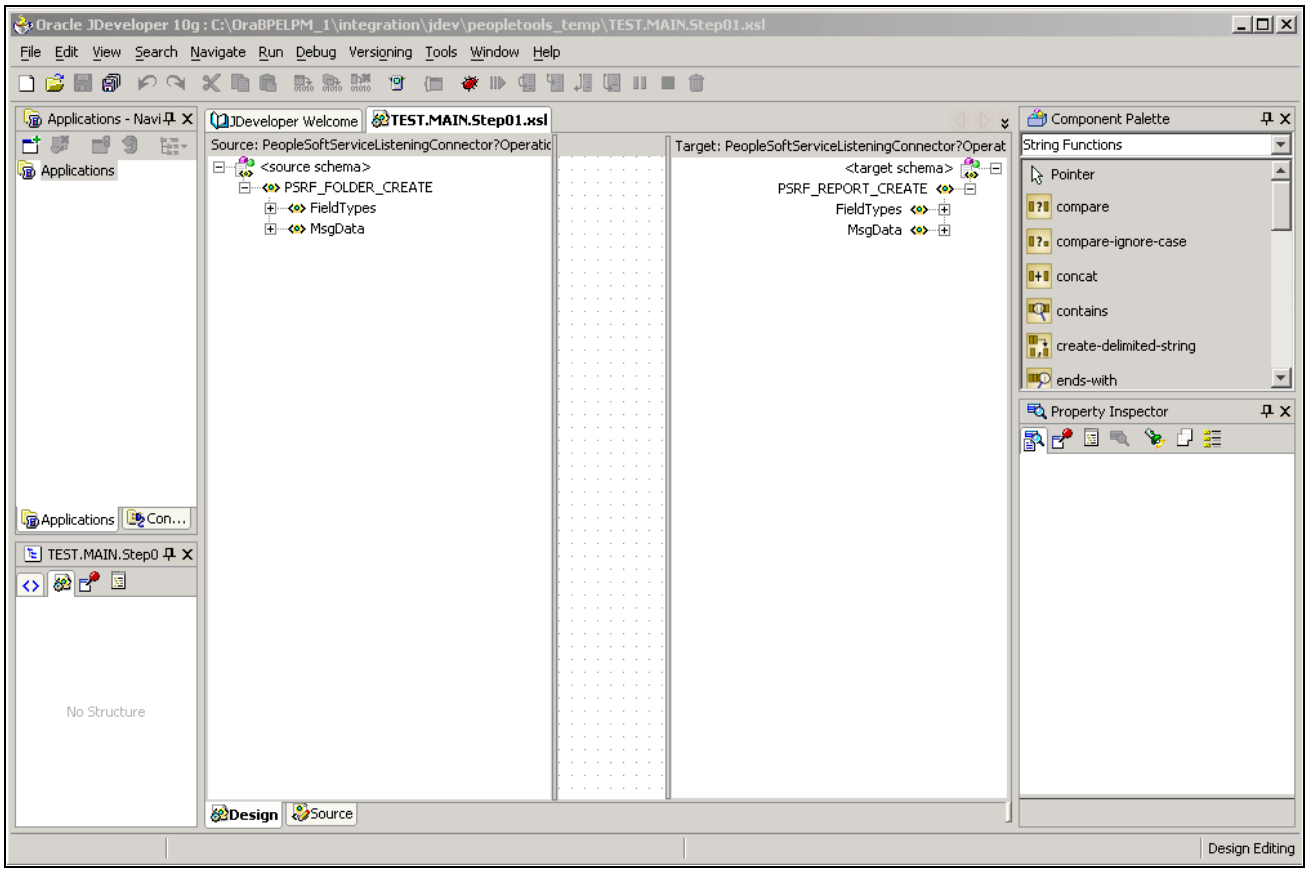


JDeveloper Welcome window

Note. The first time you launch Oracle XSL Mapper a Configure Tile Type Associations dialog box appears. While using the mapper you do not work with any Java files and you can disregard the dialog box.

The Oracle JDeveloper Welcome window displays only the first time you access JDeveloper. When you subsequently open Oracle XSL Mapper, the transform program appears in the Design view.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Oracle XSL Mapper–Design view

The transform program name in the mapper takes the following format:

`<transform_program_name>.<section_name>.<step_name>.xsl`

To launch Oracle XSL Mapper:

1. Create a *Transform Only* application engine program and define the program properties described earlier in this chapter.
2. Double-click the XSLT action.
3. Access the Oracle XSL Mapper–Design view.

   When accessing the mapper for the first time, when you double-click the XSLT action and launch Oracle JDeveloper the JDeveloper Welcome window appears. To switch to the Design view, above the Source pane, click the transform file name tab or from the Window menu click the transform file name.

   In subsequent attempts to launch the mapper, double-clicking the XSLT action automatically opens the transform program in the Design view.

Accessing Oracle JDeveloper Documentation and Online Resources

This section provides information for access online Help, documentation and other resources for using Oracle XSL Mapper.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Online Help

Online Help is available via the Help menu while working with Oracle XSL Mapper.

Additional Oracle JDeveloper Documentation and Online Resources

As mentioned earlier in this section, Oracle XSL Mapper is a plug-in to Oracle JDeveloper. Documentation for the mapper is contained in the Oracle JDeveloper documentation set.

Note. The information provided in this section is current as of the publish date of this PeopleBook.

On the Developer Welcome page of the Oracle XSL Mapper there are links to documentation and online resources located on the Oracle web site.

In addition you can visit the Oracle Technology Network website to access documentation. Oracle JDeveloper documentation resources that may be helpful include:

- *Oracle Application Development Framework Guidelines Manual* .
- *Oracle Application Development Framework Case Manual.*
- *Installation Guide.*
- *Release Notes.*
- *J2EE Developer Online Help (the main documentation library)*

Navigating in Oracle XSL Mapper

This discusses how to:

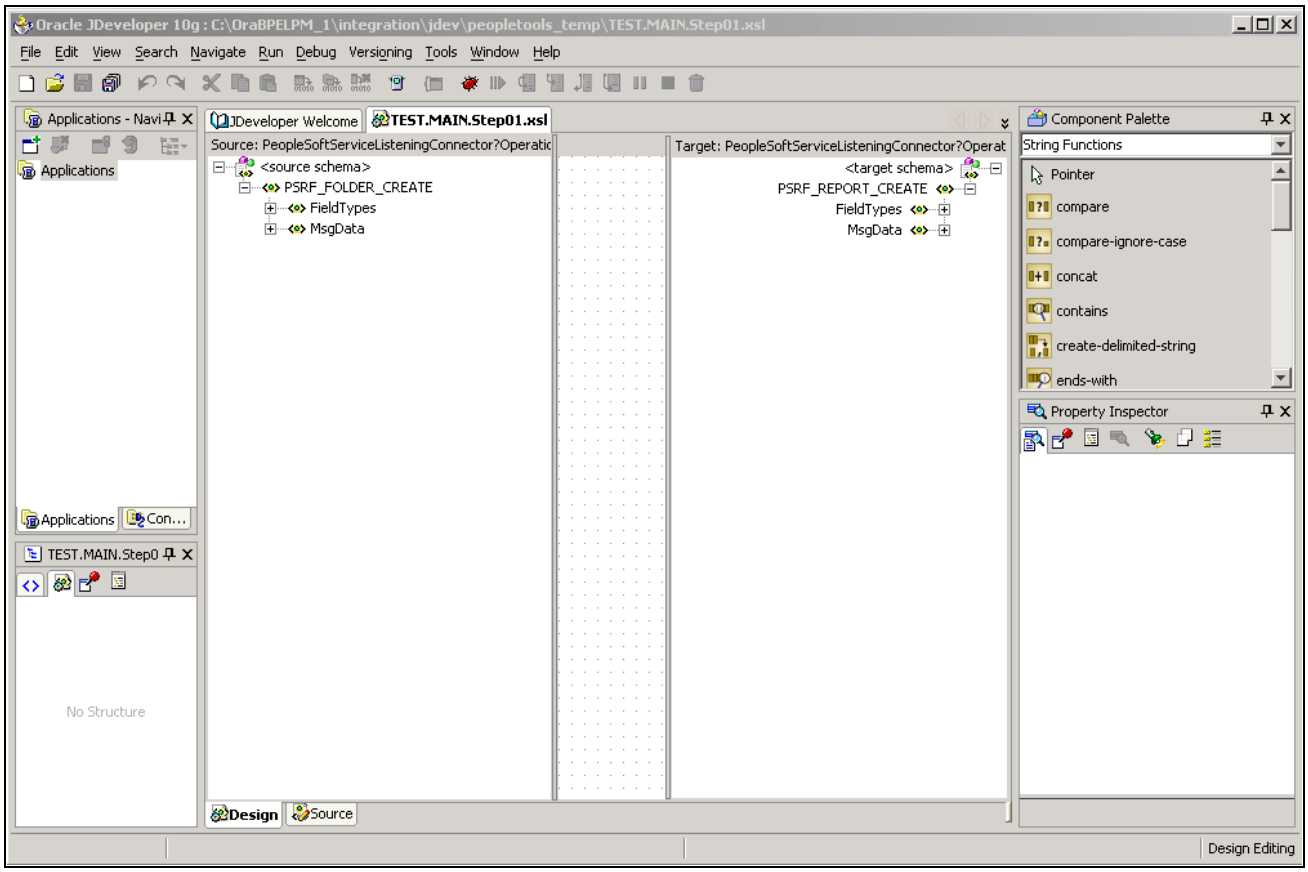
- Navigate in the Design View.
- Navigate in the Source View.

Note. This section features a brief discussion of some of the key components and areas of the Oracle XSL Mapper development tool. The Oracle JDeveloper documentation provides in-depth details about using the Oracle XSL Mapper.

Navigating in the Design View

To access the Design view of Oracle XSL Mapper, click the Design tab at the bottom of the mapper. The following graphic shows the Oracle XSL Mapper – Design view:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Oracle XSL Mapper – Design view

Use the Design view to map records and fields.

A few of the key areas of the Design view are discussed here. For additional information on Design view features, see the Oracle XSL Mapper documentation.

Title Bar	The title bar displays the full path to the Oracle JDeveloper installation location and the name of the current transform program.
Developer Welcome tab	Click the tab to display links to online resources, such as documentation, online demos and code samples.
Transform Program name tab	Click to access the transform program.
Source	The Source pane in the Oracle XSL Mapper main development view provides a hierarchical view of the source or input message and schema. Click the plus (+) button and the minus (-) button to expand and collapse data shown. Drag the edges of the pane in or out to adjust the viewing area.

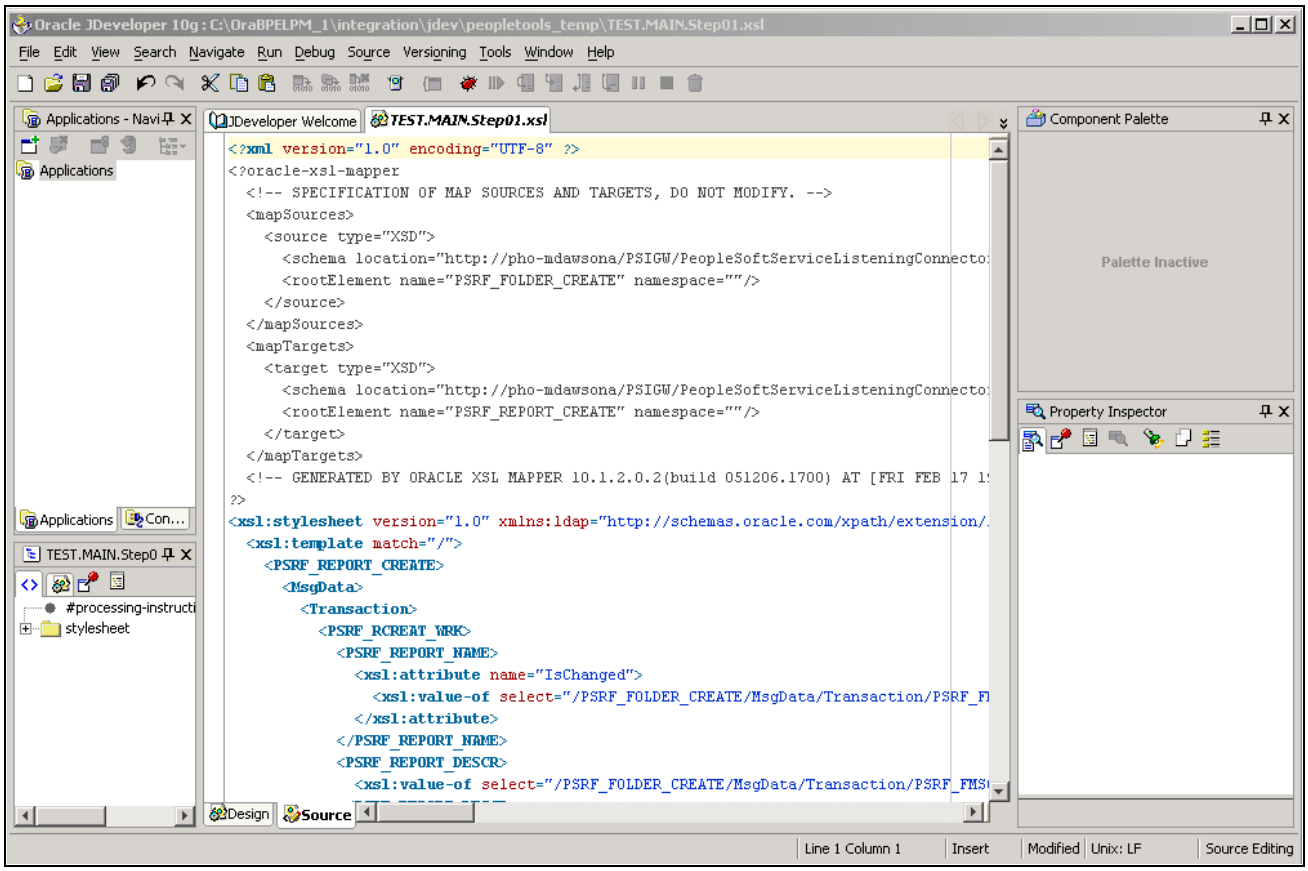
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Target	The Target pane in the Oracle XSL Mapper main development view provides a hierarchical view of the target or output message and schema. Click the plus (+) button and the minus (-) button to expand and collapse data shown. Drag the edges of the pane in or out to adjust the viewing area.
Design tab	Located at the bottom left-side of the window, click the tab to display the area where you map records and fields.
Source tab	Located at the bottom left-side of the window, click the tab to view and edit the raw XSLT code generated by Oracle XSL Mapper.

Navigating in the Source View

To access the Source view of Oracle XSL Mapper, click the Source tab at the bottom or the mapper. The following graphic shows the Oracle XSL Mapper–Source view:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Oracle XSL Mapper–Source view

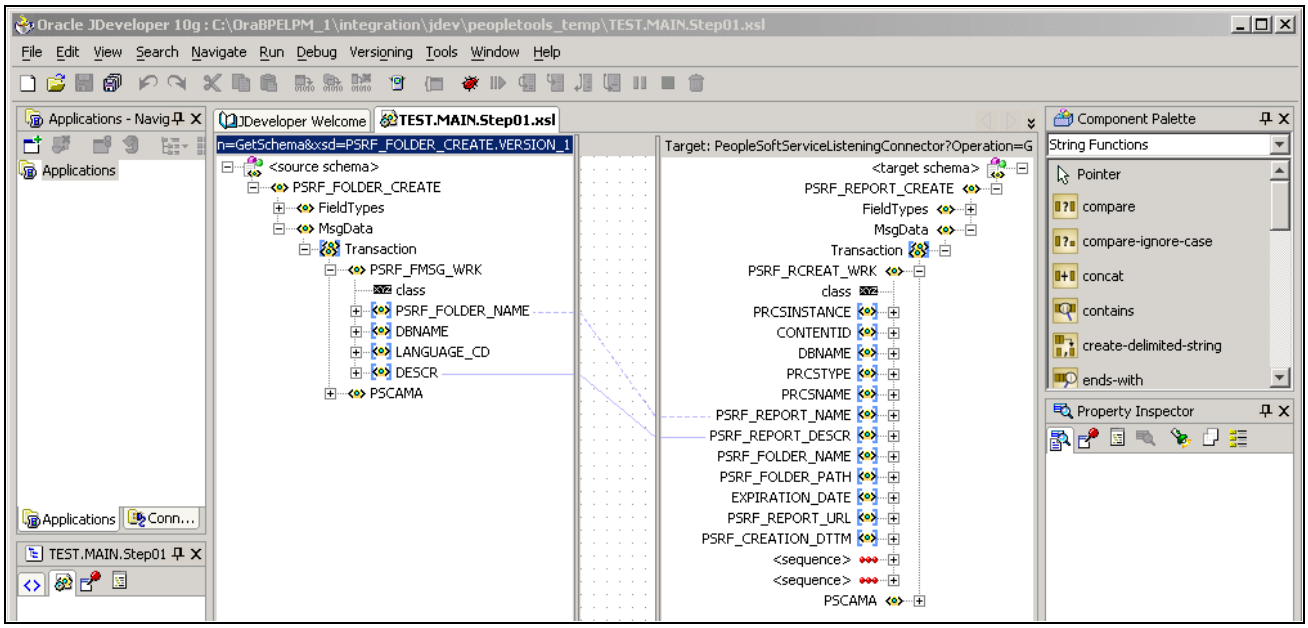
Use the Source view to view and edit the raw XSL code generated by the mapper.

For additional information on Source view features, see the Oracle XSL Mapper documentation.

Mapping Records and Fields

To map records and fields, you drag record names and field names from the source pane to the target pane in the mapper. The following graphic shows and example of mapping several fields:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Mapping fields in Oracle XSL Mapper–Design view

A solid green line appears as you drag the cursor from the source pane to the target pane. When you release the cursor, the line turns blue to show the association.

To map records and fields:

1. Open the Design view of Oracle XSL Mapper.
2. Expand the contents of both the source and target panes.

   Click the plus (+) button to expand each level or section until all records and fields appear.

   Note. When working with rowset-based messages the content to map is located in the MsgData section.

   ---

3. In the source pane, click on the icon to the left of a record or field name and drag it to the name of the record or field in the target pane to which you want to map.

   Repeat this step for each record or field to map.
4. Click the Save button.

Deleting Record and Field Maps

If you have not saved the program and want to delete the last map you made, simply right click in the Design view and select Undo Link. Otherwise, follow the instructions in this section.

To delete a record or field map:

1. In the source pane, click the record or field name.
2. From the Edit menu, click Delete.
3. Save the changes.

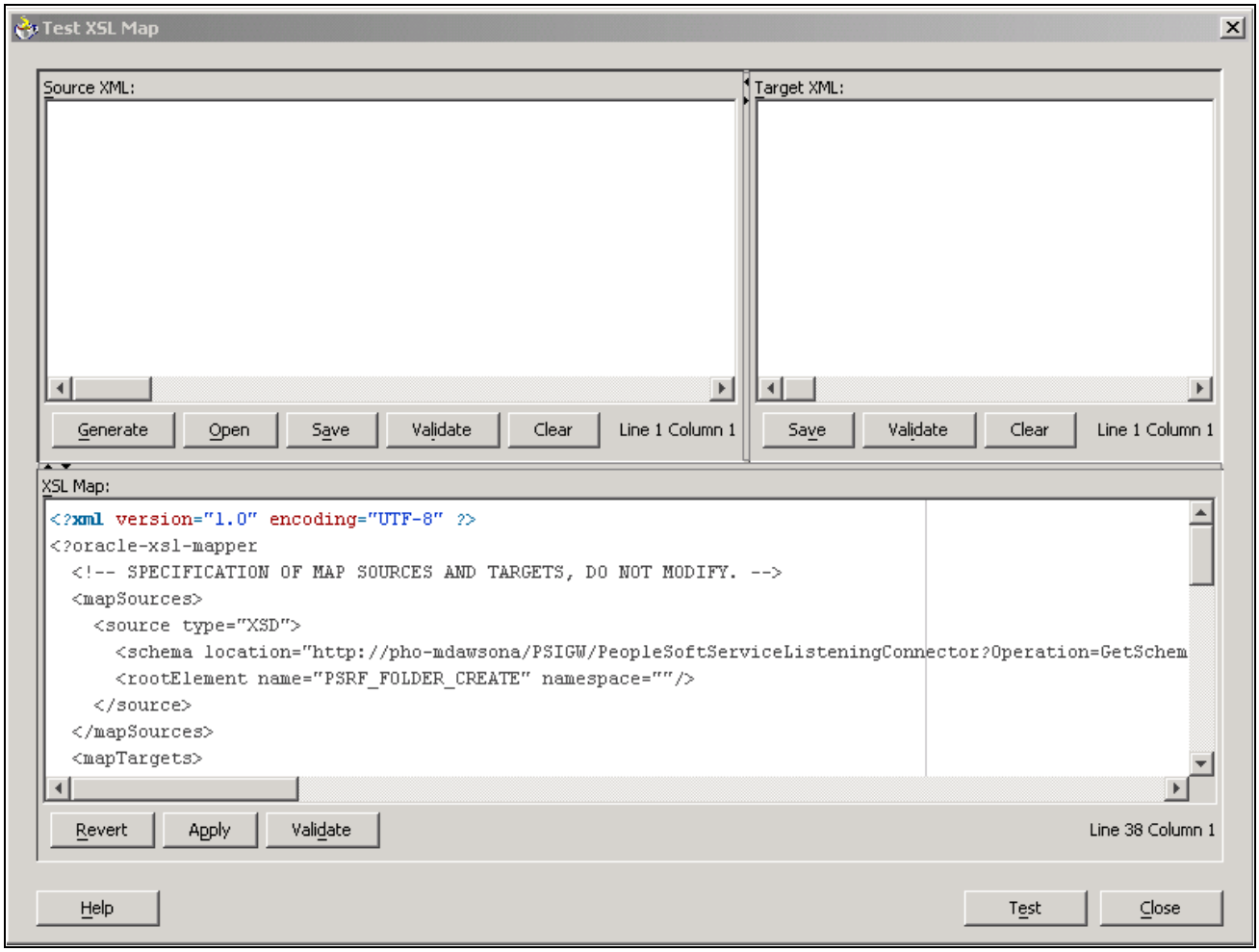
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Viewing Raw XSLT Code

After you have made all of the record and field maps for the program. Click the Source tab at the bottom of the window to generate and view the raw XSL.

Testing XSL Maps

You can test XSL maps for validity that you generate in the mapper using the Test XSL Map window shown in the following graphic:



Test XSL Map window

The code created by the record and field mapping displays in the bottom portion of the window.

Note. This Oracle XSL Mapper test feature tests the validity of the XSLT map code. It does not test the transformation. To test a transformation, use the Transformation Test tool in the PeopleSoft Pure Internet Architecture.

To test XSLT code:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Oracle XSL Mapper–Source view.
2. Right-click anywhere in the code area and select Test.

   The Test XSL Map window appears.
3. In the XSL Map section of the window, click the Validate button.

   A validation success or error message displays.

See Also

PeopleTools 8.52: Integration Broker Testing Utilities and Tools, "Using the Transformation Test Utility"

Adding and Modifying XSL Map Code

If you add code to XSLT that you create using Oracle XSL Mapper that contains any PeopleSoft-related elements it will not compile successfully, unless you create a configuration file of PeopleSoft elements that the mapper should ignore. After you create this file specify the file location and name in the Oracle XSL Mapper Preferences.

Creating Elements to Ignore Configuration Files

For example, you could add code that contains the following elements:

```
<element name = "psft_function">
<element name = "psft_workingstorage">
```

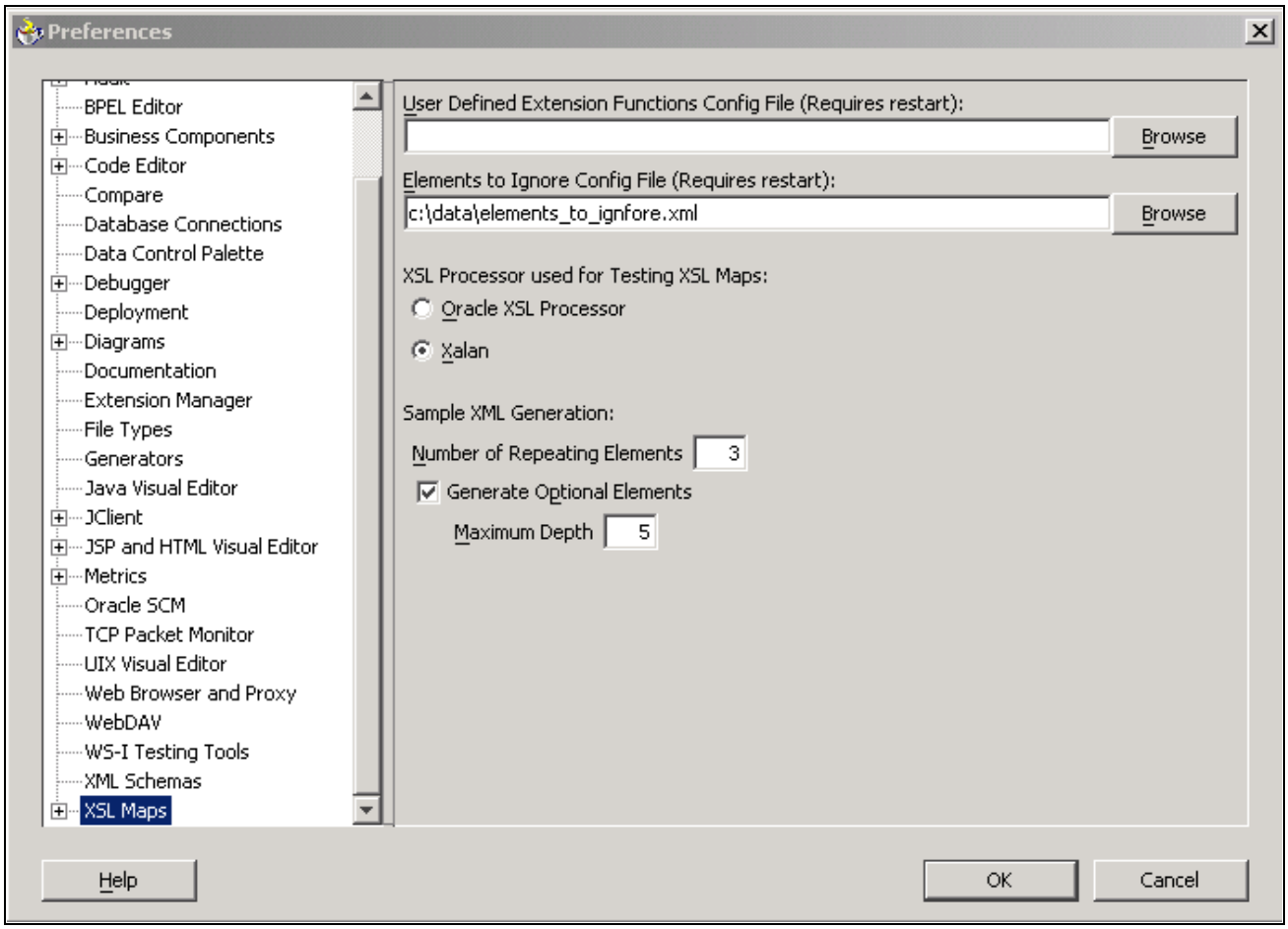
You would create a file using the following format, which tells Oracle XSL Mapper to ignore those elements:

```
<?XML version='1.0' encoding='windows-1252'?>
<elements-to-ignore>
     <element name = "psft_function"/>
     <element name = "psft_workingstorage"/>
</elements-to-ignore>
```

Specifying Elements to Ignore Configuration Files in Oracle XSL Mapper Preferences

Once you create the file of PeopleSoft elements to ignore, enter the file location and name in the mapper Preferences dialog box. The following graphic shows the Preferences dialog box:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Preferences dialog box

To specify an elements to ignore configuration file in Oracle XSL Mapper preferences:

1. In Oracle XSL Mapper, from the Tools menu, select Preferences.

   The Preferences dialog appears.

2. In the list on the left-side of the window, click XSL Maps.

3. In the Elements to Ignore Conf. File (Requires Restart) field, enter or browse to the location of the elements to ignore configuration file that you created.

4. Click the OK button.

You must restart Oracle JDeveloper for the system to recognize the configuration file.

Invoking Transform Programs

Your transform program is invoked by PeopleSoft Integration Broker if you specify its name in the a routing definition for a service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Accessing Transform Message Data

When PeopleSoft Integration Broker invokes a transform program, it inserts the message content into a PeopleCode system variable, *%TransformData*, which remains in scope throughout the program. Each step can access the variable in turn and modify its content, which then becomes available to the next step.

XSLT and PeopleCode steps access %TransformData differently:

- In XSLT, the data is automatically made available to your program. The XSLT program is literally a presentation of the output structure and data, which includes *xsl*tags that reference, process and incorporate the input data into the output structure. There's no need to explicitly refer to %TransformData, which automatically receives the interpreted result of the XSLT.
- In PeopleCode, use the PeopleCode *TransformData* class to access %TransformData. You then access the XML data as a property of the TransformData object called *XmlDoc*, which you assign to an XmlDoc object and process normally. Because the XmlDoc object is a reference to the data portion of %TransformData, your modifications are automatically passed back to the system variable.

Using the TransformData Class

The PeopleCode TransformData class has several properties:

Property	***Description***
XmlDoc	Contains the XML message data. You can assign this to an XmlDoc object and process the data using the XmlDoc class methods and properties. This property is read/write.
Status	Communicates the success or failure of the transform program step to PeopleSoft Integration Broker. Set to *0* for success, the default value. Set to *1* to indicate that the message failed a filtering step. Set to *2* to indicate an error occurred. This property is read/write. See Chapter 18, "Applying Filtering, Transformation and Translation," Filtering Messages, page 442.
SourceNode	The name of the node sending the message. This property is read only.
DestNode	The name of the node receiving the message. This property is read only.
SourceMsgName	The name of the message at the sending node. This property is read only.
DestMsgName	The name of the message at the receiving node. This property is read only.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Property	*Description*
SourceMsgVersion	The name of the message version at the sending node. This property is read only.
DestMsgVersion	The name of the message version at the receiving node. This property is read only.
routingDefnName	Retrieves the routing definition name for the transaction. You can then programmatically retrieve the routing properties specified on the Routings - Routing Properties page.
rejectTransform	Terminates the transaction for asynchronous service operation types. If you set this property for a transform in inbound asynchronous transaction the system will not create a subscription contract. In the Service Operation Monitor Details page for the transaction an informational message will be part of the error message link indicating that the transaction was terminated. If you set this property for an outbound asynchronous transactions the publication contract status in the Service Operation Monitor will be updated to *Done*. The message will not be sent out and an error message will once again be part of the Service Operation Monitor Details page for the transaction indicating the transaction was terminated.

Note. Because transform programs can apply to both request and synchronous response messages, the node *sending* the message could be a synchronous transaction target node that's sending a response back to the synchronous transaction source node, which in this case is the *receiving* node.

Handling Non-XML Data

Because they work only with XML DOM-compliant data, neither XSLT nor PeopleCode transform steps can process non-XML data. The XML DOM provides a way to incorporate such data into an XML structure so your transform programs won't produce errors.

If you're generating a non-XML outbound message in your PeopleSoft application, it's up to you to insert your message content into a special element containing a CDATA tag, as follows:

```
<any_tag psnonxml="yes">
   <![CDATA[your_nonXML_message_content]]>
</any_tag>
```

Note. *Any_tag* can be any tag you want to use.

The following restrictions apply to the content of inbound non-XML messages, such as those in CSV or PDF format, sent by third-party applications:

- Inbound non-XML text messages must be encoded as UTF-8-compliant characters.
- Inbound non-text, or binary, messages must be encoded in base64 format.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Renaming or Deleting Transform Programs

To invoke a transform program, you specify it as part of a routing definition. If you subsequently rename or delete that transform program, it still appears in the routing definition, so service operations using that modifier will fail. To prevent this, do the following:

- If you rename a transform program, make sure you reselect it by its new name in any routing definitions that apply it.
- If you delete a transform program, make sure you select a different program (or no program) in any routing definitions where that apply it.

Filtering Messages

This section provides an overview of message filtering and discusses how to work with a PeopleCode filtering example.

Understanding Message Filtering

You use filtering to suppress an input message based on its content. For example, you can suppress all inbound purchase order messages that specify order quantities less than the minimum number required for a discount.

Place filtering steps early in your Application Engine transform program; each message suppressed by the filter is one less message for subsequent steps to process.

You must use PeopleCode for filtering, so it'll probably be a distinct step. Because you must use the XmlDoc and XmlNode classes in your PeopleCode transform steps, you can analyze messages in any way that those classes support.

Filtering requires the following actions in your PeopleCode program:

1. Retrieve the message content from the %TransformData system variable.
2. Examine your filtering criteria.
3. If the message meets your criteria, do nothing further. It remains intact in the %TransformData system variable for the next transform program step to process.
4. If the message fails to meet your criteria, replace the entire message content with a single node called *Filter*, containing the reason it failed.

   ```
   <?xml version="1.0"?>
   <Filter>reason_for_failure</Filter>
   ```

5. Set the TransformData *Status* property to *1* to indicate failure.

PeopleSoft Integration Broker examines the *Status* property after each step, and terminates the transform program if its value is *1*. You can then view the message in Integration Broker Monitor and see the reason for the failure.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleCode Filtering Example

The following example of filtering presents an input message, the PeopleCode filtering program, and the resulting output message.

Input Message

This is the input to the filtering step. Notice the line item order quantities (shown in emphasis):

```
<?xml version="1.0"?>
<PurchaseOrder>
   <Destination>
      <Address>123 Vine Street</Address>
      <Contact>
         <Name>Joe Smith</Name>
      </Contact>
      <Delivery type="ground">
         <Business>FedEx</Business>
      </Delivery>
   </Destination>
   <Payment>
      <CreditCard cardtype="visa">9999-9999-9999-9999</CreditCard>
   </Payment>
   <LineItems count="2">
      <Li locale="en_us" number="1">
         <Quantity>4</Quantity>
         <ProductName>pencil</ProductName>
         <UOM>box</UOM>
      </Li>
      <Li locale="en_us" number="2">
         <Quantity>10</Quantity>
         <ProductName>paper</ProductName>
         <UOM>large box</UOM>
      </Li>
   </LineItems>
</PurchaseOrder>
```

Note. Although this input message isn't in the PeopleSoft rowset-based message format, it is valid XML.

PeopleCode Filtering Program

This filtering program examines the line item order quantities of the input message and generates the output message that follows. The key statements are in bold:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
/* Get the data from the AE Runtime */
Local TransformData &tempData = %TransformData;
/* Set a temp object to contain the incoming document */
Local XmlDoc &tempDoc = &tempData.XmlDoc;

/* Find the line items quantities contained in the incoming Purchase Order */
Local array of XmlNode &quantities =
 &tempDoc.DocumentElement.FindNodes("LineItems/Li/Quantity");
/* Temp storage of a node */
Local XmlNode &tempNode;

/* Loop through the quantities and make sure they are all above 5 */
For &i = 1 To &quantities.Len
   /* Set the temp node*/
   &tempNode = &quantities [&i];
   /* Make sure the node isn't empty*/
   If ( Not &tempNode.IsNull) Then

      /* Check the value, if not greater than 5 this does not pass filter*/
      If (Value(&tempNode.NodeValue) < 5) Then
         /* Clear out the doc and put in the "Filter" root node */
         If (&tempDoc.ParseXmlString("<?xml version=""1.0""?><Filter/>")) Then
            /* Get the new root node and set the value
            /*  to be the reason for failing filter     */
            &rootNode = &tempDoc.DocumentElement;
            &rootNode.NodeValue = "Line item quantity was found
            that was less than 5!";
            /* Set the status of the transformation to 1 for failed filter*/
            &tempData.Status = 1;
         End-If;
         Break;
      End-If

   End-If
End-For;
```

Output Message

This is the result of applying the PeopleCode filtering program:

```
<?xml version="1.0"?>
<Filter>Line item quantity was found that was less than 5!</Filter>
```

Applying Transformations

This section provides an overview of transformation and discusses using XSLT for transformation.

Understanding Transformation

Use a transformation when one node sends a request or response message with a data structure different from the structure required by the other node. One or both of the participating nodes can be PeopleSoft applications. At either end of the transaction, any of the following structure types may be required:

- The PeopleSoft rowset-based message format.
- An XML DOM-compliant non-rowset-based structure. This is generic XML data.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- A SOAP-compliant XML structure. This is also XML DOM-compliant.
- A non-XML structure. Third-party applications are more likely than PeopleSoft applications to require this type.

Your transformation can be between different structure types or between different structures of the same type.

See Also

Chapter 5, "Understanding Supported Message Structures," page 39

Using XSLT for Transformation

An XSLT transformation simulates the original message structure, then specifies how to treat nodes within that structure. You can:

- Copy the original content of a node without changing anything.
- Define and insert a new version of a node.
- Enter any structure or content directly.
- Eliminate a node by omitting reference to it, or by not inserting anything new in its place.

XSLT Transformation Example

The XSLT wrapper is required.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

The primary node tag matches the original message structure by matching its top level content tag, the message name (QE_SYNC_MSG). Between the message template tags, you can insert any structure or content you want. PeopleSoft Integration Broker replaces each *xsl* tag with the data it references, producing a transformed message as the output of the step.

```
<xsl:template match="QE_SYNC_MSG">
   <QE_SYNC_MSG>
      <xsl:copy-of select="FieldTypes"/>
      <MsgData>
         <Transaction>
            <xsl:apply-templates select="MsgData/Transaction/QE_SALES_ORDER"/>
            <xsl:copy-of select="MsgData/Transaction/PSCAMA"/>
         </Transaction>
      </MsgData>
   </QE_SYNC_MSG>
</xsl:template>
```

The following node example is defined to match a record in the input message by its top level content tag, the record name (QE_SALES_ORDER). This template is applied by the *xsl:apply-templates* tag of the preceding node (shown emphasized).

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Between the record template tags, you can insert any structure or content you want. In this example, *90* is prepended to the QE_ACCT_ID value, and the QE_ACCOUNT_NAME field is renamed to QE_ACCOUNT (shown emphasized). Also, any existing value in the DESCRLONG field is removed, and the remaining fields are passed through with their original values.

```
<xsl:template match="QE_SALES_ORDER">
   <QE_SALES_ORDER><xsl:attribute name="class">
   <xsl:value-of select="@class"/></xsl:attribute>
      <xsl:variable name="temp" select="QE_ACCT_ID"/>
      <QE_ACCT_ID><xsl:value-of select="concat(90,$temp)"/></QE_ACCT_ID>
      <QE_ACCOUNT><xsl:value-of select="QE_ACCOUNT_NAME"/></QE_ACCOUNT>
      <QE_ADDRESS><xsl:value-of select="QE_ADDRESS"/></QE_ADDRESS>
      <QE_PHONE><xsl:value-of select="QE_PHONE"/></QE_PHONE>
      <QE_FROMROWSET/>
      <QE_TOROWSET/>
      <QE_SEND_SOA_BTN/>
      <QE_SEND_SOS_BTN/>
      <DESCRLONG></DESCRLONG>
   </QE_SALES_ORDER>
</xsl:template>
```

Finally, you need the closing wrapper tag:

```
</xsl:stylesheet>
```

Note. You can find more information about XSLT at the World Wide Web Consortium (W3C) web site.

Note. A working transformation example using XSLT is provided in the PeopleTools SDK. The location of the example is <PS_HOME> \sdk\pstransform\samples\TRANSFORMTST.xml.

See Also

http://www.w3.org/Style/XSL/

Applying Message Transformations at the Integration Gateway

Typically, you apply filtering, transformation, and data translation to a message at the node level on the application server by using a transaction modifier to invoke an Application Engine transform program. However, on systems with high transaction volumes, Application Engine transformations can constrict message throughput. To improve performance, you can apply XSLT transformation programs at an integration gateway.

Note. While you may apply transformations at the integration gateway level, PeopleSoft strongly recommends that you apply them at the application server level due to a more robust infrastructure to support them.

See http://www.apache.com.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Applying Message Transformations at the Integration Gateway

Only XSLT transformations can be applied at the gateway. Message filtering, data translation, and PeopleCode transformations must still be applied at the node using an Application Engine transform program, and can be applied in addition to gateway-based transformations.

You can apply XSLT transformations at any gateway that handles the message you want to transform.

When a gateway with transformation enabled processes an IBRequest, it examines the transformation properties in the integrationGateway.properties file to determine if they specify a transformation for the same message with the same source and target nodes as the IBRequest. If these values match, the gateway compiles the specified XSLT transformation program and applies it to the message, then sends the transformed message to the target node.

Note. The IBRequest can specify only a RequestingNode or only a DestinationNode, but it must specify at least one of these values—ig.DefaultServer.LocalNode supplies the other one.

With synchronous transactions, the gateway applies transformations only to the request message, not to the response message. If the original message is compressed and base64 encoded, the gateway decompresses and decodes it before applying the transformation, then compresses and encodes it again before sending.

Note. The integration gateway retains all compiled XSLT transformation programs in a memory cache to improve performance during subsequent transformations. If you edit the code of a transformation program that's been used before, you must purge the compiled programs from the cache so the new version will be recompiled. To do this, click the Refresh button on the gateway definition.

Developing and Implementing Gateway-Based Transformation Programs

Developing and implementing gateway-based transform programs requires the following activities:

1. Determine if the message you want to transform qualifies for gateway-based transformation:

   - The message content must be XML-DOM compliant.
   - The message must not have nonrepudiation activated.

2. Develop the XSLT transformation program.

   See Chapter 18, "Applying Filtering, Transformation and Translation," Applying Transformations, page 444.

   You can develop, test and debug the program within Application Engine, but you must save the program code as an external text file. Place the file in any location that can be accessed from the integration gateway machine, for example:

   ```
   C:\XSLProgs\MyTransform.xsl
   ```

3. Configure the appropriate gateway property settings in the integrationGateway.properties file to enable the transformation.

   See Chapter 18, "Applying Filtering, Transformation and Translation," Setting Integration Gateway Properties for Gateway-Based Transformations, page 448.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Refresh the gateway properties.

Setting Integration Gateway Properties for Gateway-Based Transformations

To apply gateway-based transformations, set the following properties in the integrationGateway.properties file.

For each message you want to transform, you must create a set of property entries using the same number, which associate a given transformation program with that message. However, you can specify the same transformation program for multiple messages.

When entering these settings, each transformation must be numbered for identification, using the convention ig.transform1, ig.transform2, ig.transform3, and so on.

Property	***Description***
ig.isGatewayTransformationEnabled	Specify whether transformation is enabled for this gateway. Valid values are: • *TRUE.* Transformation is enabled. • *FALSE.* Transformation is disabled — the integration gateway will ignore the other transformation properties. This is the default value.
ig.DefaultServer.LocalNode	Enter the name of the node definition that will be used as the source or destination node for a given transformation if either of those values isn't identified; for example you must specify: `ig.DefaultServer.LocalNode=DEF_NODE` All transformations require that you specify both a source node and a destination node. This property applies if either the ig.transformN.SourceNode property or the ig.transformN.DestinationNode property is empty or invalid, or if the IBRequest doesn't specify either RequestingNode or DestinationNode.
ig.transforms	Specify the number of transformations configured in the integrationGateway.properties file; for example: `ig.transforms=7`

ig.transform*N*.XSL	Enter the full path and filename of transformation program *N*.
	Your path specification must use either double back slashes or single forward slashes as separators; for example:
	`ig.transform4.XSL=C:\\XSLProgs\\MyTransform.xsl`
	`ig.transform4.XSL=C:/XSLProgs/MyTransform.xsl`
	`ig.transform4.XSL=/usr/xsls/MyTransform.xsl`

ig.transform*N*.MessageName	Enter the name of the message to be transformed by transformation program N; for example: `ig.transform4.MessageName=MY_MSG_A`

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Property	*Description*
ig.transform*N*.SourceNode	Enter the name of the source node from which the original message is being sent, or enter the value *ANY*; for example: `ig.transform4.SourceNode=NODE_Aig.transform4.SourceNode=ANY` If this value is *ANY*, the value of the ig.DefaultServer.LocalNode property will be used instead.
ig.transform*N*.DestinationNode	Enter the name of the target node to which the transformed message is being sent, or enter the value ANY; for example: `ig.transform4.DestinationNode=NODE_Big.transform4.DestinationNode=ANY` If this value is *ANY*, the value of the ig.DefaultServer.LocalNode property will be used instead.
ig.transform*N*.DestinationMessageName	(Optional.) Enter the name that the target node uses for the transformed version of the message, if it's different from the original message name; for example: `ig.transform4.DestinationMessageName=MY_MSG_B` This enables the gateway to rename the message before sending it, so the target node will recognize and accept it.

Understanding Logged Errors

If an error occurs when you refresh the gateway properties or during a transformation, it's entered in the gateway's error log file.

Integration Gateway Refresh Errors

After you specify integration gateway properties and refresh the gateway, errors can be generated for the following reasons:

- No value is specified for ig.transform*N*.XSL.
- No value is specified for ig.transform*N*.MessageName.
- No value is specified for both ig.DefaultServer.LocalNode and ig.transform*N*.SourceNode.
- No value is specified for both ig.DefaultServer.LocalNode and ig.transform*N*.DestinationNode.
- The gateway is in the process of loading, compiling or caching a transformation program.

Runtime Transformation Errors

Errors are generated for the following reasons when the gateway attempts to apply a transformation:

- Nonrepudiation is enabled for the message.
- The integration gateway is unable to transform the message.
- The integration gateway is unable to decompress and decode the message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- The integration gateway is unable to compress and encode the message.
- The IBRequest does not specify a RequestingNode and no value is specified for ig.DefaultServer.LocalNode.
- The IBRequest does not specify a DestinationNode and no value is specified for ig.DefaultServer.LocalNode.
- The IBRequest specifies neither a RequestingNode nor a DestinationNode.

Performing Data Translation

This section provides an overview of data translation and discusses how to:

- Define codeset groups.
- Define codesets.
- Define codeset values.
- Import and export codesets between databases.
- Delete codesets.
- Use XSLT for data translation.
- Work with an XSLT translation example.
- Work with a PeopleCode translation example.

Understanding Data Translation

Use data translation to modify message content rather than structure, although you can also make local structural changes. It's most appropriate when the sending and receiving systems use different field values, or different combinations of fields and their values, to represent the same information.

Following is a sample scenario:

- Application A transmits customer names in four fields: *Title, First, Middle,* and *Last.*
- Application B uses two fields: *Last* and *First.* It doesn't use a title, but includes the middle name as part of the *First* field.
- Application C uses only one field: *AccountID.*

Clearly the representation used by one application won't be understood by either of the other two. PeopleSoft Integration Broker can apply a transform program to translate each of these representations into a version appropriate to the receiving application.

One Integration Broker node can store in its codeset repository the equivalent fields and values used by another node. When it receives a message from the other node containing a customer name, it can use its codeset repository to translate the information into the form it prefers. It can likewise reverse the process for messages outbound to the other node.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

For a given integration, you can allocate the responsibility for performing data translation in different ways. You can distribute the translation activity among the participating nodes, or you can designate one Integration Broker node to do all the data translation as a hub, whether the messages are inbound, outbound, or being redirected between the other nodes. Using a single node, if possible, can reduce the need for duplicating repository data.

Data Translation Elements

The following elements constitute the codeset repository, managed as PeopleSoft Pure Internet Architecture components:

Codeset group — Maintains a list of the significant data fields and their values that a particular node might send in an initial message. These are name/value pairs a translation program might find (match) and use as the basis for determining what the result message should contain. These name/value pairs are known as *match names* and *match values.*

Each PeopleSoft Integration Broker node that requires data translation must belong to a codeset group.

Codeset — A specific set of match name/match value pairs selected from an existing codeset group. The selected name/value pairs are the basis for possible field value combinations that you want to match in a message, and to which your translation program can respond by modifying the message content. Each codeset typically represents one set of fields that require translation for a given message.

Codeset values — A codeset value is a named value you define, also known as a *return value.* Your translation program can output the return value as a result of matching a specific combination of match values from a codeset.

You associate multiple combinations of codeset values with the combination of an initial codeset group, a codeset from that initial group, and a result codeset group. For each permutation of match values selected from the codeset, you define a different combination of codeset values to apply to your result message.

The other key element of data translation is your translation program, which invokes the codesets and codeset values you've defined.

Data Translation Development Sequence

You must initially define these elements in a particular order:

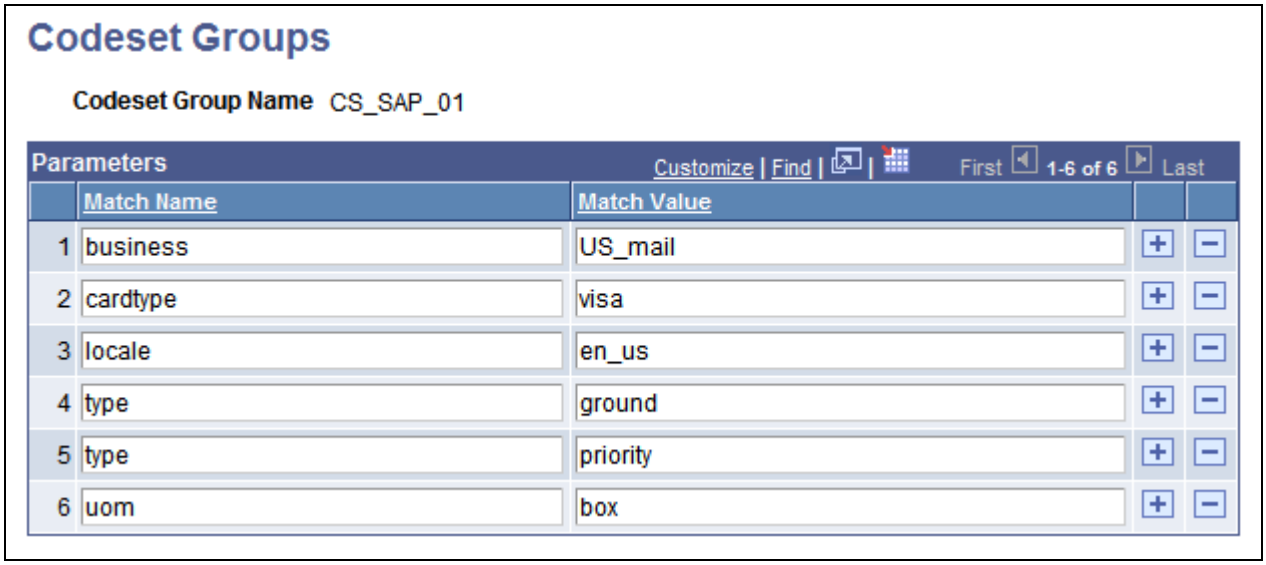
1. Two codeset groups.
2. A codeset based on one of the codeset groups.
3. A set of codeset values.
4. A data translation program, in XSLT or PeopleCode.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

However, it's unlikely that you'll be able to fully define any of these elements without some trial and error. You may find you'll have to modify and extend each element in turn as you develop your data translation program.

Defining Codeset Groups

Use the Codeset Groups page (IB_CODESETGRPVAL) in the Codeset Groups component (IB_CODESETGROUP) to define codeset groups. To access the Codeset Groups page, select PeopleTools, Integration Broker, Integration Setup, Codesets, Codeset Groups.



Codeset Groups page

To define a codeset group:

1. Add a new value, enter a codeset group name, and click Add.

   Enter a name that reflects a common quality of the nodes you plan to assign to this group; for example, the name of the software they all use to manage shipping.

   The Codeset Group page appears.

2. Add a new row.

3. Enter a match name.

   This is the name of a data field that might be part of a message sent by a node belonging to this codeset group. You don't have to create an entry for every field, just the ones that you'll need to translate or use for reference in a translation.

4. Enter a match value.

   This is one of the possible values of the data field represented by the match name.

5. Repeat steps 2 through 4 for each significant name/value pair that you expect to appear in a message.

   This doesn't need to be all possible values of all of the message fields, just the names and values you expect to require translation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

6. Assign one or more nodes to this codeset group.

   Every initial and result node involved in a data translation must belong to a codeset group. You must assign each participating node to an appropriate codeset group by an entry in its node properties.

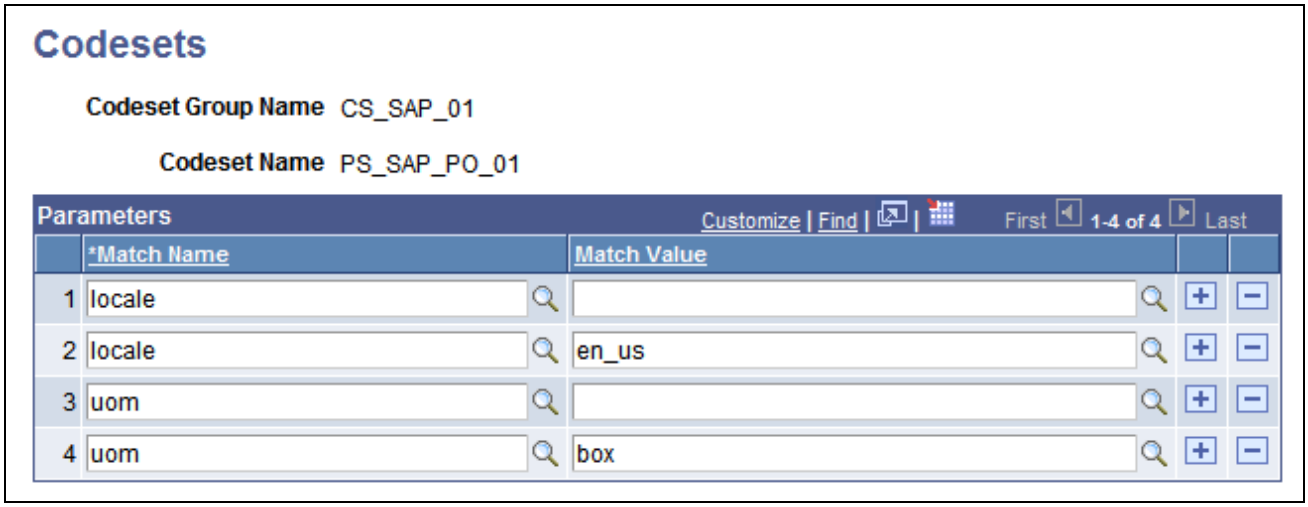
   Note. The assignment for each node is required only in the database of the node performing the data translation. This translating node needn't be either the source or the target. Multiple nodes that represent data the same way may be assigned to the same codeset group.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes," Configuring Nodes

Defining Codesets

Use the Codesets page (IB_CODESET) in the Codesets component. (IB_CODESET). Select PeopleTools, Integration Broker, Integration Setup, Codesets, Codesets to access the Codeset page.



Codesets page

To define a codeset:

1. Add a new value and enter a codeset group name on which to base this codeset.
2. Enter a codeset name and click Add.

   Enter a name that reflects the purpose of this codeset; for example, *SAP_SHIP_METHOD*, to translate the representation of a shipping method in a message. The Codesets page appears.
3. Add a new row.
4. Select a match name from the set defined for the associated codeset group.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Select a match value from the set defined for the selected match name.

   Note. You can leave the value blank. If so, you should do the same for each match name in this codeset, in addition to any other values you select for them. A combination consisting of *all* blank values is treated as a wild card by PeopleSoft Integration Broker, which enables it to respond to unanticipated values specified in your translation program with default behavior that you define.

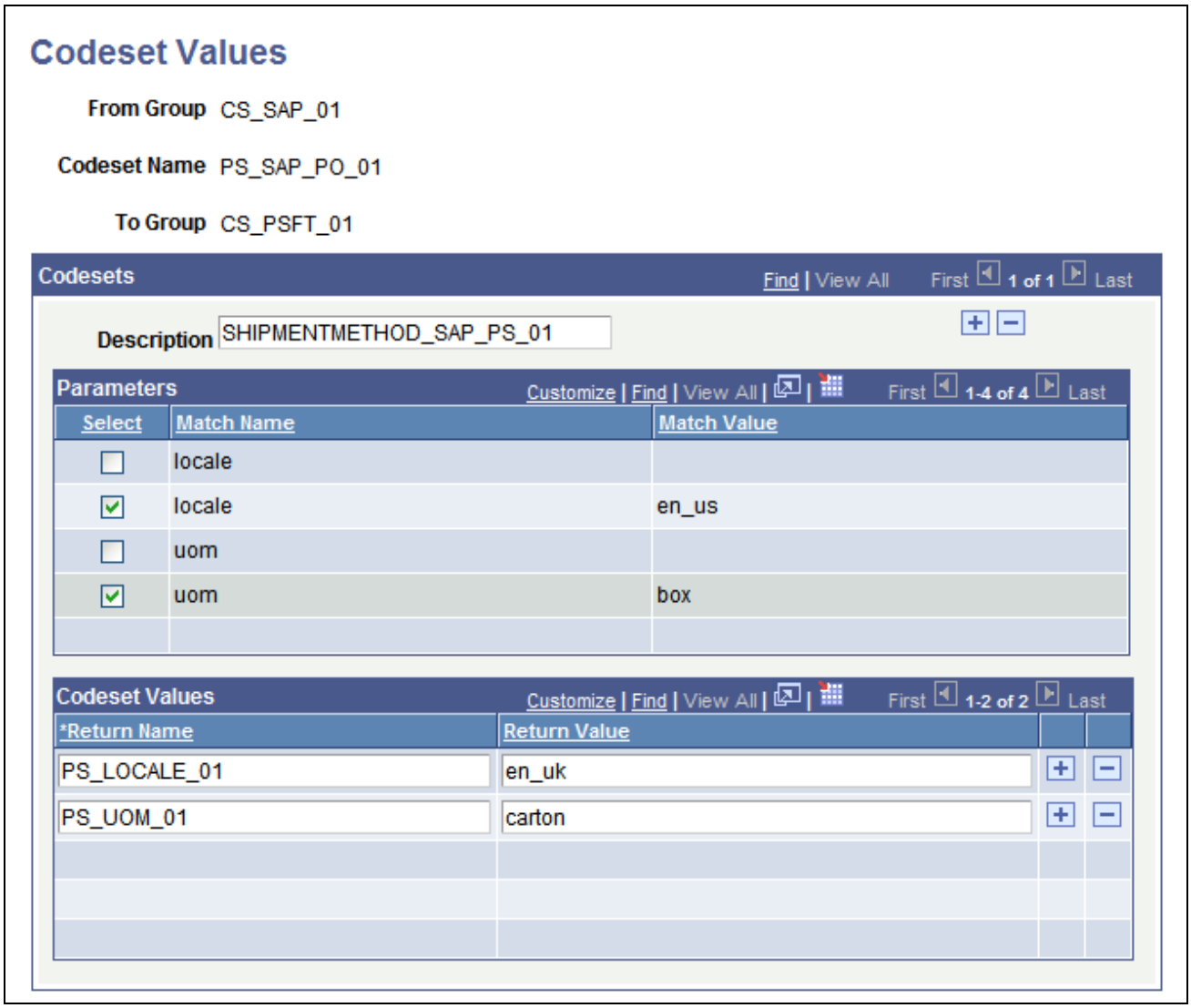
6. Repeat steps 3 through 5 to enter all the name/value pairs that may need to be matched.

   The name/value pairs you select should encompass only the possible value combinations that your translation program needs to match for a single translation. You define a different codeset for each translation based on this codeset group.

Defining Codeset Values

Use the Codeset Values page (IB_CODESETVAL) in the Codeset Values component (IB_CODESETVAL). Select PeopleTools, Integration Broker, Integration Setup, Codesets, Codeset Values to access the Codeset Values page.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Codeset Values page

To define codeset values:

1. Add a new value and select a codeset group name for the *From* group.

   This is the codeset group to which the initial node belongs.

2. Select a codeset name from the codesets based on the group you selected.

   This is the codeset whose match name/match value permutations you wish to match.

3. Select a codeset group name for the *To* group.

   This is the codeset group to which the result node belongs.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Click Add.

   The Codeset Values page appears. The upper grid contains the selected codeset's match name/match value pairs, and the lower grid contains the return values you specify. Each permutation that you define has its own Description field, which can help you distinguish between permutations that may be subtly different from each other.

   Note. To configure an existing codeset values definition, enter its *From* group, codeset name and *To* group on the search page.

   ---

5. Select check boxes to define a permutation of match name/match value pairs.

   For each match name, you can select at most one match value.

   A permutation consisting of all blank values serves as a wild card; it matches any input value combination that isn't matched by any other permutation. However, a permutation with some blank and some non-blank values works differently; it requires the names with blank values to actually match blank field values in the input data.

   Note. You'll generally define only permutations that you expect the input data to contain, but make sure you allow for unforeseen match values by including permutations with all blank values. You can then specify default return values for those permutations. With a large number of match names in the codeset, you can make sure to catch all unforeseen combinations by defining a permutation with all blank match values.

   ---

6. In the Code Set Values grid, enter a return name, and a return value for that name.

   You can use any return name you want, because only your codeset translation program refers to it. Your translation program can use the return value as a field value or as a node name in the output data.

   Important! The set of return names you define *must be identical* for all of the permutations of match name/match value pairs for the current codeset in this definition. Your translation program invokes the codeset and applies the return names from this definition, but it can't anticipate which permutations will be matched, or which actual return values it's applying — just the return names.

   ---

7. (Optional.) In the Code Set Values grid, add a new row and repeat step 6.

   Add as many return name/return value pairs as you need for your output based on the current permutation. If the permutation is matched in the input data, the code set values you define for that permutation become available for you to call and insert in the output data.

8. (Optional.) At the top level of this page, add a new row and repeat steps 5 through 7.

   This inserts a new permutation row, in which you can define a different permutation of match name/match value pairs that you expect for the current codeset. For each permutation, you'll define a separate, independent set of codeset values.

Importing and Exporting Codesets Between Databases

PeopleSoft provides two Data Mover scripts that you can use to import and export codesets between databases:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- CODESET_DELETE_IMPORT.DMS

  Use this script to purge and then import codeset data into a target database.

- CODESET_EXPORT.DMS

  Use this script to export codeset data from a source database to a target database.

Deleting Codesets

Before you delete a codeset, you must delete any codeset values associated with it.

Deleting Codeset Values

To delete codeset values for a codeset:

1. Select PeopleTools, Integration Broker, Integration Setup, Codesets, Codeset Values. The Codeset Values page displays.
2. In the Find an Existing Value tab, in the Codeset Name field, enter the name of the code set you want to delete, or use the Lookup button to locate it.
3. In the Codeset Values section, clear the Select box for each match name corresponding to the code set match name you want to delete, or click the minus (-) button to delete the entire code set scroll area.

   Repeat this process for as many codeset match names that are used.
4. Click the Save button.

Deleting Codesets

To delete a codeset:

1. Select PeopleTools, Integration Broker, Integration Setup, Codesets, Codeset.
2. Select the codeset to delete. The Codeset page displays.
3. Locate the row that contains the codeset you want to delete, and click the minus (-) button on that row.
4. Click the Save button.

Using XSLT for Data Translation

Once you've defined the match name/match value permutations for a given codeset and defined the return values for those permutations, you can write an XSLT translation program that invokes the codeset and applies the return values.

An XSLT translation is based on XSLT transformation structure. However, although you could combine both tasks into a single program, it's better to keep them separate for easier understanding and maintenance.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Psft_function Nodes

To implement data translation capability, PeopleSoft Integration Broker provides a custom XSLT tag called *psft_function.* Each *psft_function* node in your program comprises a single instance of data translation that invokes a particular codeset and applies a specified set of codeset values.

Runtime invocation of codesets in XSLT is a two part process: first the input document is transformed and then all instances of psft_function are resolved in the output document.

Note. You can insert a *psft_function* node anywhere inside the XSLT template containing the fields you want to translate. However, you'll find it easiest to place it at or near the point in the template where the return values will go, to avoid having to specify a complex path to that location.

The *psft_function* tag has the following attributes:

Attribute	*Use*
name	Set to *codeset.*
codesetname	Identifies the codeset whose name/value permutations you want to match in the input data. The routing that invokes this transform program identifies the initial and result nodes involved, and PeopleSoft Integration Broker examines their definitions to determine the *From* group and *To* group. The combination of these two keys and the codeset name identifies the codeset values definition to apply.
source	(Optional.) Overrides the name of the initial node specified by the routing. PeopleSoft Integration Broker uses the specified node's codeset group as the *From* group key, thus invoking a different codeset values definition.
dest	(Optional.) Overrides the name of the result node specified by the routing. PeopleSoft Integration Broker uses the specified node's codeset group as the *To* group key, thus invoking a different codeset values definition.

Note. The *source* and *dest* attributes don't change the initial and result nodes specified in the routing; they just invoke the codeset groups to which those nodes belong.

Following is an example of *psft_function* using all of its attributes:

```
<psft_function name="codeset" codesetname="PS_SAP_PO_01"
 source="SAP_02" dest="PSFT_03">...</psft_function>
```

Parm and Value Nodes

The *psft_function* node can contain two tags, *parm* and *value:*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Use the *parm* tag to specify a match name from the codeset values definition that you specified for this translation. You do this with the tag's only attribute: *name*. Set this to a match name from the codeset values definition.

  The *parm* node should contain a match value, usually specified as an *xsl:value-of* tag that identifies where the value resides in the input data. Use one *parm* node for each distinct match name in the codeset values definition.

- Use the *value* tag to specify a return name from the codeset values definition that you specified for this translation. Also use the *value* tag to identify where to place the return value assigned to that return name for the matched permutation and how to apply that value.

  Use one *value* node for each return name in the codeset values definition that you want in your output.

Value Tag Attributes

The *value* tag has the following attributes:

Attribute	Use
name	Identifies a return name from the codeset values definition you specified for this translation. The return value assigned to this return name can be used as a data value or as a node name in your output depending on the attributes you specify.
select	Identifies an XSLT path (XPATH) to the location where the return value should be applied in the output data.
createIfDNE	(Optional.) Set to *yes* to ensure that the node specified by the *select* attribute is created if it does not exist yet. The return value is inserted as the value of that node.
createNodeFromValue	(Optional.) Set to *yes* to use the return value as the name of a new node, created where the *select* attribute specifies. The *value* tag can contain a valid XSLT value for that node, usually specified as an *xsl:value-of* tag that identifies where the value resides in the input data.

Following is an example of a *value* node:

```
<value name="PS_RET_01" select="." createNodeFromValue="yes"><xsl:value-of
 select="CreditCard"/></value>
```

See Also

Chapter 18, "Applying Filtering, Transformation and Translation," Using XSLT for Transformation, page 445

http://www.w3.org/Style/XSL/

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

XSLT Translation Example

The following example of XSLT data translation presents an example input message, the XSLT translation program, and the resulting output message.

Input Message

This is the input to the XSLT translation:

```
<?xml version="1.0"?>
<PurchaseOrder>
   <Destination>
      <Address>123 Vine Street</Address>
      <Contact>
         <Name>Joe Smith</Name>
      </Contact>
      <Delivery type="ground">
         <Business>FedEx</Business>
      </Delivery>
   </Destination>
   <Payment>
      <CreditCard cardtype="visa">9999-9999-9999-9999</CreditCard>
   </Payment>
   <LineItems count="2">
      <Li locale="en_us" number="1">
         <Quantity>15</Quantity>
         <ProductName>pencil</ProductName>
         <UOM>box</UOM>
      </Li>
      <Li locale="en_us" number="2">
         <Quantity>10</Quantity>
         <ProductName>paper</ProductName>
         <UOM>large box</UOM>
      </Li>
   </LineItems>
</PurchaseOrder>
```

Note. Although this input message isn't in the PeopleSoft rowset-based message format, it is valid XML.

XSLT Data Translation Program

This translation program processes the input message in this example and generates the output message that follows. The statements shown emphasized demonstrate some uses of the *psft_function* node:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="PurchaseOrder">
      <po>
         <xsl:apply-templates/>
      </po>
   </xsl:template>

   <xsl:template match="Destination">
      <dest>
         <address><xsl:value-of select="Address"/></address>
         <name><xsl:value-of select="Contact/Name"/></name>
         <delivery>
            <type>
               <psft_function name="codeset" codesetname="PS_SAP_PO_03"
               dest="PSFT_03">
                  <parm name="type"><xsl:value-of select="Delivery/@type"/>
                  </parm>
                  <value name="PS_RET_01" select="."/>
               </psft_function>
            </type>
            <carrier>
               <psft_function name="codeset" codesetname="PS_SAP_PO_03"
               source="SAP_03">
                  <parm name="Business"><xsl:value-of select="Delivery/
                  Business"/></parm>
                  <value name="PS_RET_01" select="."/>
               </psft_function>
            </carrier>
         </delivery>
      </dest>
   </xsl:template>

   <xsl:template match="Payment">
      <payment>
         <psft_function name="codeset" codesetname="PS_SAP_PO_02">
             <parm name="cardtype"><xsl:value-of select="CreditCard/
             @cardtype"/></parm>
             <value name="PS_RET_01" select="."
             createNodeFromValue="yes"><xsl:value-of select="CreditCard"/>
             </value>
         </psft_function>
      </payment>
   </xsl:template>

   <xsl:template match="Li">
      <li><xsl:attribute name="id"><xsl:value-of select="@number"/></xsl:attribute>
         <name><xsl:value-of select="ProductName"/></name>
         <qty><xsl:value-of select="Quantity"/></qty>
         <uom>
            <psft_function name="codeset" codesetname="PS_SAP_PO_01">
                <parm name="locale"><xsl:value-of select="@locale"/></parm>
                <parm name="uom"><xsl:value-of select="UOM"/></parm>
                <value name="PS_RET_01" select="."/>
                <value name="PS_RET_02" select="../type" createIfDNE="yes"/>
            </psft_function>
         </uom>
      </li>
   </xsl:template>
</xsl:stylesheet>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Output Message

This is the result of applying the XSLT translation:

```
<po>
   <li id="">
      <name>pencil</name>
      <qty>15</qty>
      <uom>Carton</uom>
      <type>Bic</type>
   </li>
   <li id="">
      <name>paper</name>
      <qty>10</qty>
      <uom>Box</uom>
      <type>Bic</type>
   </li>
   <dest>
      <address>123 Vine Street</address>
      <name>Joe Smith</name>
      <delivery>
         <type>Ground</type>
         <carrier>Federal Express</carrier>
      </delivery>
   </dest>
   <payment>
      <VISA>4024-9920-9892-8982</VISA>
   </payment>
</po>
```

PeopleCode Translation Example

Although XSLT is the recommended language for using the codeset repository to translate message data, you can use PeopleCode for this purpose as well. Because XSLT works only with XML DOM-compliant message data, you must use PeopleCode if the message you're translating contains non-XML data, including formats like comma separated values (CSV).

Once you've defined the match name/match value permutations for a codeset with respect to a given target codeset group and defined the return values for those permutations, you can write a PeopleCode translation program that invokes that codeset and applies the return values.

FindCodeSetValues Built-in Function

To implement data translation capability, Integration Broker provides a PeopleCode built-in function called *FindCodeSetValues*, which takes four parameters and returns a two dimensional array.

The following example of PeopleCode data translation presents an example input message, the PeopleCode translation program, and the resulting output message.

Input Message

This is the input to the PeopleCode translation:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<Header>
  <LANGUAGE_CODE>en_us</LANGUAGE_CODE>
  <STATUS_CODE>0</STATUS_CODE>
</Header>
```

PeopleCode Data Translation Program

This translation program processes the input message in this example, and generates the output message that follows. The statement shown emphasized demonstrates the use of the FindCodeSetValues function:

```
/* Get the data from the AE Runtime */
Local TransformData &incomingData = %TransformData;

/* Set a temp object to contain the incoming document */
Local XmlDoc &tempDoc = &incomingData.XmlDoc;

/* Find the Language and status codes value*/
Local string &langCode = &tempDoc.DocumentElement.FindNode("LANGUAGE_CODE").
   Node Value;
Local string &statusCode = &tempDoc.DocumentElement.FindNode("STATUS_CODE").
   Node Value;

/* Create an array to hold the name value pairs */
Local array of array of string &inNameValuePairsAry;

/* Load the array with some values */
&inNameValuePairsAry = CreateArray(CreateArray("LANG", &langCode),
 CreateArray("STATUS", &statusCode));

/* Find the codeset values */
&outAry = FindCodeSetValues("STATUS_CHANGE", &inNameValuePairsAry,
 &incomingData.SourceNode, &incomingData.DestNode);

/* Create the new output doc */
If &tempDoc.ParseXmlString("<?xml version=""1.0""?><NewHeader/>") Then

   /* Make sure something was returned */
   If &outAry.Len > 0 Then

      /* Create the new Status Code Node */
      Local XmlNode &statusNode = &tempDoc.DocumentElement.AddElement("STATUS");

      /* Since this is a 2D array, get the Return Value*/
      &statusNode.NodeValue = &outAry [1][2];
   End-If;
End-If;
```

Output Message

This is the result of applying the PeopleCode translation:

```
<?xml version="1.0"?>
<NewHeader>
   <STATUS>Open</STATUS>
</NewHeader>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleCode Language Reference, "PeopleCode Built-in Functions," FindCodeSetValues

Rejecting Transformation Programs

Situations may arise when you may want to terminate a transaction. For example, you may not want a transaction published to a specific node.

Using the %TransformData.rejectTransform property and the %IB_Transform_Rejected built-in function, you can terminate asynchronous transactions based on content data.

You set this property in a PeopleSoft Application Engine transform program as follows:

```
%TransformData.rejectTransform = %IB_Transform_Rejected;
```

If you set the %TransformData.rejectTransform property within a transform of an inbound asynchronous transaction, the system will not create a subscription contract and the data is not sent. If you view such a transaction in the Service Operations Monitor, the Operation Instances page displays a status of *Done*. If you open the Asynchronous Details page in the monitor, an Error link displays. If you click the link an informational message appears that indicates that the transaction was terminated.

If you set this property for an outbound asynchronous transaction, the Publication Contracts page in the Service Operations Monitor will show a contract status of *Done* for the transaction. However, the system does not send the message and, as with the inbound scenario, the Asynchronous Details page for the transaction will display an Error link. If you click the link an informational message appears that indicates that the transaction was terminated.

See Also

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Monitoring Asynchronous Service Operations," Asynchronous Service Operation Statuses

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Monitoring Asynchronous Service Operations," Monitoring Asynchronous Service Operation Instances

PeopleTools 8.52: Integration Broker Service Operations Monitor, "Viewing Asynchronous Service Operation Details"

Terminating Transformation Programs

If you need to terminate a transform program for reasons that aren't considered error conditions by PeopleSoft Integration Broker, you can use a PeopleCode step to force the transform program to terminate and generate a readable error message as well.

To generate an error:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Replace the entire message content with a single node called *Error*, containing the reason for the error.

   ```
   <?xml version="1.0"?>
   <Error>reason_for_error</Error>
   ```

2. Set the TransformData *Status* property to *2* to indicate error status.

PeopleSoft Integration Broker examines the *Status* property after each step and terminates the transform program if its value is *2*. You can then view the message in Service Operations Monitor and see the reason for the error.

Note. If an XSLT or PeopleCode step fails for reasons that you haven't taken into account, Integration Broker automatically sets the *Status* property to *2* and aborts the transform program, but you can't provide your own error message.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 19

Managing Error Handling, Logging, Tracing, and Debugging

This chapter discusses how to:

- Manage integration gateway message and error logging.
- Manage application server logging and tracing.
- Debug integrations.

Understanding Error Handling, Logging, Tracing and Debugging

This chapter provides an overview of error handling, logging, tracing and debugging.

Error handling, logging, tracing, and debugging with PeopleSoft Integration Broker can occur on an integration gateway, application server, or application engine, depending on the type and location of processing.

Understanding Integration Gateway Error Handling

Error handling is an integration gateway service that assists connectors to manage errors that occur during processing. Errors on the integration gateway are handled by target connectors and listening connectors.

This section discusses:

- Target connector error handling.
- Listening connector error handling.
- Integration gateway exception types.

Target Connector Error Handling

The Target Connector Interface (TCI) specifies the methods that target connectors must implement for the integration gateway to manage them. These methods include a set of standard exceptions that target connectors generate when they experience errors during processing.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Listening connectors or the gateway manager catch these exceptions and provide an appropriate implementation for each. When the source of the message is an integration engine, the gateway manager catches the exceptions. Otherwise, listening connectors are responsible for handling exceptions that are generated during processing.

Listening Connector Error Handling

Unlike target connectors, listening connectors are not managed by the gateway manager and, therefore, do not adhere to any interface. However, a listening connector must invoke the gateway manager to pass a message from the integration gateway to the integration engine.

The gateway manager has predefined exceptions.

In general, exceptions are thrown in a target connector and caught by a listening connector. As a result, a listening connector must catch these exceptions and handle them as appropriate. Typically, the listening connector generates an error message and sends it back to the requester.

Integration Gateway Exception Types

This section discusses integration gateway exception types.

Standard Exceptions

The following standard error and exception types are handled by the integration gateway, target connectors, and listening connectors:

Exception Type	*Description*
DuplicateMessageException	A target connector attempted to process a message that has already been processed. This is usually discovered based on an error that is attained from the external system that is being contacted. Of the connectors that are delivered with the PeopleSoft software, only the PeopleSoft 8.1 target connector (PSFT81TARGET) can generate this exception. Target connectors are not required to generate this exception.
ExternalApplicationException	The message reached its intended destination but could not be processed. Determining that the destination could not process a message requires significant knowledge of the destination system, which a target connector might not have. Whenever possible, a target connector should attempt to determine this situation; otherwise this task must be decentralized and handled outside of the integration gateway. For example, the HTTP target connector (HTTPTARGET) generates this exception when the external system returns an HTTP system code of *500*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Exception Type	*Description*
ExternalSystemContactException	The target connector cannot establish a connection with the intended destination. This is one of the most common exceptions. When this exception is thrown during an asynchronous transaction, PeopleSoft Integration Broker tries to resend the message until successful.
GeneralFrameworkException	A general error occurred.
InvalidMessageException	A connector or the gateway manager determined that the message cannot be processed because of missing or erroneous information in a request or response.
MessageMarshallingException	A gateway service's attempt to get information from an IBRequest or IBResponse failed. This can occur when the gateway services attempt to access a content section of a document by using an out-of-range index from one of the following methods: • GetContentSectionAt(index) • GetContentSectionInfoAt(index) • RemoveContentSectionAt(index) If you try to access IBRequest or IBResponse with an out-of-range index by using any of these methods, this exception is thrown automatically and processing is interrupted.
MessageUnmarshallingException	A gateway service's attempt to build an IBRequest or IBResponse failed. Failure can occur when: • Instantiating an IBRequest or IBResponse from a Multipurpose Internet Mail Extensions (MIME) format where the message that was sent does not comply with the PeopleSoft MIME format. • Instantiating an IBRequest by using the PS_XML format and passing an invalid PS_XML message. This is typically from the HTTP listening connector. • Setting invalid values to methods, such as setTransactionID or setMessageType. These failures cause the integration gateway to generate this exception automatically and processing is interrupted.

Java Exceptions

Target connectors and listening connectors can handle miscellaneous Java exceptions, such as NullPointerException and ArrayOutOfBoundsException.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Managing Integration Gateway Message and Error Logging

This section provides an overview of message and error logging and discusses how to:

- Set up message and error logging.
- View non-English characters in integration gateway log files.
- Manage message logging.
- Manage error logging.

Understanding Message and Error Logging

Error and message logging is a gateway service that you use to monitor messages that flow through the integration gateway.

Logging takes place within both target and listening connectors. Connectors can log all message requests and responses. As a result, you can use logging to:

- Track message flow.
- Troubleshoot processing errors.

Setting Up Message and Error Logging

By default, an integration gateway logs all errors and warnings, as well as information of important, standard, and low importance.

Set up message and error logging by using the integrationGateway.properties file. Use the Logging Setting section to view or change default settings, such as the level of gateway logging, where the system writes log files, the maximum size of the log file, and the number of file backups or archives to keep.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Setting Logging Properties

Viewing Non-English Characters in Integration Gateway Log Files

To view non-English characters in integration gateway log files, enable UTF-8 encoding in your web browser. For example, if you are using Microsoft Internet Explorer, you can enable UTF-8 encoding by selecting View, Encoding, Unicode (UTF-8). If you are using Mozilla FireFox, you can enable UTF-8 encoding by selecting View, Character Encoding, Unicode (UTF-8).

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Managing Message Logging

Message logging records the following information for messages that pass through the integration gateway:

- Time and date.
- Message description.
- Content of the passed message object.
- Message level.

The default location of the integration gateway message log is <PIA_HOME>\webserv\<DOMAIN>\applications\peoplesoft\PSIGW.war\msgLog.html.

Change the location of the log in the integrationGateway.properties file.

Message Logging in Target Connectors

Message logging in a target connector occurs:

- Before delivering the request to the external system.

  The connector logs the request in the format in which the external system delivered it. For example, an HTTP target connector logs the exact HTTP output stream request. The PeopleSoft target connector logs the MIME request to be sent to the integration engine.

- After it receives a response from the external system.

  The connector logs the response in the format in which it receives it. For example, an HTTP target connector logs the exact HTTP input stream response. The PeopleSoft target connector logs the MIME response that it received from the integration engine.

Message Logging in Listening Connectors

Message logging in a listening connector occurs:

- At the point where the request enters the system.

  The connector logs the request in the format in which the sending system delivers it. For example, the HTTP listening connector logs the exact HTTP input stream request. The PeopleSoft listening connector logs the MIME request that it received from the integration engine.

- Following the delivery of a response to the requestor system.

  The connector logs the response in the format in which it was delivered. For example, the HTTP listening connector logs the exact HTTP output stream response. The PeopleSoft listening connector logs the MIME response that it sent back to the integration engine.

Message Logging Methods and Parameters

Invoke the logMessage method for integration gateway message logging:

```
logMessage(String Description, Object message, int MessageLevel)
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Use the following parameters:

Parameter	Description
Description	Specify a description as a string.
Object	Specify the message object. Typically this object is an IBRequest or IBResponse. If another object is passed, the toString method is invoked for the object, and the result is logged.
MessageLevel	Set the relative importance of the information that you are logging. The ig.log.level property setting in the integrationGateway.properties file determines the log level that is currently in effect. If the MessageLevel value that is passed here is less than or equal to the ig.log.level property setting, the message is written to the log file. Values are: • *3*: Important information. • *4*: Standard information. • *5*: Low-importance information. The ig.messageLog.filename property in the integrationGateway.properties file determines the log file location.

Managing Error Logging

Error logging captures processing errors that occur in the integration gateway. When an error occurs, the following information is logged:

- Error level.
- Description.
- Message catalog entry information.
- Stack trace identifying the problem.
- IBRequest and IBResponse (if available).

The default location of the integration gateway error log is <PIA_HOME>\webserv\<DOMAIN>\applications\peoplesoft\PSIGW.war\errorLog.html

Change the location of the log in the integrationGateway.properties file.

Error Logging Methods and Parameters

Invoke the logError method for integration gateway error logging:

```
logError (String Description, IBRequest, IBResponse, int ErrorLevel, Throwable)
```

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Use the following parameters:

Parameter	*Description*
Description	Specify a description as a string.
IBRequest	Specify the IBRequest for this transaction, if available. If not available, pass Null.
IBResponse	Specify the IBResponse for this transaction, if available. If not available, pass Null.
ErrorLevel	Specify whether the log is written to permanent storage. This determines the severity of the error. The ig.log.level property in the integrationGateway.properties file determines the log level that is currently in effect. If the ErrorLevel value that is passed here is less than or equal to the ig.log.level property setting, the error is written to the log file. Values are: • *100*: Language exception. • *1*: Standard gateway exception. • *2*: Warning. The ig.errorLog.filename property in the integrationGateway.properties file determines the log file location.
Throwable	Specify the Java exception or error that is associated with the error. This is used to log the stack trace that is associated with the error.

The gateway manager and delivered listening connectors feature built-in error logging that invokes the logError method. The delivered target connectors do not feature built-in error logging, and instead generate errors to the gateway manager or listening connectors, where they are handled or logged.

Managing Application Server Logging and Tracing

Use the PeopleSoft Application Server Administration menu to:

- View application server and Oracle Tuxedo log files.

  See *PeopleTools 8.52: System and Server Administration*, "Using PSADMIN Menus," Editing Configuration and Log Files.

- Trace Structured Query Language (SQL) and PeopleCode on your domains.

  See *PeopleTools 8.52: System and Server Administration*, "Setting Application Server Domain Parameters," Trace Options.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Set the level of network tracing (log fence).

  See *PeopleTools 8.52: System and Server Administration*, "Setting Application Server Domain Parameters," Domain Settings.

- View the certificate authentication logs, including information about mismatched distinguished names and certificates that are not in the database.

  This information is contained in the APPSRV.LOG file.

You can also use the tracing functionality in PeopleSoft Application Engine, which enables you to monitor the performance of transforms in your implementation of PeopleSoft Integration Broker.

See Also

Chapter 18, "Applying Filtering, Transformation and Translation," page 415

PeopleTools 8.52: Application Engine, "Tracing Application Engine Programs"

Debugging Integrations

This section discusses how to:

- Debug handler PeopleCode.
- Handle common issues.

Debugging Handler PeopleCode

Use the Handler Tester utility to debug service operation handler PeopleCode.

The Handler Tester utility enables you to use the PeopleSoft Pure Internet Architecture to test any of the following handler types:

- OnSend.
- OnRequest
- OnRouteReceive
- OnRouteSend.
- OnAckReceive
- OnNotify.

You can test handlers without setting up a routing definition, without having pub/sub booted on your application server, and without impacting other developer activity on the system.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: Integration Broker Testing Utilities and Tools, "Using the Handler Tester Utility"

Handling Common Issues

Use this table to handle common issues in PeopleSoft Integration Broker:

Area or Suspected Issue	*Debugging Suggestion*
Application server exceptions.	Check the application server log: <PS_CFG_HOME>\appserv\<Domain>\LOGS\ appsrv.log
Message handlers are not running.	Check the application server domain status or queue status in the PeopleSoft Application Server Administration menu (PSAdmin). Select Domain Status, Server Status or Domain Status, Queue Status.
Integration gateway.	Check the integrationGateway.properties file and verify the property settings. The default file location is <PIA_HOME>\webserv\<DOMAIN>\applications\peoplesoft\PSIGW.war.war\WEB-INF\integrationGateway.properties.
Integration gateway.	Check the integration gateway message log. The default file location is <PIA_HOME>\webserv\<DOMAIN>\applications\peoplesoft\PSIGW.war.war\msgLog.html.
Queues are paused.	Check the Service Operations Monitor. Select PeopleTools, Integration Broker, Service Operations Monitor, Administration, Queue Status.
A node is paused.	Check the Service Operations Monitor. Select PeopleTools, Integration Broker, Service Operations Monitor, Administration, Node Status.
Incorrect gateway uniform resource locator (URL).	Check the Gateways component to verify that the integration gateway URL is correct. Select PeopleTools, Integration Broker, Configuration, Gateways.
Node inactive.	Check the node definition. Select PeopleTools, Integration Broker, Integration Setup, Nodes.
Subscription PeopleCode is missing or incorrect.	Check the Service Operations Monitor. Select PeopleTools, Integration Broker, Monitoring, Asynchronous Services, Subscription Contracts..

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Area or Suspected Issue	*Debugging Suggestion*
A service operation is inactive.	Check the service operation definition in the PeopleSoft Pure Internet Architecture. Select PeopleTools, Integration Broker, Integration Setup, Service Operations.
There are transform problems.	• Check the Application Engine object in PeopleSoft Application Designer. • For before and after images, check the Service Operations Monitor. For asynchronous service operations, , select PeopleTools, Integration Broker, Service Operations Monitor, Monitoring, Asynchronous Details. Click the View XML link for the publication contract or subscription contract. For synchronous service operations, select PeopleTools, Integration Broker, Service Operations Monitor, Monitoring, Synchronous Details. Use the Log Type drop-down list box to select Request Transformed or Response Transformed, and then click View XML. • Verify that the TraceAE flag in the following directory equals 8192: <PS_CFG_HOME>\appserv\<Domain>\psappsrv.cfg Setting the TraceAE flag in the psappsrv.cfg file instructs the application server to generate a transformation trace log with the .aet extension, written to the following directory: <PS_CFG_HOME>\appserv\<Domain>\LOGS\ <operID>_<machine name>.AET The log file contains: • The original XML structure as it entered the transformation engine. • The output of the XML as it passed through each step of the transform program.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Area or Suspected Issue	*Debugging Suggestion*
Integration Broker security	Set the application server logging level to 4 or greater to capture information related to the following situations: • No routing at source node. • No routing at target node. • User invoking a service operation has different permissions than specified on the service operation on the source node. • No node password found on source node. • No node password found on target node. • Mismatched node password on source and target nodes. • No permissions set for the service operation on the source node.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 20

Providing Services

This chapter discusses how to:

- Provide services.
- Access generated WSDL and WADL documents.
- Delete WSDL and WADL documents.

Understanding Providing Services

This chapter describes how to provide services using the Provide Web Service wizard, access generated WSDL and WADL documents, and delete WSDL and WADL document.

The chapter also describes WSDL and WADL documents that the system generates.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Understanding the Provide Web Service Wizard

PeopleSoft Integration Broker features a Provide Web Service wizard that steps you through the task of providing web services.

The Provide Web Service component (IB_WSDL_EXPORT) features a wizard you can use to provide web services. You can publish WSDL and WADL documents to the WSDL repository in the PeopleSoft system or publish WSDL documents to external UDDI repositories.

After you generate a WSDL or WADL document, the Provide Web Service wizard displays a URL for each document you generated. This enables you to access the WSDL document or WADL document content using the URL. In addition, PeopleSoft Integration Broker provides a WSIL URL which lists the provided services and corresponding WSDL URLs.

You can use the Provide Web Service wizard to select one or more services for which to generate WSDL documents or WADL documents. The system generates a separate document for each service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Providing WSDL Documents

This section provides an overview of provided WSDL documents provided by the Provide Web Service wizard.

WSDL Features

Provided WSDL documents include support WS-interoperability standards for WSDL and include:

- WS-Addressing header elements for asynchronous request/response operation types.
- WS-Security elements UsernameToken and SAMLToken.

In addition, provided WSDL document feature PartnerLinkType elements which are used when consumed by a BPEL application.

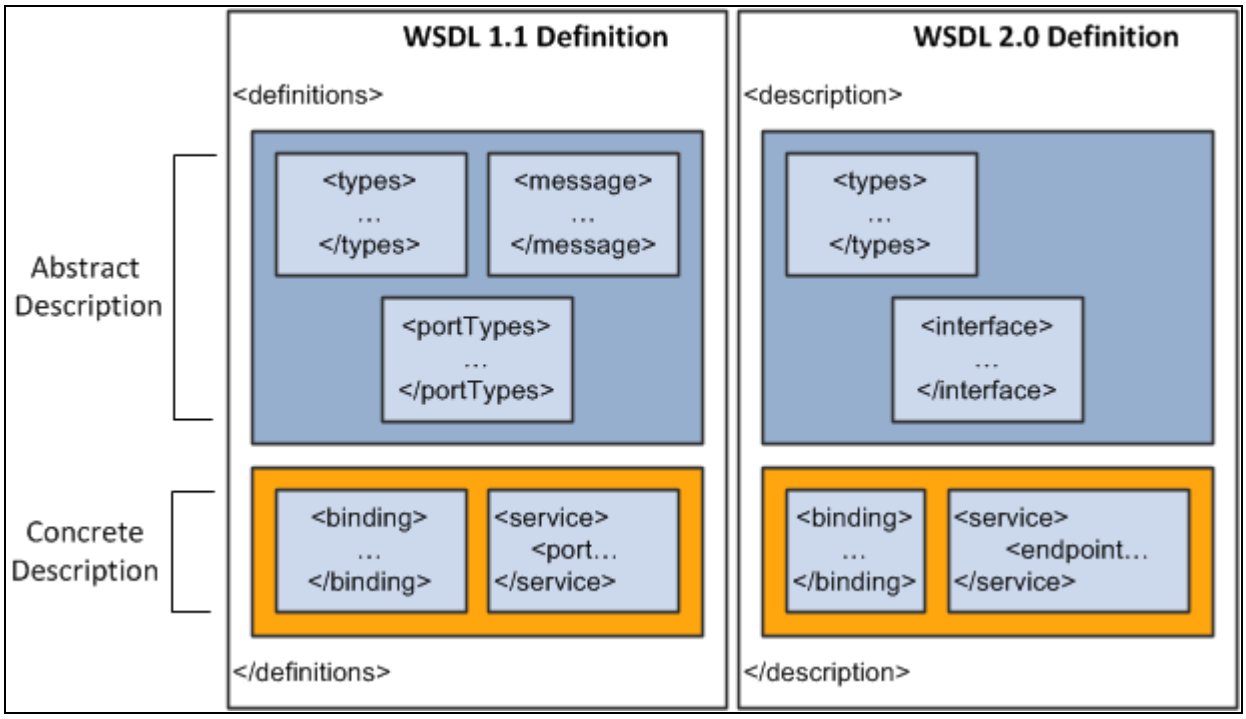
WSDL Document Specifications

Integration Broker can provide WSDL documents in WSDL 1.1 or WSDL 2.0 format. By default, the Provide Web Service wizard used to provide WSDL documents generates WSDL in 1.1 format and features and option to generate WSDL documents in WSDL 2.0 format.

See Chapter 20, "Providing Services," Step 1: Select Services to Provide, page 503.

The following graphic illustrates the differences between WSDL 1.1 and WSDL 2.0 documents at the abstract and concrete layers:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Abstract and concrete layers of WSDL 1.1 and WSDL 2.0 documents

Additional information about the elements in WSDL 1.1 documents and WSDL 2.0 documents is provided elsewhere in this chapter.

See Chapter 20, "Providing Services," Provided WSDL Documents, page 483.

Supported Operation Types for WSDL Documents

The Provide Web Service wizard can create WSDL documents for service operations having the following operation types:

- Synchronous.
- Asynchronous one-way.
- Asynchronous request/response.

Requirements for Nonrowset-Based Message Schemas

This section discusses requirements and considerations for creating nonrowset-based message schemas for service operations in order to generate WSDL documents using the Provide Web Service wizard.

Note. The PeopleSoft system automatically generates message schemas for rowset-based messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Target Namespace

Nonrowset-based message schemas must contain a target namespace. If no target namespace exists in the schema an error occurs when the system generates the WSDL document.

You may define multiple schema imports to the same target namespace, but different schema locations must be defined.

Multiple Root Element and Complex Type Tags

If the PeopleSoft system finds multiple root <element> or <complexType> tags in nonrowset-based message schemas, only the first one is referenced in the WSDL document or container message schema.

In addition, the WSDL would allow schema imports to the same target namespace but different schema locations and use <xsd:include> when the schema Namespace is same as the WSDL namespace.

See Also

Chapter 20, "Providing Services," Prerequisites for Providing Services, page 500

Locations for Publishing WSDL Documents

Using the Provide Web Service wizard, you can publish WSDL documents to the follow locations:

- PeopleSoft WSDL repository. (Application database.)

  The PeopleSoft WSDL repository is the default publishing location. All generated WSDL documents are published to the PeopleSoft WSDL repository. You may publish WSDL documents to a UDDI repository in addition to the WSDL repository.

- Universal Description, Discovery, and Integration (UDDI) repositories.

  Services published to UDDI repositories are available to other PeopleSoft and external systems. If another PeopleSoft system wants to invoke an exported service from UDDI, it can consume the WSDL document from the UDDI reference into its system to create a service and routing definition.

UDDI Repositories and Endpoints

When publishing a WSDL document to a UDDI repository, the PeopleSoft system publishes the current endpoint value defined in the Target Location field in the Service Configuration page.

The endpoint value in the actual WSDL document is a dynamic one, since you can change the target location value, for example, when you move from development to production.

If you change the target location you should change the endpoints of previously published WSDL documents, either manually in the UDDI registry or by republishing the WSDL documents to your UDDI repository using the changed endpoint.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

WSDL URL Formats

After you a publish a WSDL document using the Provide Web Service wizard, the system displays a WSDL URL. The URL provided is the path to the WSDL document location in the WSDL repository in the PeopleSoft Pure Internet Architecture. The URL is used by external systems that will be invoking a PeopleSoft service.

The default URL format is path style. The following example shows a WSDL URL in path format:

```
http://localhost/PSIGW.war/PeopleSoftServiceListeningConnector/PT_WORKLIST.1.wsdl
```

The path style URL is generated by appending the WSDL document name to the target location value specified in the Service Configuration page.

PeopleSoft Integration Broker also supports a query parameter format. The following example shows a WSDL URL in query parameter format:

```
http://PeopleSoftServiceListeningConnector?Operation=GetWSDL&wsdl=PT_WORKLIST.1
```

The query parameter style URL is generated by passing either the WSDL document name or service name.version or service alias.version as a query parameter.

PeopleSoft still supports the query parameter format, however path format is preferred. Note that if using query parameter format, manual intervention may be required if the schema target location is changed.

Provided WSDL Documents

Every WSDL document you generate using PeopleSoft Integration Broker is divided into sections. This section describes the WSDL document sections and provides an example of the WSDL template that the PeopleSoft system uses to generate WSDL, as well as example WSDL documents for each of the supported operation types.

Sections of Provided WSDL Documents

WSDL documents that you provide using PeopleSoft Integration Broker contain the following sections:

Section	***WDSL 1.1 Description***	***WSDL 2.0 Description***
<definitions>	Specifies the namespaces for the WSDL document, W3C XML Schema and SOAP. A unique namespace will be captured from the Service definition, which will be used to define the WSDL namespaces. The format of this namespace is as follows: http://xmlns.oracle.com/Enterprise/<App Name>/<Service Name>. When a service is defined within an application database, the namespace field is defaulted to the service namespace defined on the Service Configuration page.	NA. The <description> element serves the same purpose in WSDL 2.0 documents. See <description>.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Section	*WDSL 1.1 Description*	*WSDL 2.0 Description*
<description>	NA. The <definitions> element serves the same purpose in WSDL 1.1 documents. See <definitions>.	Specifies the namespaces for the WSDL document, W3C XML Schema and SOAP. A unique namespace will be captured from the Service definition, which will be used to define the WSDL namespaces. The format of this namespace is as follows: http://xmlns.oracle.com/Enterprise/<App Name>/<Service Name>. When a service is defined within an application database, the namespace field is defaulted to the service namespace defined on the Service Configuration page.
<partnerLinktype>	A partnerLinkType defines the role of services and the port Type.	No change from WSDL 1.1. See WSDL 1.1 description.
<types>	Captures the simple and complex types required by the schema of the request and response message definitions of the service operations. For services with component interface handlers, some of the system methods, such as Create and Get, will require complex message types resembling the structure of the component interface buffer.	No change from WSDL 1.1. See WSDL 1.1 description.
<message>	Defines the abstract messages required for the selected operations. The data types for these are obtained from the Types section of the WSDL document.	There are no <message> elements in WSDL2.0. Instead, the operations within the interface refer to the schema elements directly.
<portType>	Features a named set of abstract operations and the abstract messages involved. This section includes all operations of the service selected for export.	NA. See <interface>
<interface>	NA See <portType>	WSDL 2.0 features an <interface> element that replaces the <portType> element in WSDL 1.1. Unlike the <portType> element in WSDL 1.1, the operations within the <interface> element do not point to messages, but instead point to the schema elements contained within the <types> element.
<binding>	Specifies the network protocol and data format of messages used for a specific port type. For providing web services PeopleSoft utilizes SOAP packaging and HTTP transport protocols. The data format of messages is the Document style format.	No change from WSDL 1.1. See WSDL 1.1 description.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Section	WDSL 1.1 Description	WSDL 2.0 Description
<operation>	This is an abstract definition of a service operation, which specifies request/response/fault messages.	No change from WSDL 1.1. See WSDL 1.1 description.
<service>	A service groups together endpoints that implement a common interface.	No change from WSDL 1.1. See WSDL 1.1 description.
<port>	In WSDL 1.1 the <port> element contains the definition of the address to which the client should request the service. The <port> element also has reference to binding used by the service	NA. See <endpoint>.
<endpoint>	NA. See <port>.	In WSDL 2.0 the <port> element is replaced by the <endpoint> element. The location attribute of endpoint contains the definition of the address to which the client should request the service. The <endpoint> element also has reference to binding used by the service

Note. In WSDL documents generated by the PeopleSoft system, WS-Security policies are assigned to the bind operation section.

Example 1: WSDL Template

The following example is the WSDL document template that the PeopleSoft system uses when it generates WSDL documents. The elements in bold are WSDL document sections discussed in the previous section:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
definitions name="DefinitionsName"
  targetNamespace="NamespaceURI"
  xmlns:prefix="NamespaceURI"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <plnk:partnerLinkType name="PartnerLinkTypeName">
    <!-- Provider Role -->
       <plnk:role name="ProviderRoleName">
       <plnk:portType name="ProviderPortTypeReference">
        </plnk:role>
  <!-- In case of Async Request/Response this role is also required -->
  <plnk:role name="RequestorRoleName">
    <plnk:portType name="CallbackPortTypeReference">
     </plnk:role>
    </plnk:partnerLinkType>
<types>
    <!-- One or more schemas -->
  </types>
  <message name="MessageName">
    <part name="PartName" type="TypeNameReference"/>
  </message>
  <portType name="PortName">
    <operation name="OperationName">
      <input message="MessageNameReference"/>
      <output message="MessageNameReference"/>
      <fault message="MessageNameReference"/>
    </operation>
  </portType>
  <binding name="BindingName" type="PortNameReference">
    <soap:binding style="rpc|document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="OperationName">
      <soap:operation soapAction="ActionValue"/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="TargetNamespace"
          use="encoded"/>
     </input>
     <output>
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="TargetNamespace"
        use="encoded"/>
     </output>
    </operation>
  </binding>
  <service name="ServiceName">
    <port name="PortName" binding="BindingNameReference">
      <soap:address location="URL"/>
   </port>
  </service>
</definitions>
```

Example 2: Synchronous WSDL Example

The following example shows a synchronous WSDL document provided by the PeopleSoft system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0" ?>
<wsdl:definitions name="PT_WORKLIST.1"
  targetNamespace="http://xml.namespace.oracle.com/services"
  xmlns:GetWorklistEntryStatusRequest.v1="http://xmlns.oracle.com/
   Enterprise/Tools/schemas/PT_WL_GET_INSTANCE_REQ_CONT.v1"
  xmlns:GetWorklistEntryStatusResponse.v1="http://xmlns.oracle.com/
   Enterprise/Tools/schemas/PT_WL_GET_INSTANCE_RESP_CONT.v1"
  xmlns:OperationFault.V1="http://xmlns.oracle.com/schemas/Fault"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http:
   //xml.namespace.oracle.com/services"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
  <wsp:UsagePolicy wsdl:Required="true" />
  <plnk:partnerLinkType name="PT_WORKLIST_PartnerLinkType">
    <plnk:role name="PT_WORKLIST_Provider">
      <plnk:portType name="tns:PT_WORKLIST_PortType" />
    </plnk:role>
  </plnk:partnerLinkType>
  <wsdl:types>
    <xsd:schema elementFormDefault="qualified"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/
       schemas/PT_WL_GET_INSTANCE_REQ_CONT.v1" schemaLocation="Get
        WorklistEntryStatusRequest.v1.xsd" />
      <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/
        schemas/PT_WL_GET_INSTANCE_RESP_CONT.v1" schemaLocation="
         GetWorklistEntryStatusResponse.v1.xsd" />
      <xsd:import namespace="http://xmlns.oracle.com/schemas/Fault"
       schemaLocation="OperationFault.V1.xsd" />
     </xsd:schema>
    </wsdl:types>
  <wsdl:message name="GetWorklistEntryStatusRequest.v1">
    <wsdl:part element="GetWorklistEntryStatusRequest.v1:GetWorklist
     EntryStatusRequest" name="parameter" />
  </wsdl:message>
  <wsdl:message name="GetWorklistEntryStatusResponse.v1">
    <wsdl:part element="GetWorklistEntryStatusResponse.v1:GetWorklist
     EntryStatusResponse" name="parameter" />
  </wsdl:message>
  <wsdl:message name="OperationFault.V1">
    <wsdl:part element="OperationFault.V1:OperationFault" name="parameter" />
  </wsdl:message>
  <wsdl:portType name="PT_WORKLIST_PortType">
  <wsdl:operation name="GetWorklistEntryStatus">
    <wsdl:documentation>Get worklist keys and status</wsdl:documentation>
    <wsdl:input message="tns:GetWorklistEntryStatusRequest.v1" name=
     "GetWorklistEntryStatusRequest.v1" />
    <wsdl:output message="tns:GetWorklistEntryStatusResponse.v1" name=
     "GetWorklistEntryStatusResponse.v1" />
    <wsdl:fault message="tns:OperationFault.V1" name="OperationFault.V1" />
   </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="PT_WORKLIST_Binding" type="tns:PT_WORKLIST_PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/
      soap/http" />
    <wsdl:operation name="GetWorklistEntryStatus">
      <soap:operation soapAction="GetWorklistEntryStatus.V1" style=
       "document" />
      <wsp:Policy wsu:Id="UsernameTokenSecurityPolicyPasswordOptional"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wss-wssecurity-utility-1.0.xsd">
        <wsp:ExactlyOne>
          <wsp:All>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
            <wsse:SecurityToken wsp:Usage="wsp:Required" xmlns:wsse="
             http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
             wssecurity-secext-1.0.xsd">
             <wsse:TokenType>wsse:UserNameToken</wsse:TokenType>
             <Claims>
               <SubjectName MatchType="wsse:Exact" />
               <UsePassword wsp:Usage="wsp:Optional" />
             </Claims>
            </wsse:SecurityToken>
          </wsp:All>
        </wsp:ExactlyOne>
      </wsp:Policy>
    <wsdl:input name="GetWorklistEntryStatusRequest.v1">
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       use="literal" />
    </wsdl:input>
    <wsdl:output name="GetWorklistEntryStatusResponse.v1">
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       use="literal" />
    </wsdl:output>
    <wsdl:fault name="OperationFault.V1">
      <soap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/" name="OperationFault.V1" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="WorklistServices">
  <wsdl:documentation>Peopletools Worklist</wsdl:documentation>
  <wsdl:port binding="tns:PT_WORKLIST_Binding" name="PT_WORKLIST_Port">
    <soap:address location="http://sbandyop-pc/PSIGW.war/PeopleSoftService
     ListeningConnector" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Example 3: Asynchronous Request/Response WSDL Document

The following example shows an asynchronous request/response WSDL document provided by the PeopleSoft system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<wsdl:definitions name="PT_WORKLIST.1"
targetNamespace="http://xml.namespace.oracle.com/services"
xmlns:CreateWorklistEntryRequest.v1="http://xmlns.oracle.com/Enterprise/
  Tools/schemas/PT_WL_CREATE_REQUEST_CONT.v1"
xmlns:CreateWorklistEntryResponse.v1="http://xmlns.oracle.com/Enterprise/
  Tools/schemas/PT_WL_CREATE_RESPONSE_CONT.v1"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://xml.
  namespace.oracle.com/services" xmlns:wsa="http://schemas.xmlsoap.org/ws/
  2003/03/addressing"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
  <wsp:UsagePolicy wsdl:Required="true"/>
  <plnk:partnerLinkType name="PT_WORKLIST_PartnerLinkType">
    <plnk:role name="PT_WORKLIST_Provider">
      <plnk:portType name="tns:PT_WORKLIST_PortType"/>
    </plnk:role>
    <plnk:role name="PT_WORKLIST_Requester">
      <plnk:portType name="tns:PT_WORKLIST_CallbackPortType"/>
    </plnk:role>
  </plnk:partnerLinkType>
  <wsdl:types>
    <xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/
      2001/XMLSchema">
      <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/
        schemas/PT_WL_CREATE_REQUEST_CONT.v1" schemaLocation="CreateWorklist
        EntryRequest.v1.xsd"/>
      <xsd:import namespace="http://xmlns.oracle.com/Enterprise/Tools/
        schemas/PT_WL_CREATE_RESPONSE_CONT.v1" schemaLocation="CreateWorklist
         EntryResponse.v1.xsd"/>
      <xsd:import namespace="http://schemas.xmlsoap.org/ws/2003/03/
        addressing" schemaLocation="http://schemas.xmlsoap.org/ws/2003/
        03/addressing/"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="CreateWorklistEntryRequest.v1">
    <wsdl:documentation>Create worklist item Request</wsdl:documentation>
    <wsdl:part element="CreateWorklistEntryRequest.v1:CreateWorklist
      EntryRequest" name="parameter"/>
  </wsdl:message>
  <wsdl:message name="CreateWorklistEntryResponse.v1">
    <wsdl:part element="CreateWorklistEntryResponse.v1:CreateWorklist
      EntryResponse" name="parameter"/>
  </wsdl:message>
  <wsdl:message name="InitiateHeader">
    <wsdl:documentation>SOAP Header message for correlating Asynchronous
       callback</wsdl:documentation>
    <wsdl:part element="wsa:MessageID" name="MessageID"/>
    <wsdl:part element="wsa:ReplyTo" name="ReplyTo"/>
  </wsdl:message>
  <wsdl:message name="CallbackHeader">
    <wsdl:documentation>SOAP Header message for callback Asynchronous
       operation
    </wsdl:documentation>
    <wsdl:part element="wsa:RelatesTo" name="RelatesTo"/>
  </wsdl:message>
  <wsdl:portType name="PT_WORKLIST_PortType">
    <wsdl:operation name="CreateWorklistEntry">
      <wsdl:documentation>Create worklist Entry. This is the Request Operation
        in a Asynchronous Request/Response pair. Callback Operation :
          CreateWorklistEntry_CALLBACK</wsdl:documentation>
      <wsdl:input message="tns:CreateWorklistEntryRequest.v1" name="
         CreateWorklistEntryRequest.v1"/>
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="PT_WORKLIST_CallbackPortType">
  <wsdl:operation name="CreateWorklistEntry_CALLBACK">
    <wsdl:documentation>Create worklist Entry - Callback. This is the
       Callback Operation in a Asynchronous Request/Response pair.
    </wsdl:documentation>
    <wsdl:input message="tns:CreateWorklistEntryResponse.v1" name=
      "CreateWorklistEntryResponse.v1"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="PT_WORKLIST_Binding" type="tns:PT_WORKLIST_PortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.
    org/soap/http"/>
  <wsdl:operation name="CreateWorklistEntry">
    <soap:operation soapAction="CreateWorklistEntry.V1" style="document"/>
    <wsp:Policy wsu:Id="UsernameTokenSecurityPolicyPasswordOptional"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
       wssecurity-utility-1.0.xsd">
      <wsp:ExactlyOne>
        <wsp:All>
          <wsse:SecurityToken wsp:Usage="wsp:Required" xmlns:wsse=
            "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
              wssecurity-secext-1.0.xsd">
            <wsse:TokenType>wsse:UserNameToken</wsse:TokenType>
            <Claims>
              <SubjectName MatchType="wsse:Exact"/>
              <UsePassword wsp:Usage="wsp:Optional"/>
            </Claims>
          </wsse:SecurityToken>
        </wsp:All>
      </wsp:ExactlyOne>
    </wsp:Policy>
    <wsdl:input name="CreateWorklistEntryRequest.v1">
      <soap:header encodingStyle="" message="tns:InitiateHeader" part=
        "MessageID" use="literal" wsdl:required="false"/>
      <soap:header encodingStyle="" message="tns:InitiateHeader" part=
        "ReplyTo" use="literal" wsdl:required="false"/>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        use="literal"/>
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="PT_WORKLIST_CallbackBinding" type="tns:
   PT_WORKLIST_CallbackPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.
    org/soap/http"/>
  <wsdl:operation name="CreateWorklistEntry_CALLBACK">
    <soap:operation soapAction="CreateWorklistEntry_CALLBACK.V1" style=
      "document"/>
    <wsp:Policy wsu:Id="UsernameTokenSecurityPolicyPasswordOptional"
       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
         wssecurity-utility-1.0.xsd">
      <wsp:ExactlyOne>
        <wsp:All>
          <wsse:SecurityToken wsp:Usage="wsp:Required" xmlns:wsse=
             "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                wssecurity-secext-1.0.xsd">
            <wsse:TokenType>wsse:UserNameToken</wsse:TokenType>
            <Claims>
              <SubjectName MatchType="wsse:Exact"/>
              <UsePassword wsp:Usage="wsp:Optional"/>
            </Claims>
          </wsse:SecurityToken>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
            </wsp:All>
          </wsp:ExactlyOne>
        </wsp:Policy>
        <wsdl:input name="CreateWorklistEntryResponse.v1">
          <soap:header encodingStyle="" message="tns:CallbackHeader"
             part="RelatesTo" use="literal" wsdl:required="true"/>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             use="literal"/>
        </wsdl:input>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="WorklistServices">
      <wsdl:documentation>Peopletools Worklist</wsdl:documentation>
      <wsdl:port binding="tns:PT_WORKLIST_Binding" name="PT_WORKLIST_Port">
        <soap:address location="http://ORACLE_ENDPOINT"/>
      </wsdl:port>
    </wsdl:service>
    <wsdl:service name="WorklistServices_Callback">
      <wsdl:documentation>Peopletools Worklist - Callback</wsdl:documentation>
      <wsdl:port binding="tns:PT_WORKLIST_CallbackBinding" name=
         "PT_WORKLIST_Callback_Port">
        <soap:address location="http://Client.EndPoint.Set.By.Caller"/>
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Example 4: Asynchronous One-Way WSDL Document

The following example shows an asynchronous one-way WSDL document provided by the PeopleSoft system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0" ?>
<wsdl:definitions name="QEPC_FLO_MSG.1" targetNamespace="http://xmlns.
oracle.com/Enterprise/Tools/services/QEPC_FLO_MSG.1"
   xmlns QEPC_FLO_MSG.VERSION_1="http://xmlns.oracle.com/Enterprise/
     Tools/schemas/QEPC_FLO_MSG.VERSION_1" xmlns:plnk="http://schemas.
   xmlsoap.org/ws/2003/05/partner-link/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http:
     //xmlns.oracle.com/Enterprise/Tools/services/QEPC_FLO_MSG.1"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
   <wsp:UsagePolicy wsdl:Required="true" />
   <plnk:partnerLinkType name="QEPC_FLO_MSG_PartnerLinkType">
     <plnk:role name="QEPC_FLO_MSG_Provider">
       <plnk:portType name="tns:QEPC_FLO_MSG_PortType" />
     </plnk:role>
   </plnk:partnerLinkType>
   <wsdl:types>
     <xsd:schema elementFormDefault="qualified"
       targetNamespace="http://xmlns.oracle.com/Enterprise/Tools/schemas/
         QEPC_FLO_MSG.VERSION_1"
       xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
         QEPC_FLO_MSG.VERSION_1"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <xsd:element name="QEPC_FLO_MSG" type="xsd:string" />
     </xsd:schema>
   </wsdl:types>
   <wsdl:message name="QEPC_FLO_MSG.VERSION_1">
     <wsdl:part element="QEPC_FLO_MSG.VERSION_1:QEPC_FLO_MSG" name=
       "parameter" />
   </wsdl:message>
   <wsdl:portType name="QEPC_FLO_MSG_PortType">
   <wsdl:operation name="QEPC_FLO_MSG">
     <wsdl:documentation>Test</wsdl:documentation>
     <wsdl:input message="tns:QEPC_FLO_MSG.VERSION_1" name="QEPC_FLO_
      MSG.VERSION_1" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="QEPC_FLO_MSG_Binding" type="tns:QEPC_FLO_MSG_
    PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.
      org/soap/http" />
    <wsdl:operation name="QEPC_FLO_MSG">
      <soap:operation soapAction="QEPC_FLO_MSG.v1" style="document" />
        <wsp:Policy wsu:Id="UsernameTokenSecurityPolicyPasswordOptional"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
          wss-wssecurity-utility-1.0.xsd">
            <wsp:ExactlyOne>
              <wsp:All>
                <wsse:SecurityToken wsp:Usage="wsp:Required" xmlns:wsse="
                 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                 wssecurity-secext-1.0.xsd">
                   <wsse:TokenType>wsse:UserNameToken</wsse:TokenType>
                   <Claims>
                     <SubjectName MatchType="wsse:Exact" />
                     <UsePassword wsp:Usage="wsp:Optional" />
                   </Claims>
                   </wsse:SecurityToken>
              </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>
        <wsdl:input name="QEPC_FLO_MSG.VERSION_1">
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
           encoding/" use="literal" />
        </wsdl:input>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="QEPC_FLO_MSG">
      <wsdl:documentation>File Utilities Test</wsdl:documentation>
      <wsdl:port binding="tns:QEPC_FLO_MSG_Binding" name="QEPC_FLO_MSG_Port">
        <soap:address location="http://sbandyop-pc/PSIGW.war/PeopleSoft
         ServiceListeningConnector" />
      </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

PartnerLinkType Support

A service may play a single or dual role in a partnership with a business process.

In a one-way partnership the service may play a single role of provider, whereas in a two-way partnership the service may play the roles of a provider as well as a requester (for callbacks). This type of partnership is termed by Business Process Execution Language (BPEL) as a PartnerLinkType.

A service may participate in different types of partnerships with a process or another service. In each of these partnerships, the service may play a single or dual role.

PartnerLinkType Structure

To ease the task of process developers consuming PeopleSoft services, a basic PartnerLinkType structure is provided in the PeopleSoft-provided WSDL. Process developers may or may not choose to use this PartnerLinkType structure.

The following table describes details of the PartnerLinkType structure for each service operation type:

Operation Type	*PartnerLinkType Description*
Synchronous	The PartnerLinkType has a single Provider role.
Asynchronous one-way	The PartnerLinkType has a single Provider role.
Asynchronous Request/Response	The PartnerLinkType has two roles for the Provider portType and the Requester Callback portType.

The following sections feature examples of the PartnerLinkType structures the PeopleSoft system generates for each service operation type.

Example 1: Synchronous PartnerLinkType Structure

The following example shows the PartnerLinkType structure that the PeopleSoft system generates for an inbound synchronous service operation:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<portType name="HelloWorldSync">
        <operation name="process">
            <input message="client:HelloWorldSyncRequestMessage"/>
            <output message="client:HelloWorldSyncResponseMessage"/>
        </operation>
    </portType>

    <plnk:partnerLinkType name="HelloWorldSyncPLType">
        <plnk:role name="HelloWorldSyncProvider">
              <plnk:portType name="wsdl_target:HelloWorldSync"/>
        </plnk:role>
      </plnk:partnerLinkType>
```

Example 2: Asynchronous One-Way PartnerLinkType Structure

The following example shows the PartnerLinkType structure that the PeopleSoft system generates for an inbound asynchronous one-way service operation.

```
<portType name="UpdateOrderAsync">
        <operation name="UpdateOrder">
            <input message="client:OrderRequestMessage"/>
        </operation>
    </portType>

    <plnk:partnerLinkType name="UpdateOrderAsyncPLType">
        <plnk:role name="UpdateOrderAsyncProvider">
              <plnk:portType name="wsdl_target:UpdateOrderAsync"/>
        </plnk:role>
      </plnk:partnerLinkType>
```

Example 3: Asynchronous Request/Response PartnerLinkType Structure

The following example shows the PartnerLinkType structure that the PeopleSoft system generates for an inbound asynchronous request/response service operation:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<!--
  PortType definition
  -->

    <!-- portType implemented by the QuoteConsumer PeopleSoft service -->
    <portType name="QuoteConsumer">
        <operation name="GetQuote">
            <input message="tns:QuoteConsumerRequestMessage"/>
        </operation>
    </portType>

    <!-- portType implemented by the requester of QuoteConsumer PeopleSoft service
         for asynchronous callback purposes
         -->
    <portType name="QuoteConsumerCallback">
        <operation name="GetQuoteCallback">
            <input message="tns:QuoteConsumerResultMessage"/>
        </operation>
    </portType> <!--
  PartnerLinkType definition
  -->

    <!-- the QuoteConsumer partnerLinkType binds the service and
         requestor portType into an asynchronous conversation.
         -->
    <plnk:partnerLinkType name="QuoteConsumerPLType">
        <plnk:role name="QuoteConsumerProvider">
              <plnk:portType name="wsdl_target:QuoteConsumer"/>
        </plnk:role>
        <plnk:role name="QuoteConsumerRequester">
              <plnk:portType name="wsdl_target:QuoteConsumerCallback"/>
        </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Understanding WADL Documents

This section discusses WADL documents.

Supported Operation Types for WADL Documents

PeopleSoft Integration Broker provides WADL documents for REST services only. REST services are synchronous service operations.

Locations for Publishing WADL Documents

Using the Provide Web Service wizard, you can publish WADL documents to the PeopleSoft WSDL repository in the PeopleTools database.

You can access a WADL document in the WSDL Repository using the Services component or by using the URL generated by the Provide Web Service Wizard when you provide a REST service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 20, "Providing Services," Accessing Generated WSDL Documents and WADL Documents, page 510

WADL URL Format

After you a publish a WADL document using the Provide Web Service wizard, the system displays a WADL URL. The URL provided is the path to the WADL document in the WSDL repository in the PeopleSoft Pure Internet Architecture. The URL is used by external systems that will be invoking a PeopleSoft service.

The WADL URL format is path style. The following example shows a WADL URL in path format where the service provided is *QE_WEATHERSTATION*:

```
http://<machine>:<port>/PSIGW/RESTListeningConnector/QE_WEATHERSTATION.1.wadl
```

Provided WADL Documents

Every WADL document you generate using PeopleSoft Integration Broker is divided into sections. This section describes the WADL document sections and provides an example of a WADL document generated by the system.

The following example shows a WADL document provided by the PeopleSoft system:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<application xmlns="http://xmlns.oracle.com/Enterprise/Tools/services"
xmlns:QE_FLIGHTPLAN_SYNC.VERSION_1="http://xmlns.oracle.com/Enterprise/
Tools/schemas/QE_FLIGHTPLAN_SYNC.VERSION_1" xmlns:WEATHERDATA.v1="http:
//xmlns.oracle.com/Enterprise/Tools/schemas/WEATHERDATA.v1" xmlns:
WEATHER_FAULT.v1="http://xmlns.oracle.com/Enterprise/Tools/schemas/
Weather.Weather_Fault.v1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <grammars>
    <include href="WEATHERDATA.v1.xsd"/>
    <include href="WEATHER_FAULT.v1.xsd"/>
    <include href="QE_FLIGHTPLAN_SYNC.VERSION_1.xsd"/>
  </grammars>
  <resources base=" http://kcollins-t61/PSIGW/RESTListeningConnector/
   QE_LOCAL">
    <resource uri="WeatherStation.v1">
      <method href="#GETWeatherStation.v1"/>
      <method href="#HEADWeatherStation.v1"/>
      <method href="#POSTWeatherStation.v1"/>
      <method href="#PUTWeatherStation.v1"/>
      <method href="#DELETEWeatherStation.v1"/>
    </resource>
  </resources>
  <method id="DELETEWeatherStation.v1" name="DELETE">
    <request>
      <resource path="weather/{state}/{city}?forecast={day}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
        <param name="day" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}/{city}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}">
        <param name="state" required="true" style="template" type="xsd:string"/>
      </resource>
    </request>
    <response Status="200">
      <representation element="WEATHERDATA.v1:WeatherInfo" id="WEATHERDATA.v1.xsd"
       mediaType="text/xml"/>
    </response>
  </method>
  <method id="GETWeatherStation.v1" name="GET">
    <request>
      <resource path="weather/{state}/{city}?forecast={day}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
        <param name="day" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}/{city}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}">
        <param name="state" required="true" style="template" type="xsd:string"/>
      </resource>
      <param name="Accept-Charset" style="header" type="xsd:string">
        <doc xml:lang="ENG">Character sets that are acceptable: utf-8</doc>
      </param>
      <param name="Accept-Language" style="header" type="xsd:string">
        <doc xml:lang="ENG">Acceptable languages for response en-US</doc>
      </param>
      <param name="If-Modified-Since" style="header" type="xsd:string">
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
          <doc xml:lang="ENG">Allows a 304 Not Modified to be returned if content is
           unchanged</doc>
        </param>
        <param name="Authorization" required="true" style="header">
          <doc xml:lang="en">Web Service call must be autheticated using Basic HTTP
          auth.</doc>
        </param>
      </request>
      <response Status="200">
        <representation element="WEATHERDATA.v1:WeatherInfo" id="WEATHERDATA.v1.xsd"
         mediaType="text/html"/>
        <param name="Allow" style="header" type="xsd:string">
          <doc xml:lang="ENG">Valid actions for a specified resource.</doc>
        </param>
        <param name="Content-Language" style="header" type="xsd:string">
          <doc xml:lang="ENG">The language the content is in</doc>
        </param>
        <param name="Date" style="header" type="xsd:string">
          <doc xml:lang="ENG">The date and time that the message was sent</doc>
        </param>
        <param name="Expires" style="header" type="xsd:string">
          <doc xml:lang="ENG">Gives the date/time after which the response is⇒
 considered
           stale</doc>
        </param>
        <param name="Host" style="header" type="xsd:string">
          <doc xml:lang="ENG">Transaction Server</doc>
        </param>
        <param name="Warning" style="header" type="xsd:string">
          <doc xml:lang="ENG">A general warning about possible problems with the⇒
 entity
           body.</doc>
        </param>
        <fault Status="400" element="Weather_Fault" id="WEATHER_FAULT.v1" mediaType=⇒
"text/xml"/>
      </response>
    </method>
    <method id="HEADWeatherStation.v1" name="HEAD">
      <request>
        <resource path="weather/{state}/{city}?forecast={day}">
          <param name="state" required="true" style="template" type="xsd:string"/>
          <param name="city" required="true" style="template" type="xsd:string"/>
          <param name="day" required="true" style="template" type="xsd:string"/>
        </resource>
        <resource path="weather/{state}/{city}">
          <param name="state" required="true" style="template" type="xsd:string"/>
          <param name="city" required="true" style="template" type="xsd:string"/>
        </resource>
        <resource path="weather/{state}">
          <param name="state" required="true" style="template" type="xsd:string"/>
        </resource>
        <param name="Accept-Charset" style="header" type="xsd:string">
          <doc xml:lang="ENG">Character sets that are acceptable: utf-8</doc>
        </param>
        <param name="Accept-Language" style="header" type="xsd:string">
          <doc xml:lang="ENG">Acceptable languages for response en-US</doc>
        </param>
        <param name="If-Modified-Since" style="header" type="xsd:string">
          <doc xml:lang="ENG">Allows a 304 Not Modified to be returned if
          content is unchanged</doc>
        </param>
      </request>
      <response Status="0">
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      <param name="Allow" style="header" type="xsd:string">
        <doc xml:lang="ENG">Valid actions for a specified resource.</doc>
      </param>
      <param name="Content-Language" style="header" type="xsd:string">
        <doc xml:lang="ENG">The language the content is in</doc>
      </param>
      <param name="Date" style="header" type="xsd:string">
        <doc xml:lang="ENG">The date and time that the message was sent</doc>
      </param>
      <param name="Expires" style="header" type="xsd:string">
        <doc xml:lang="ENG">Gives the date/time after which the response is
        considered stale</doc>
      </param>
      <param name="Host" style="header" type="xsd:string">
        <doc xml:lang="ENG">Transaction Server</doc>
      </param>
      <param name="Warning" style="header" type="xsd:string">
        <doc xml:lang="ENG">A general warning about possible problems with
         the entity body.</doc>
      </param>
    </response>
  </method>
  <method id="POSTWeatherStation.v1" name="POST">
    <request>
      <resource path="weather/{state}/{city}?forecast={day}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
        <param name="day" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}/{city}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}">
        <param name="state" required="true" style="template" type="xsd:string"/>
      </resource>
      <representation element="QE_FLIGHTPLAN_SYNC.VERSION_1:QE_FLIGHTPLAN_SYNC"
      id="QE_FLIGHTPLAN_SYNC.VERSION_1.xsd" mediaType="text/xml"/>
    </request>
    <response Status="201">
      <representation element="WEATHERDATA.v1:WeatherInfo" id="WEATHERDATA.v1.xsd"
       mediaType="text/xml"/>
      <fault Status="400" element="Weather_Fault" id="WEATHER_FAULT.v1" mediaType="
       text/xml"/>
    </response>
  </method>
  <method id="PUTWeatherStation.v1" name="PUT">
    <request>
      <resource path="weather/{state}/{city}?forecast={day}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
        <param name="day" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}/{city}">
        <param name="state" required="true" style="template" type="xsd:string"/>
        <param name="city" required="true" style="template" type="xsd:string"/>
      </resource>
      <resource path="weather/{state}">
        <param name="state" required="true" style="template" type="xsd:string"/>
      </resource>
      <representation element="QE_FLIGHTPLAN_SYNC.VERSION_1:QE_FLIGHTPLAN_SYNC" id=
      "QE_FLIGHTPLAN_SYNC.VERSION_1.xsd" mediaType="text/xml"/>
    </request>
    <response Status="200">
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      <representation element="WEATHERDATA.v1:WeatherInfo" id="WEATHERDATA.v1.xsd"
       mediaType="text/xml"/>
    </response>
  </method>
</application>
```

WSDL Document and WADL Document Versioning

When the Service System Status in the Services Configuration page is set to *Production* and you attempt to regenerate a WSDL document or a WADL document for a service, PeopleSoft Integration Broker versions and stores the WSDL document or WADL document in the WSDL repository under the following condition: You have previously generated a WSDL document or a WADL document for the same service and new service operations have been added that you selected to include in the new WSDL document or WADL document.

In this case, the system appends a version number to the service namespace to enable unique qualification of elements and attributes in the new document version. The new version number is also appended to the document name. The latest document version is marked as *Default* in the WSDL repository.

When the Service System Status in the Services Configuration page is set to *Development* and you regenerate a WSDL document or a WADL document for a service, any existing document in the repository is overwritten.

Multiple WSDL or WADL versions generated from the same service display in a grid on the Services page and include a timestamp and version for each generated document. Only one of these can be the default version.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

Chapter 9, "Managing Services," Viewing WSDL Documents Generated for Services, page 226

Prerequisites for Providing Services

The following prerequisites exist for providing services:

- The PeopleSoft system must be configured for handling services. Use the Services Configuration page to define the service namespace, schema namespace, target location and service system status.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- If publishing services to a UDDI repository you must:

  Note. You can only publish WSDL documents to a UDDI repository.

  - Have the UDDI server set up, configured and running.
  - Have business entities and categories set up on the UDDI server that you intent to query from the Provide Web Service wizard.
  - Specify the UDDI server and other relevant information within the PeopleSoft system using the Services Configuration-UDDI Configuration page.
- Service operations in services to provide must have any-to-local routing definitions defined.
- There must be a minimum of one service operation associated with the service that you select for which to generate a WSDL document or a WADL document.
- For services that contain service operations with nonrowset-based messages, schemas must exist. The message schema must contain a target namespace. If no target namespace exists an error will occur when the system attempts to generate the WSDL or WADL document.

  PeopleSoft automatically generates schema for services that contain operations with rowset-based messages.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services"

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Specifying UDDI Repositories in PeopleSoft Systems for Providing and Consuming Services"

Chapter 17, "Managing Service Operation Routing Definitions," page 371

Chapter 10, "Managing Service Operations," page 247

Chapter 8, "Building Message Schemas," page 213

Common Elements Used in This Chapter

Description	Description of the service, service operation or WSDL source.
Fault Message	Name of the fault message. Depending on how generated, the name of the fault message can include the version in the following format: *MessageName.Version.*
Operation Type and **Operation**	Type of service operation. See Chapter 10, "Managing Service Operations," Services Operation Types, page 248.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Request Message	Name and version of the request message in the following format: *MessageName.Version.*
Response Message	Name and version of the response message in the following format: *MessageName.Version.*
Routing External Alias	External Alias from the routing definition for a service operation. Unless overridden, defaults to the format *OperationAlias.Version.*
Select	Check the box to select the WSDL or WADL.
Service	Name of the service that contains the service operations to include in the generated WSDL or WADL documents.
Service Operation	Name of the service operation for which to generate WSDL or WADL documents.

Providing Services

This section discusses how to use the Provide Web Service wizard to:

- Select services to provide.
- Select service operations.
- View WSDL documents and WADL documents.
- Specify publishing options.
- View the WSDL Generation Log.

Understanding Using the Provide Web Service Wizard

The Provide Web Service component (IB_WSDLEXP_SRCH) features a wizard you can use to provide web services. You can publish WSDL and WADL documents to the WSDL repository in the PeopleSoft system or external UDDI repositories.

After you generate a WSDL document or a WADL document, the Provide Web Service wizard displays a WSDL URL or a WADL URL for each document you generated. This enables you to access WSDL document or WADL document content using the URL.

In addition, you can modify a WSDL URL to access the WSDL document content using a WSIL URL.

Note. For a service to be available to provide, an any-to-local routing must exist for the service. In addition, there must be a minimum of one service operation associated with the service.

You can use the Provide Web Service wizard to select one or more services for which to generate WSDL documents or WADL documents. A separate document is generated for each service.

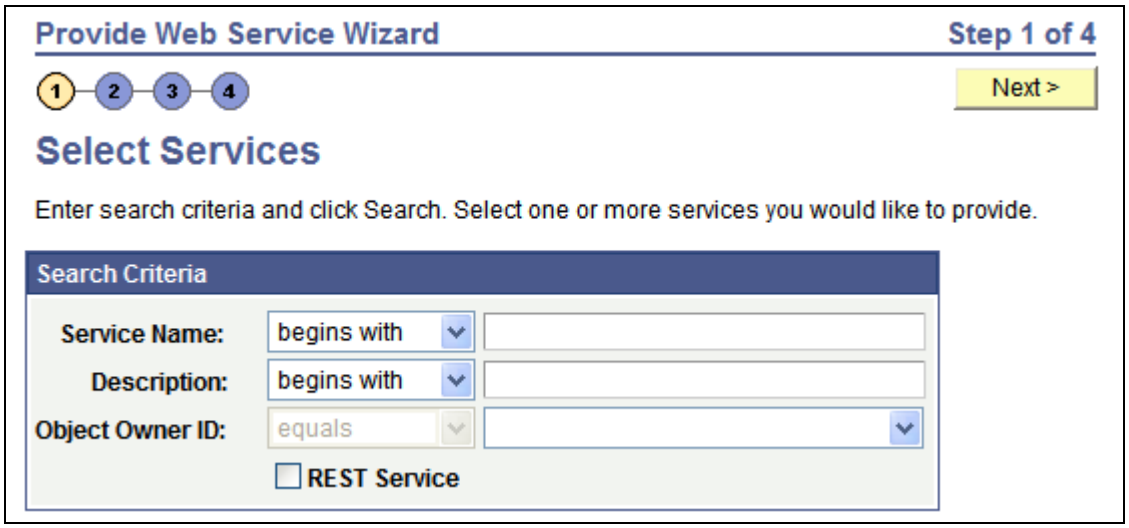
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 20, "Providing Services," Prerequisites for Providing Services, page 500

Step 1: Select Services to Provide

Use the Select Services page (IB_WSDLEXP_SRCH) to search for and select the services that contain the service operations to include the WSDL documents that you generate. The following example shows the page:



Provide Web Service - Select Services to Provide page

You can search by the full or partial service name and service description. You can also search by object owner ID, if one is defined for the service. You can enter one or more of these criteria when performing your search.

To search for REST-based services, you must check the REST Service box. To search for non-REST services, ensure that the REST Service box is clear.

To select services to provide:

1. Access the Provide Web Service Wizard – Select Services page (PeopleTools, Integration Broker, Web Services, Provide Services).
2. Enter search criteria for the services to provide by performing one or more of the following:
   - In the Service Name field, enter a full or partial service name.
   - In the Description field, enter the full or partial description of the service.
   - From the Object Owner ID drop-down list box, select the object owner of the service to provide.
   - Select no search criteria to retrieve a list of all services in the database for which any-to-local routing definitions have been generated.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Check the REST Service box to limit the search for REST-based services. Clear the REST Service box to search for non-REST services.
4. Click the Search button.

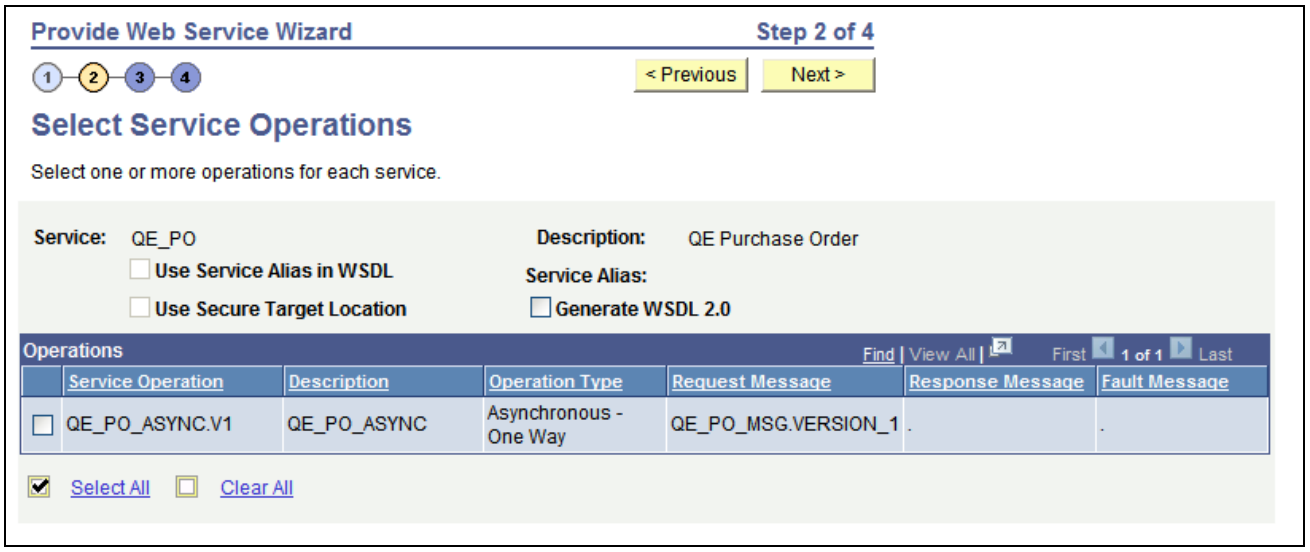
   A Services grid appears that contains the search results.

   The search results only list services which have at least one service operation with an any-to-local routing.
5. Check the box next to each name of the services to provide.

   To clear a selection, check the box again.
6. Click the Next button to proceed to the next step in the wizard, selecting service operations.

Step 2: Select Service Operations

The following graphic shows the Select Service Operations page (IB_WSDL_EXP_OPER):



Provide Web Service - Select Service Operations page

Use the page to select the service operations from each service that you selected in the previous step to include in the WSDL document.

The Use Service Alias in WSDL option on the page enables you to specify that the system use the service alias name in the generated WSDL, rather than the actual service name.

The Use Secure Target Location option enables you to export the WSDL to the URL specified in the Secure Target Location field on the Service Configuration page. If you do not check this option, WSDL is exported to the URL specified in the Target Location field on the Service Configuration page.

By default, the system generates WSDL in WSDL 1.1 format. The Generate WSDL 2.0 option enables you to generate WSDL in WSDL 2.0 format.

If you are attempting to provide a different version of an existing WSDL document, the Service System Status mode currently set, Production or Development, may impact your results.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 20, "Providing Services," WSDL Document and WADL Document Versioning, page 500.

To select service operations to include in a WSDL document:

1. Check the box next to each service operation to include.

   To clear a selection, check the box again.

2. (Optional) Select the Use Service Alias in WSDL check box to include the alias name for the service in the generated WSDL instead of the actual service name.

3. (Optional) Select the Use Target Location check box to export WSDL to the secure target location URL defined on the Service Configuration page.

4. (Optional) Select the WSDL 2.0 check box to generate WSDL in WSDL 2.0 format.

5. Click the Next button to proceed to the next step in the wizard.

The next step to providing WSDL documents is previewing the WSDL document that will be provided.

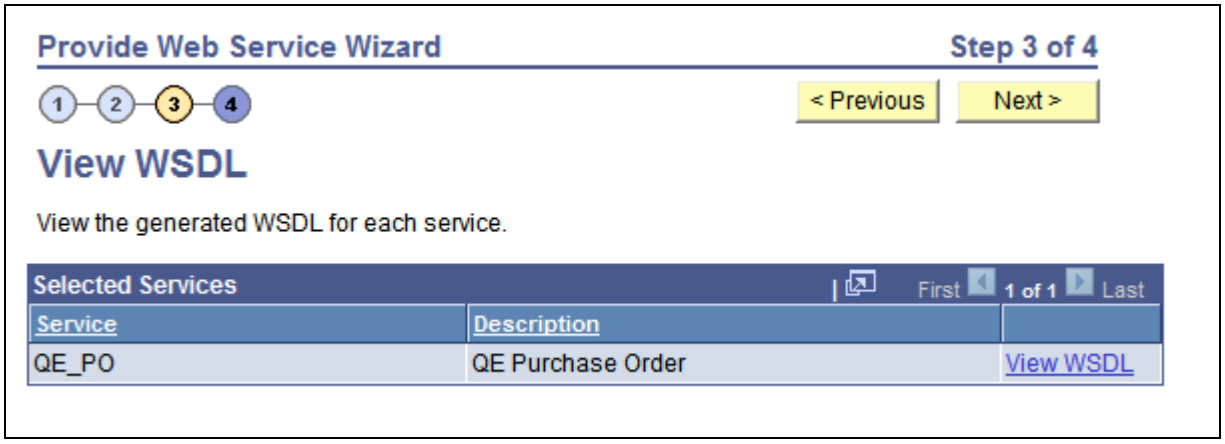
See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Configuring PeopleSoft Integration Broker for Handling Services," Target Locations

Step 3: View WSDL Documents or WADL Documents

After you select the service operations to include in a WSDL document or a WADL document, you can preview the WSDL or WADL before actually publishing it.

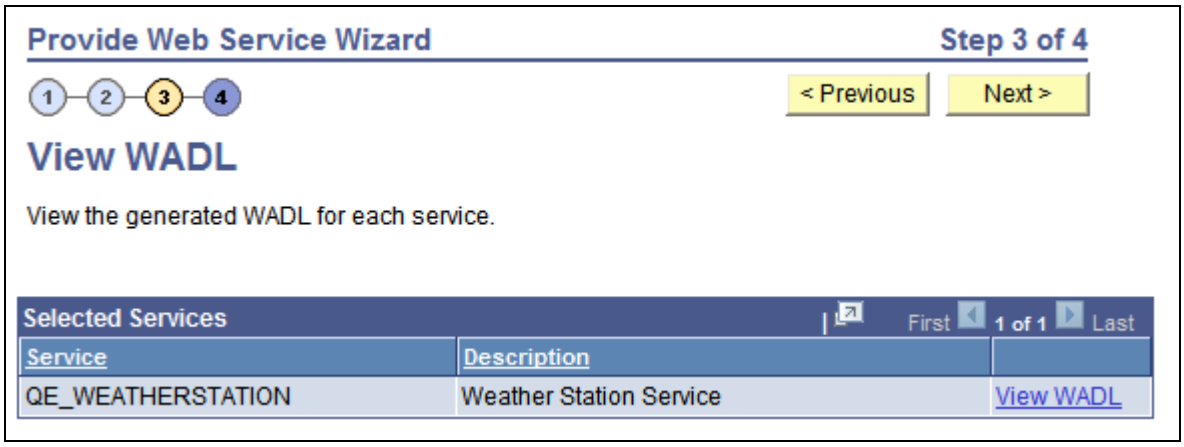
The following graphic shows the View WSDL page (IB_WSDLEXP_PVIEW). Use the page to preview a WSDL document before you actually generate it.



Provide Web Service - View WSDL page

When you are generating a WADL document the page that appears is the View WADL page shown in the following example:

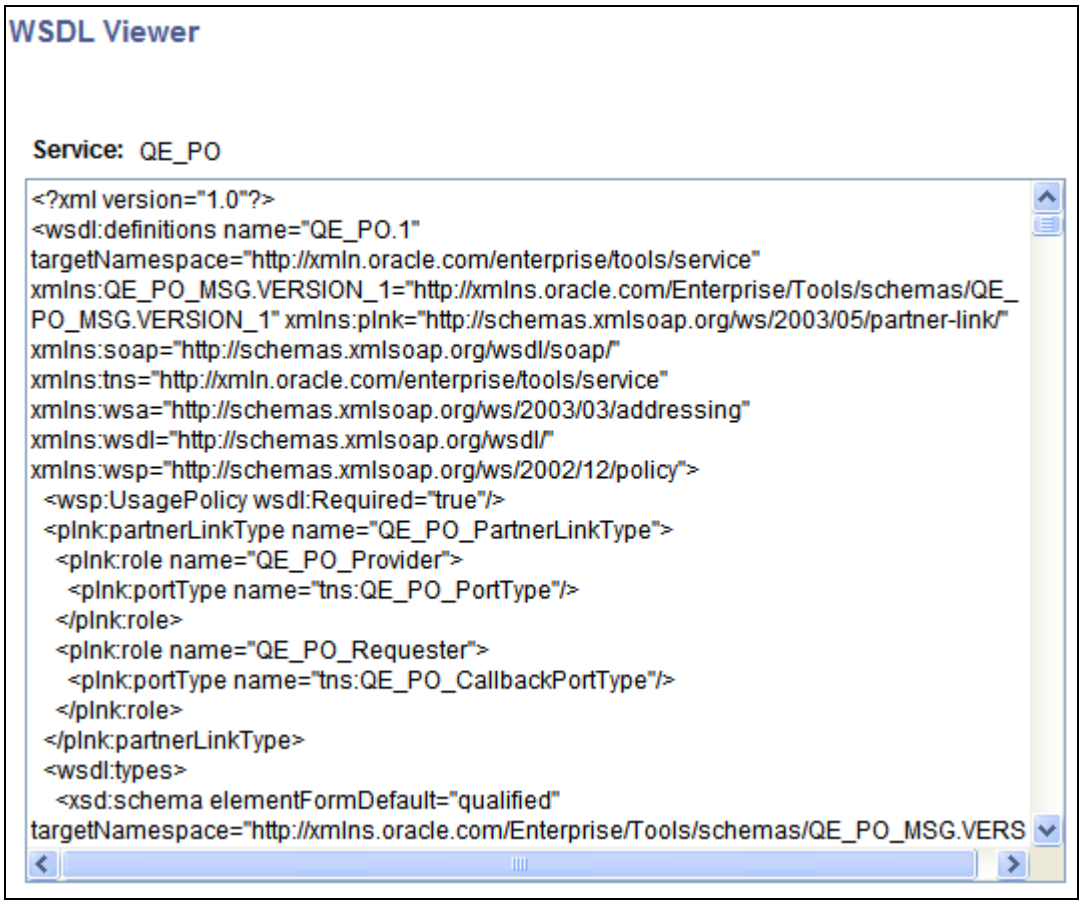
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Provide Web Service – View WADL page

Each service for which a WSDL document will be generated is listed. Click theView WSDL link to view the WSDL document for each service that you have selected.

When you click the View WSDL link or the View WADL link, the WSDL or WADL displays in the WSDL Viewer page (IB_WSDEXPPVIEW_SEC) or the WADL Viewer page, respectively. The following example shows the WSDL document for the *QE_PO* service in the WSDL Viewer page.



WSDL Viewer page displaying WSDL for the QE_PO service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

To preview WSDL documents and WADL documents:

1. Click the View WSDL link or View WADL link for a service.

   The WSDL document or WADL document for the service appears in the WSDL Viewer or WADL Viewer.

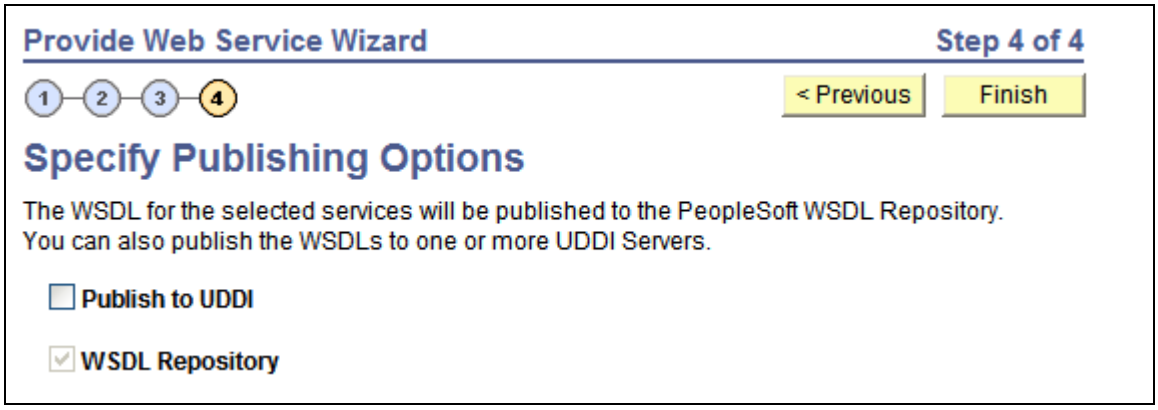
2. Click the Return button to return to the View WSDL or View WADL page.

3. Click the Next button to proceed to the next step in the wizard.

The next section discusses the next step to providing a service, selecting the location of where to publish WSDL documents and WADL documents.

Step 4: Specify Publishing Options

After you preview the WSDL document or WADL document, use the Specify Publishing Options page (IB_WSDLEXP_LOC) to specify the publish location of the generated documents. The following graphic shows the page when you are publishing a WSDL document:



Provide Web Service - Specify Publishing Options page

By default the system publishes WSDL documents and WADL document to the PeopleSoft WSDL repository.

Note. WADL documents can only be published to the WSDL Repository.

Select the Publish to UDDI check box to publish the WSDL to a UDDI repository in addition to the PeopleSoft WSDL repository.

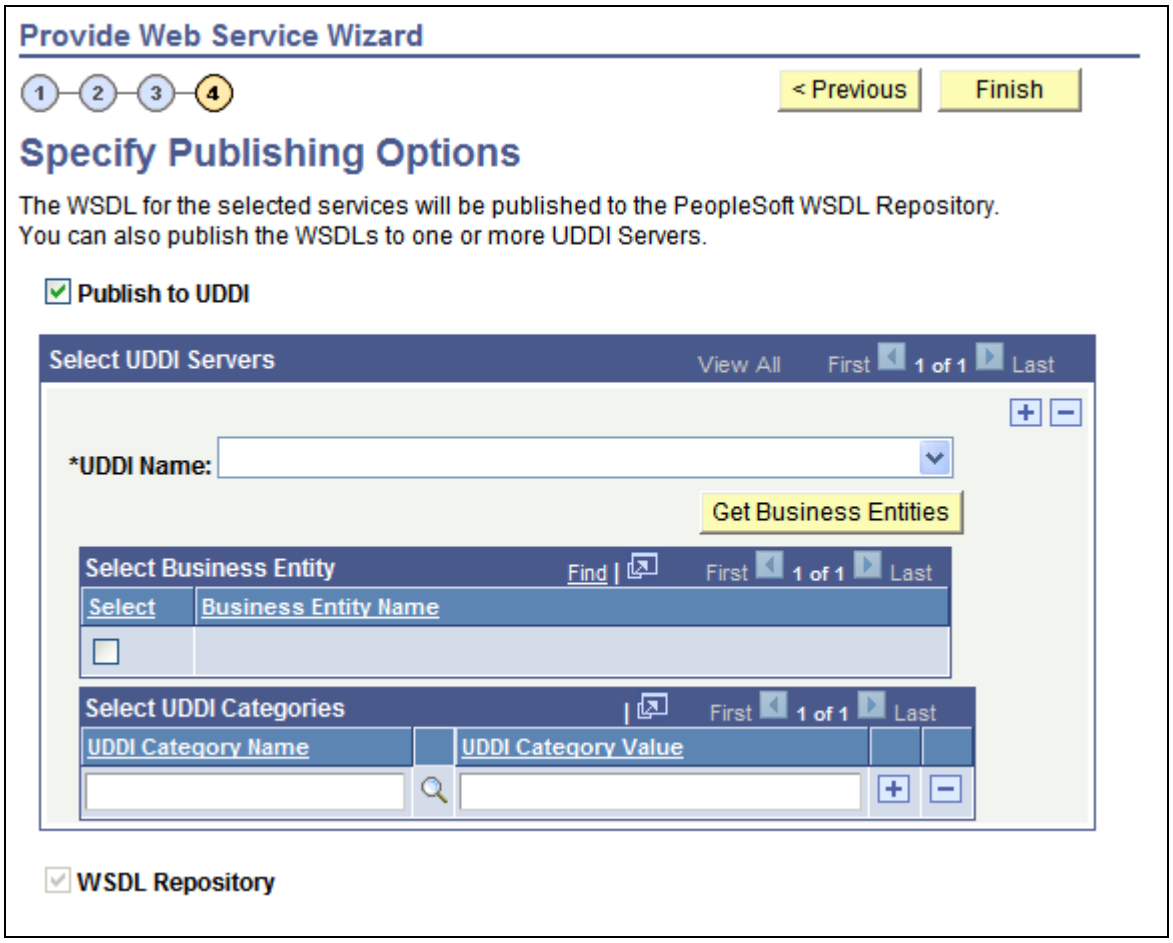
Providing WSDL Documents to UDDI Repositories

Before providing a WSDL document to a UDDI repository, you must configure the UDDI repository in the PeopleSoft system.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Specifying UDDI Repositories in PeopleSoft Systems for Providing and Consuming Services."

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

When you select the Publish to UDDI check box, the Select UDDI Servers box appears where you specify the UDDI repository to which to publish the WSDL. The following graphic shows the Select UDDI Servers section:



Select UDDI Servers section of the Specify Publishing Options page.

To provide a WSDL document to a UDDI repository:

1. From the UDDI Name drop-down list box, select the UDDI server to which you are publishing the WSDL.
2. Click the Get Bus. Entities button.

   The Select Business Entity section lists the business entities that are available to select for the UDDI server.
3. Check the box next to each business entity name to include.
4. Click the UDDI Category Name lookup button to display a list of UDDI categories and select a UDDI category. Click the OK button.
5. In the Category Value field, enter a value for the category.
6. To add additional categories, in the Select UDDI Categories section, click the plus button to add a row and repeat step 5 and step 6.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

7. Click the Finish button.

   The Results page appears and displays the WSDL generation log.

Step 5: View the WSDL/WADL Generation Log

Use the Results page shown in the following example to view the WSDL Generation Log:

The WSDL Generate Log box shows that WSDL has been exported for the QE_PO service.

The WSDL/WADL Generation Log provides the name of the services and URL for each WSDL document or WADL document generated.

You can cut and paste the URL into a browser to access the document. You can also access the document using the WSDL repository.

To provide another service, click the Provide Another Service button and return to step 1 of the wizard.

When providing non-REST service operations, a Generate SOAP Template link appears on the page. Click the link to access the Generate SOAP utility to generate SOAP message templates for request messages, response messages and fault messages found in the WSDL document. You can then use the templates to test SOAP messages in the Handler Tester, Transformation Test Tool and Send Master utilities.

See Also

PeopleTools 8.52: Integration Broker Testing Utilities and Tools, "Using the Generate SOAP Template Utility"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Accessing Generated WSDL Documents and WADL Documents

This section discusses how to:

- Use WSDL and WADL URLs to access generated WSDL and WADL documents.
- Use the WSDL repository to access generated WSDL and WADL documents.

Using WSDL and WADL URLs To Access Generated WSDL and WADL Documents

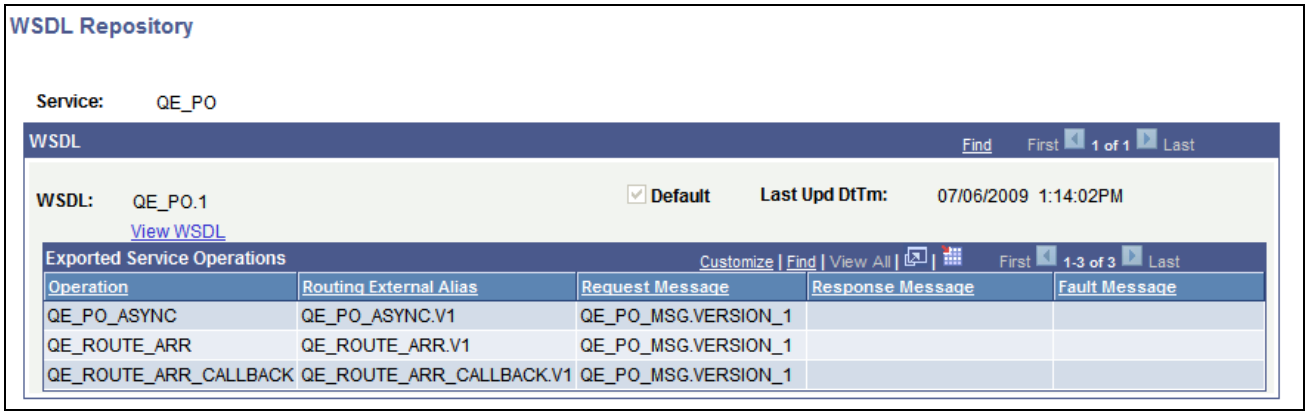
The last page of the Provide Web Service wizard, the Confirm Results page (IB_WSDLEXP_RSLTS), displays a WSDL/WADL generation log. The text box contains WSDL URLs for each WSDL document you generated or WADL URLs for each WADL document you generated.

To access a document copy the WSDL URL or WADL URL from the generation log and paste it into a browser of your choice to access the document.

Using the WSDL Repository to Access Generated WSDL and WADL Documents

Access the WSDL Repository page (PeopleTools, Integration Broker, Integration Setup, Services. Click the View WSDL link or the View WADL link.

The following example shows the WSDL Repository page:



WSDL Repository page for the QE_PO service.

The WSDL Repository page lists the WSDL or WADL documents that exist for a service.

The previous example shows the two WSDL documents, *QE_PO_ASYNC* and *QE_ROUTE_ARR*, that were exported using the Provide Web Service wizard.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The Default option on the page is selected and disabled. This indicates that the WSDL shown is the default WSDL document for the service. The WSDL document that is last generated is the default version for the service.

Note. Default WSDL documents for a service are used only when the service system status is *Production.*

To change the default version, you must provide the service again, and choose a different set of service operations to include in the new default version.

Descriptions of the other fields displayed on this page are discussed at the beginning of this chapter.

See Chapter 20, "Providing Services," Common Elements Used in This Chapter, page 501.

To access the WSDL document, click the View WSDL link. the WSDL Viewer page appears and displays the content of the WSDL document. The following example shows the WSDL Viewer displaying the *QE_PO.1* WSDL document:

The QE.PO.1 WSDL document

Click the Return button to return to the WSDL Repository page.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 20, "Providing Services," WSDL Document and WADL Document Versioning, page 500

Deleting WSDL and WADL Documents

This section discusses how to delete WSDL and WADL documents.

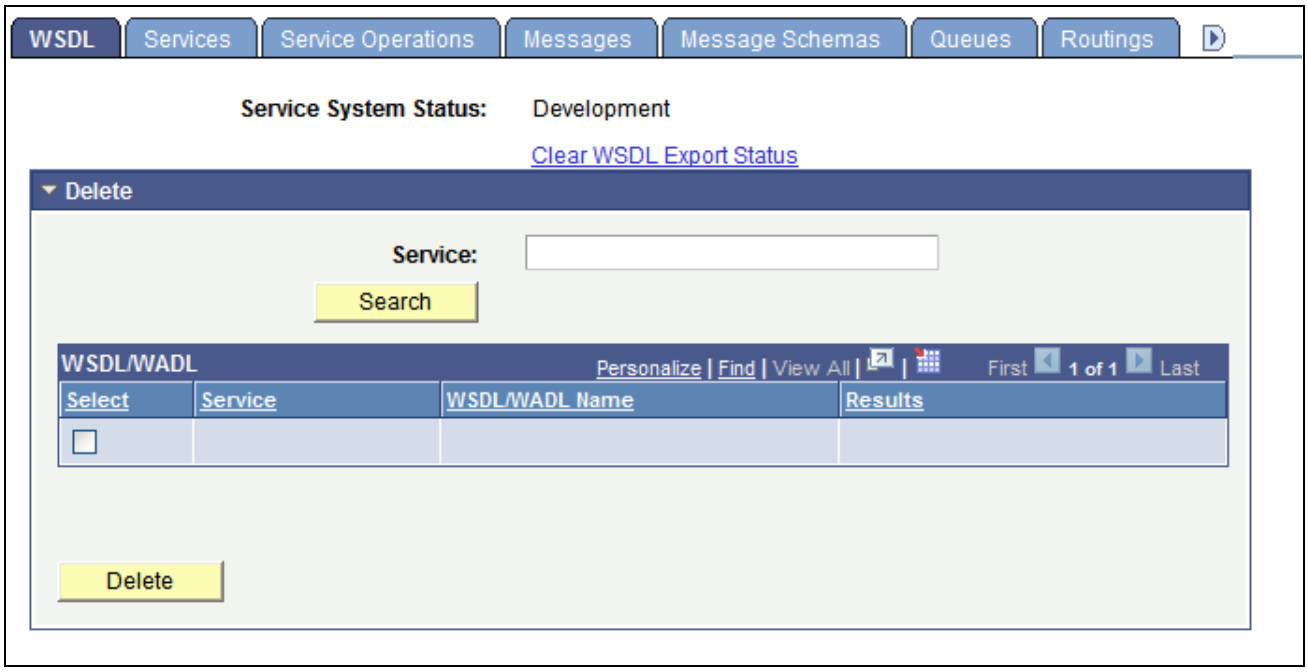
Understanding Deleting WSDL Documents

The service system status affects the ability to delete WSDL and WADL documents.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Understanding Configuring PeopleSoft Integration Broker for Handling Services and *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services," Using the Service Configuration Page to Set Service Configuration Properties.

Deleting a WSDL or WADL Document

Use the Service Administration-WSDL page to delete WSDL documents or WADL documents generated for a service. The following example shows the page:



Service Administration-WSDL page

To delete a WSDL or WADL document

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Services Administration - WSDL page (select PeopleTools, Integration Broker, Service Utilities, Service Administration).
2. Click the arrow next to the Delete section header to expand the section.
3. In the Service field, enter the name of the service that contains the WSDL document or WADL document to delete.
4. Click the Search button.

   Search results appear in the WSDL/WADL results grid.
5. In the results grid, select the check box next to the WSDL document or WADL document to delete.
6. Click the Delete button.

The Clear WSDL Export Status link that appears on this page is discussed elsewhere in this PeopleBook.

See Chapter 6, "Managing Messages," Resolving Inconsistencies in Exported WSDL and WADL Documents, page 121.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 21

Consuming Services

This chapter discuses how to:

- Consume services.
- Access integration metadata for consumed services.

Understanding Consuming Services

PeopleSoft Integration Broker can consume external services by way of consuming WSDL documents and generates PeopleSoft integration metadata, so that PeopleSoft applications can invoke outbound synchronous and outbound asynchronous services.

PeopleSoft Integration Broker features a Consume Web Service wizard that steps you through the task of consuming WSDL documents.

Understanding the Consume Web Service Wizard

This section discusses the Consume Web Service wizard.

Consume Web Service Wizard Features

The Consume Web Service wizard supports:

- WS-interoperability standards for WSDL.
- Consumption of WSDL version 1.1 documents.
- Consuming WSDL with SOAP, HTTP–GET/POST bindings.
- Nested URIs to resolve WSDL fragments.

Operation Types Supported

You can consume WSDL documents for the following service operation types:

- Synchronous.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Asynchronous one-way.
- Asynchronous request/response.

Sources for Consuming WSDL Document

You can use the Consume Web Service wizard to consume WSDL documents from the following sources:

- UDDI repositories.
- WSDL URL.
- WSIL registries.
- File.
- Legacy PeopleSoft WSDL documents.

Integration Metadata Created by the Consume Web Service Wizard

The Consume Web Service wizard creates the following integration metadata in the PeopleSoft system from the consumed WSDL documents:

Note. The internal name is the name that the PeopleSoft system assigns to the metadata and is the definition name that appears in the PeopleSoft system.

Metadata	*Internal Name*	*Version*	*Comments*
Service definitions	System generated name in the format:<*WSDL service name*>_ <*four random generated numbers*>.	Version one, denoted as: *.V1*	NA
Service operation definitions	System generated name in the format:<*WSDL Service Operation Name*>_ <*four random generated numbers*>.	Version one, denoted as: *.V1*	NA

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Metadata	*Internal Name*	*Version*	*Comments*
Message definitions	System-generated name in the format: *M<unique number>*. For example: *M120508438.*	Version one, denoted as: *.V1*	By default, the system creates document type messages. Includes request messages, response messages. and fault messages, as appropriate. You can rename the system-generated message names in the wizard using more meaningful names. The consume process also generates schemas for each message. All schemas are unstructured. If you chose not to create document type messages, it creates nonrowset-based or container messages with references to schemas. Additional information about messages that the Consume Web Service wizard generates is featured in this section.
Document definitions	The following metadata names are system generated: • Package name: *<Service_Naame>*. • Document name: *<WSDL_ Operation (input/output)_Message_Name>*.	Version one, denoted as: *.V1*	NA
Routing definitions	System-generated name in the format *~IMPORTED~<unique number>*. For example: *~IMPORTED~14857.*	Version one, denoted as: *.V1*	System creates a point-to-point routing.

Messages Generated

By default, the system creates document type messages based on the message schema(s). Documents generated are restricted. To modify the document, you must use the Restricted Documents page to clear the restricted status.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

If the system cannot create document type messages or if you chose not to create document type messages, it creates nonrowset-based or container messages with references to schemas.

See Also

PeopleTools 8.52: PeopleSoft Documents Technology, "Securing Documents"

PeopleTools 8.52: PeopleSoft Documents Technology, "PeopleSoft Documents Technology Preface"

Fault Messages

By default, the Consume Web Service wizard builds document type messages and will create a single fault message.

To read faults in document type messages generated by the Consume Web Service wizard, use the GetContentString() method of the Message object.

To process multiple faults, you can examine the stored WSDL generated by the wizard.

Another option to capturing and reading multiple faults is to clear the Build Document Messages option in Step 1 of the wizard so that the system generates nonrowset–based messages. If generating nonrowset-based messages, PeopleSoft Integration Broker creates a part message for each fault in the WSDL operation. The system then creates a container message and places the fault part messages in the container. The container message is assigned as a fault message to the created service operation.

Multiple Root Elements in Message Schemas

In a WSDL document, the schema defined in the <types> section may have multiple root elements, corresponding to multiple messages used by one or more operations. When the PeopleSoft system consumes such a WSDL document, the entire message schema contained in the WSDL document gets associated with each of the service operation messages when the PeopleSoft system generates the integration metadata.

Use the element name that appears in the Comments text box of the message definition to construct the XML data for the correct schema fragment in the message.

Delivered Queues and Nodes

PeopleSoft delivers a queue, *WSDL_QUEUE*, and a node, *WSDL_NODE*, that the Consume Web Service wizard uses as defaults.

You may use these objects or select other existing queues or nodes.

Binding Style of Consumed WSDL Documents

The binding style of consumed WSDL documents appears in the service operation definition for the consumed service. The Default Service Operation Version section of the Service Operations page features a Comments box. The binding style appears in that box.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Working with Asynchronous Request/Response Service Operations

If a WSDL document has two port types with a single input message in each operation, the Consume Web Service wizard displays step where you can convert a pair of asynchronous one-way operations to a single asynchronous request/response operation. In this step you can special the request and callback service operations and convert the operation.

Prerequisites for Consuming Services

To consume services you must set properties in the Service Configuration component as follows:

- Use the Services Configuration page to define the service namespace, schema namespace, target location, and service system status.

  See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Configuring PeopleSoft Integration Broker for Handling Services."

- If consuming services from a UDDI repository, you must first specify the UDDI server and other relevant information within the PeopleSoft system using the Services Configuration-UDDI Configuration page.

  See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Specifying UDDI Repositories in PeopleSoft Systems for Providing and Consuming Services."

To consume WSDL from a file, you must set the PS_FILEDIR environment variable on the machine on which the application server resides. If you do not set this variable and attempt to consume WSDL from a file, you will receive an error that the WSDL cannot be parsed.

See Appendix D, "Setting PS_FILEDIR and PS_SERVDIR Environment Variables," page 655.

Before consuming services evaluate your security requirements.

Common Elements Used in This Chapter

Description	Description of the WSDL source.
Endpoint	According to the W3C, "An association between a binding and a network address, specified by a URI, that may be used to communicate with an instance of a service." A URI that accepts messages containing document-oriented or procedure-oriented information.
Internal Message	Name of the consumed request message, response message or fault message in the PeopleSoft system
Internal Operation	Name of the consumed service operation in the PeopleSoft system
Internal Service	Name of the consumed service in the PeopleSoft system

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Next	Click the button to advance to the next step in the wizard.
Operation Type/Operation	Type of service operation. See Chapter 10, "Managing Service Operations," Services Operation Types, page 248.
Previous	Click the button to go back to the previous step in the wizard.
Select	Check the box to select the WSDL service or WSDL operation on which to perform an action.
View WSDL	Click the link to view the WSDL document for a service from the WSDL source.
WSDL Port	According to the W3C: • "A port indicates a specific location for accessing a service using a specific protocol and data format." • "The network address of an endpoint and the binding to which it adheres."
WSDL Fault Message	Name of the fault message specified in the WSDL document that the PeopleSoft system is consuming.
WSDL Request Message	Name of the request message specified in the WSDL document that the PeopleSoft system is consuming.
WSDL Response Message	Name of the response message specified in the WSDL document that the PeopleSoft system is consuming.
WSDL Service	The name of the external service in the WSDL document that the PeopleSoft system is consuming.
WSDL Source	Indicates the source of the service that the PeopleSoft system is consuming.
WSDL URL	Displays the WSDL URL, WSIL URL, File name or UDDI server name of the WSDL source.

See Also

http://www.w3.org/TR/wsdl

Using the Consume Web Service Wizard

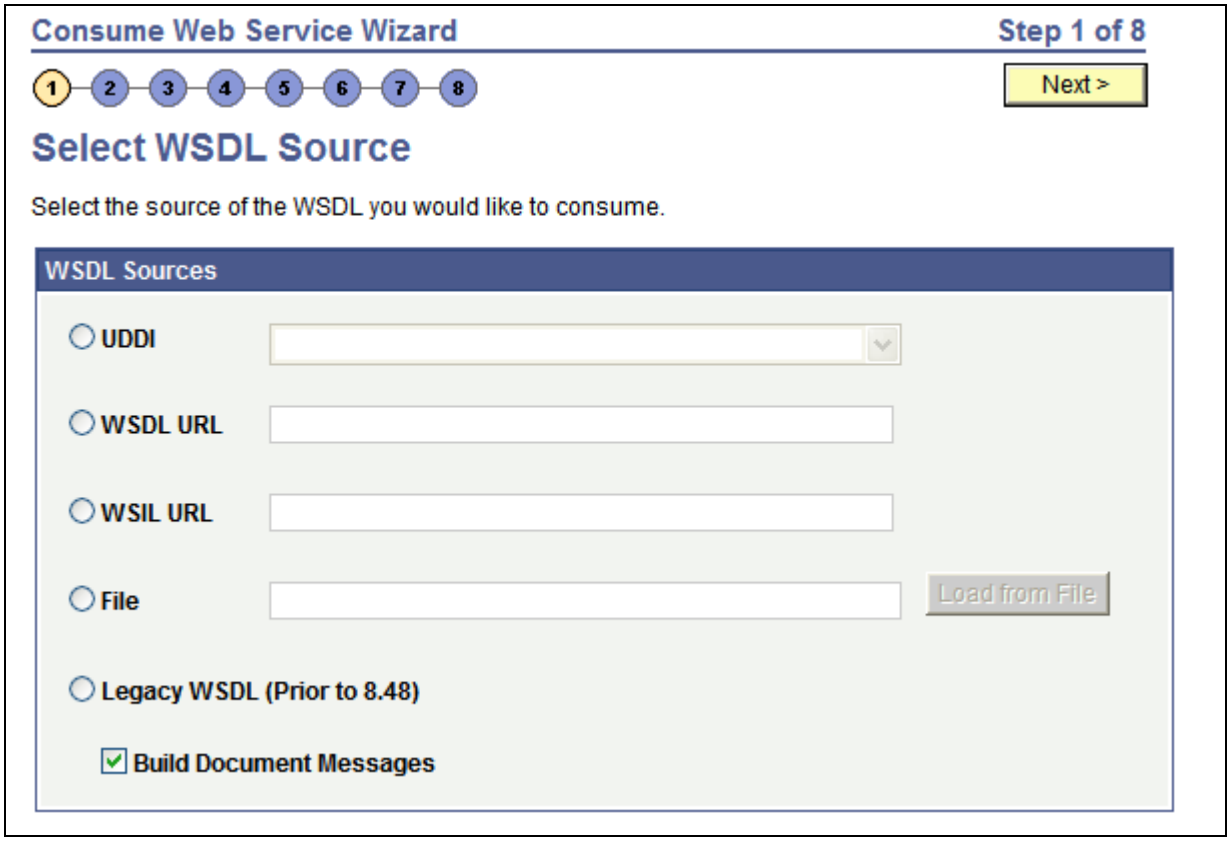
This section discusses how to use the Consume Web Service wizard to:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Select the WSDL source.
- Select a service.
- Select a service port.
- Select service operations.
- Convert asynchronous operations.
- Rename operation messages.
- Select a queue for asynchronous operations.
- Select the receiver node.
- Confirm and view results.

Step 1: Select WSDL Source

Use the Select WSDL Source page (IB_WSDL_IMP_LOC) to select the source for consuming a WSDL document. The following example shows the page:



Select WSDL Source page

To select the WSDL source:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Access the Select WSDL Source page (PeopleTools, Integration Broker, Web Services, Consume Services).
2. Select the radio button next to one of the following values and enter the information specified:

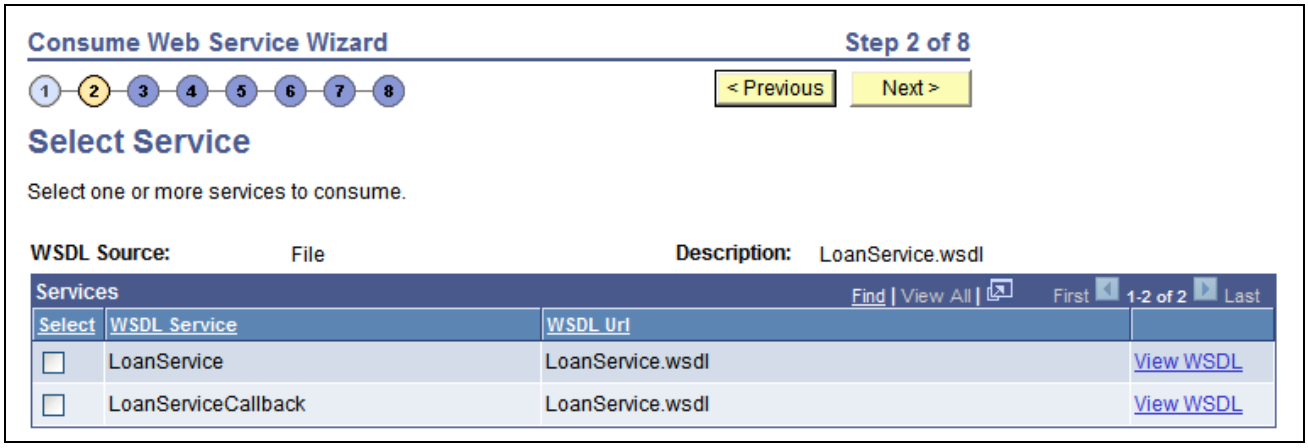
UDDI	To consume a WSDL document from a UDDI repository, select the UDDI radio button. After you select the radio button, select the UDDI repository from the drop-down list box. To use this option, you must first specify the UDDI repository in the PeopleSoft system. See Chapter 21, "Consuming Services," Prerequisites for Consuming Services, page 519.
WSDL URL	To consume a WSDL document from a WSDL URL, select the WSDL URL radio button. After you select the radio button, enter the URL in the WSDL URL field.
WSIL URL	To consume a WSDL document from a WSIL URL, select the WSIL URL radio button. After you select the radio button, enter a URL in the WSIL URL field.
File	To consume a WSDL document from a file, perform one of the following actions: • In the File field, enter the path and file name. For example: *c:\temp\sample.wsdl.* • Browse for the file location and name. a. Click the Load from File button. A file upload page appears. b. Click the Browse button to search for and select the file location and name. c. Click the Upload button. The Select WSDL Source page appears with the file location and name populated in the File field.
Legacy WSDL (Prior to 8.48)	Select this option to consume a PeopleSoft WSDL document generated from releases prior to PeopleTools 8.48.

3. At the Build Document Messages box:
   - Select the box to build document type messages for the consumed service. (Default.)
   - Clear the box to build nonrowset-based messages for the consumed service.
4. Click the Next button to proceed to the next step in the wizard.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step 2: Select Service

Use the Select Service page (IB_WSDL_IMP_SVC) to select the services to consume. The following example shows the page:



Select Service page

Before selecting services to consume, you can click the View WSDL link to view the WSDL for each service. The WSDL document opens in a browser. Close the browser when you have finished viewing the WSDL document.

WSDL documents that the PeopleSoft system consumes from pre-PeopleTools 8.48 systems display in an edit box.

To select services to consume:

1. Check the box next to each service to consume.

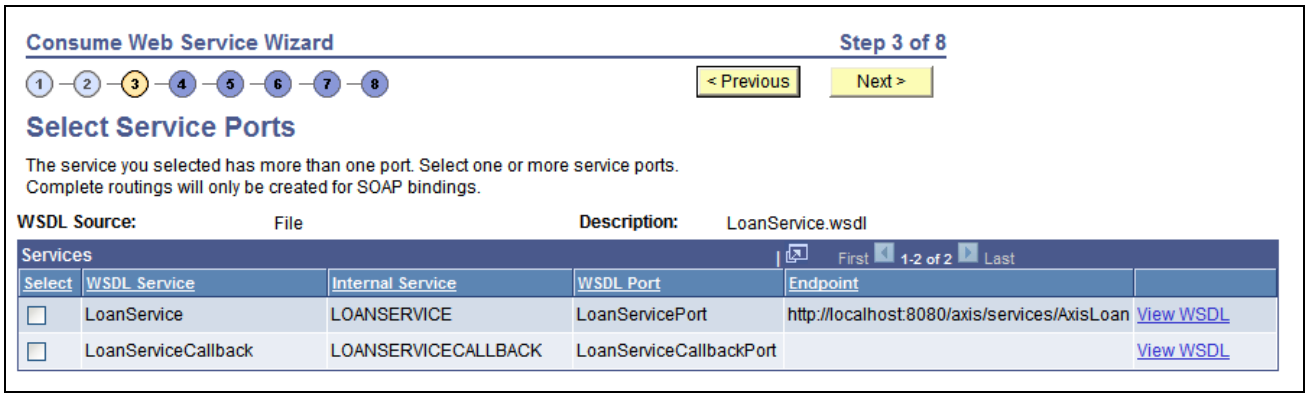
   To clear a selection, check the box again.

2. Click the Next button to proceed to the next step in the wizard.

Step 3: Select Service Ports

If a service you select has more than one port, the Select Service Ports page (IB_WSDL_IMP_SVC2) appears. The following example shows the page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



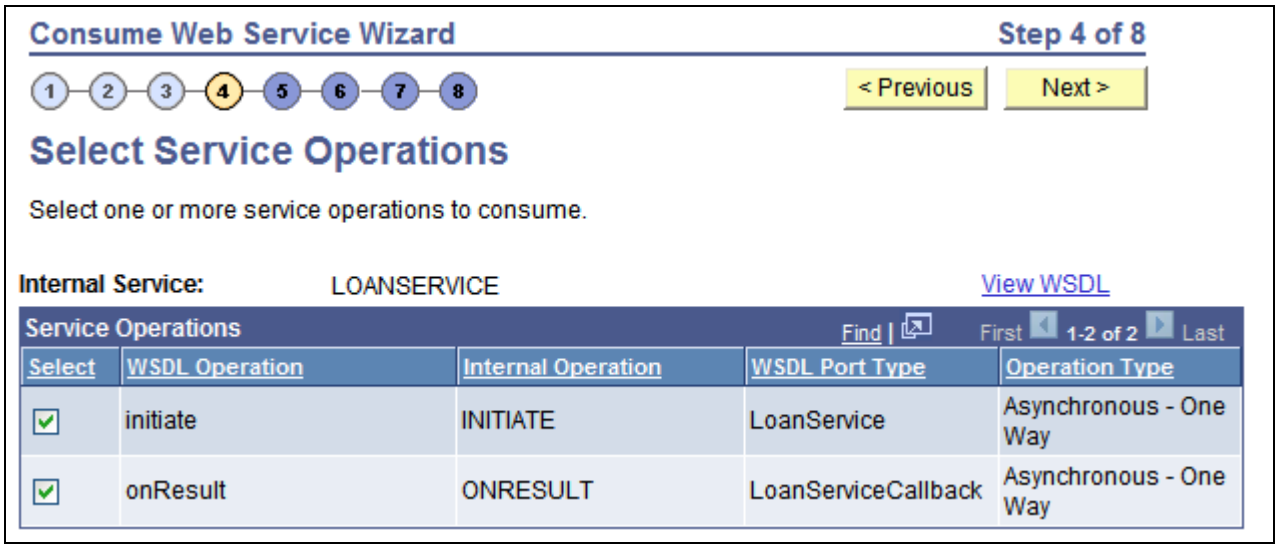
Select Service Ports page

Multiple port options only appear when you are working with asynchronous request/response operations in a WSDL document or the service has multiple bindings. Typically, when working with this operation type, you select both options.

To select service ports, in the Select column, check the box next to each service.

Step 4: Select Service Operations

Use the Select Service Operations page (IB_WSDL_IMP_OPR) to select the operations in the selected service to consume. The following example shows the page:



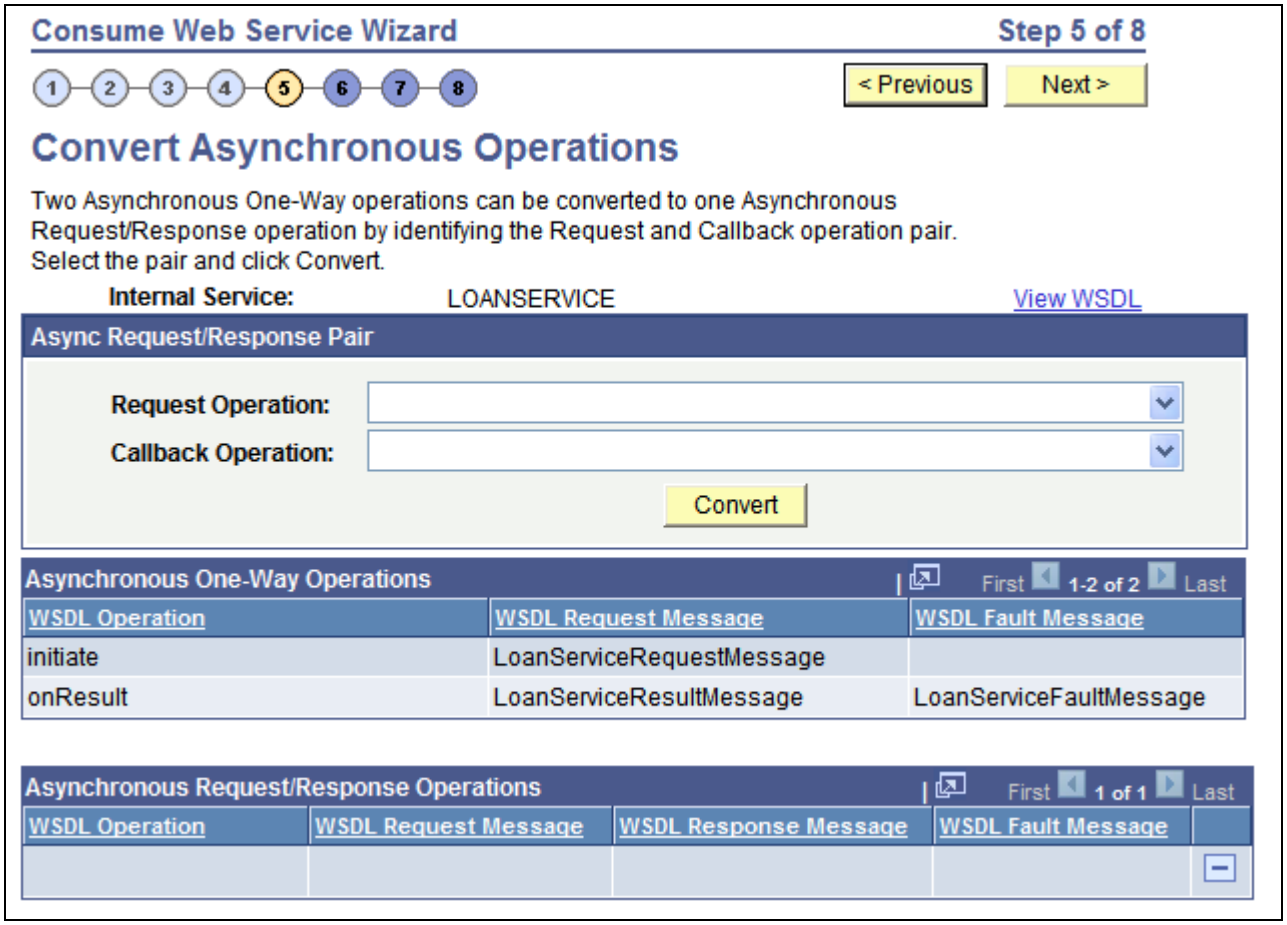
Select Service Operations page

To select service operations to consume, in the Select column, check the box next to each service operation to consume.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Step 5: Convert Asynchronous Operations

The Convert Asynchronous Operations page (IB_WSDL_IMP_ASYNOP) appears when the system detects that you are consuming two asynchronous one-way operations. The two asynchronous one-way operations appear in the Asynchronous One-Way Operations section on the page as shown in the following example:



Convert Asynchronous Operations

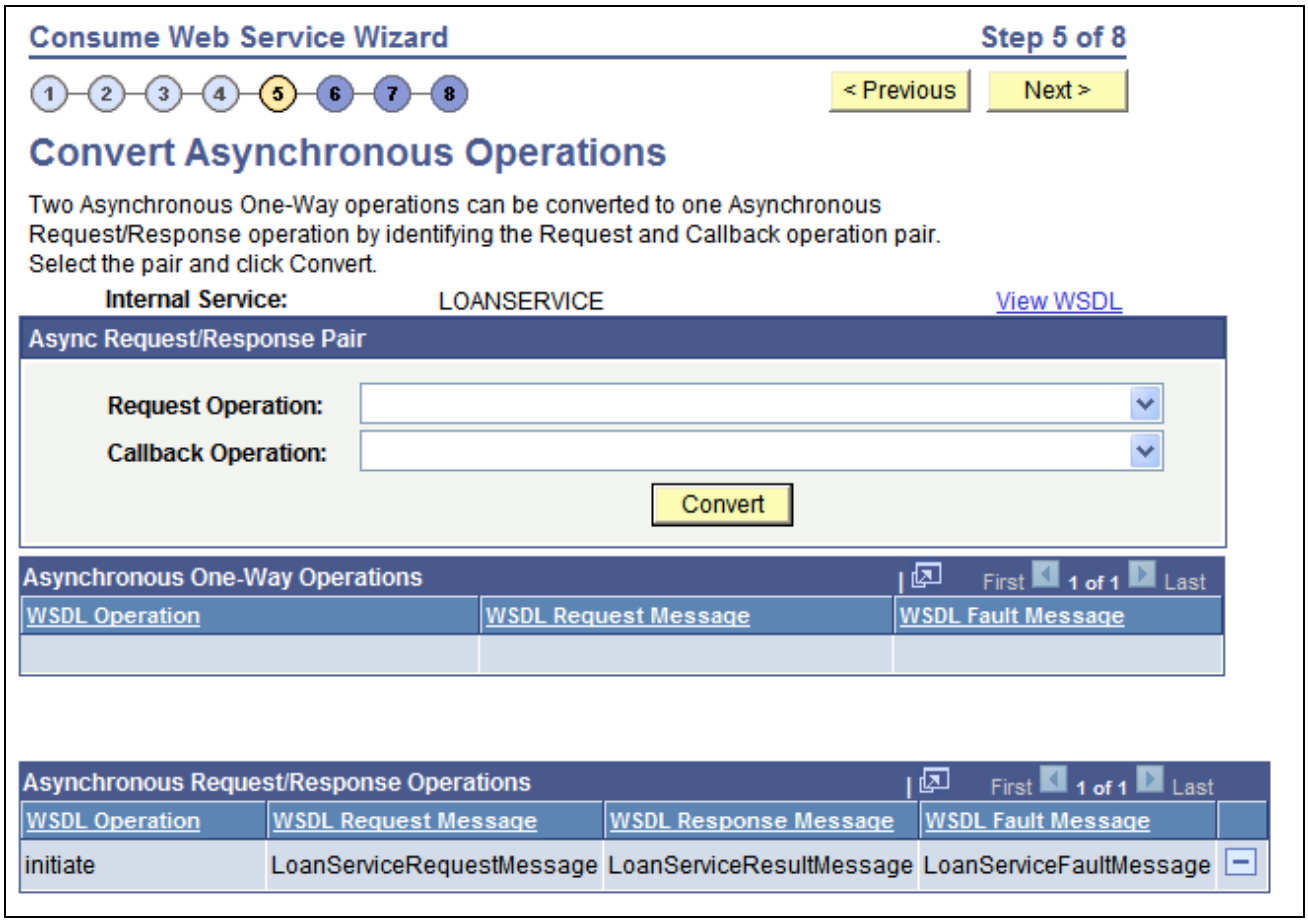
This page enables you to convert the two operations into a single asynchronous request/response operation type.

WSDL specification 1.1 has no standards for specifying an asynchronous request/response operation. Hence, the Consume Web Service wizard is not able to automatically detect such operations in a WSDL 1.1 document.

To make this conversion, you must specify the request operation and the callback operation, using the Request Operation and Callback Operation fields on the page. The system populates the possible values to select in each field from the operation selected.

After you make the conversion the new asynchronous request/response operation appears in the Asynchronous Request/Response Operations section of the page. The following example shows the Convert Asynchronous Operations page after making such a conversion:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Convert Asynchronous Operations page after the operation conversion

To convert two one-way asynchronous operations into one asynchronous request/response operation:

1. From the Request Operation drop-down list box, select the request operation.
2. From the Callback Operation drop-down list box, select the callback operation.
3. Click the Convert button.

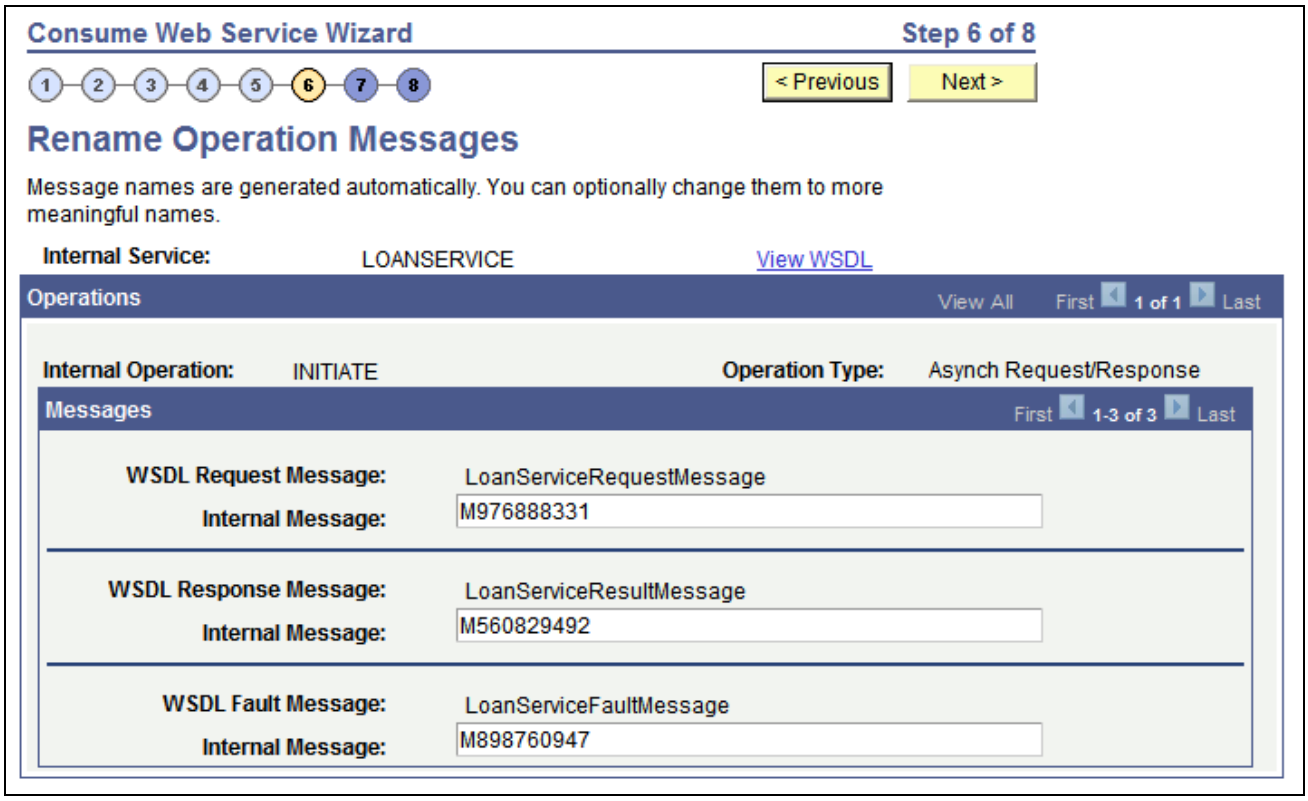
The single operation appears in the Asynchronous Request/Response Operations grid at the bottom of the page.

Clicking the minus button (-) at the end of a data row in the Asynchronous Request/Response grid undoes the conversion.

Step 6: Rename Operation Messages

The Rename Operation Messages page (IB_WSDL_IMP_MSGS) is shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

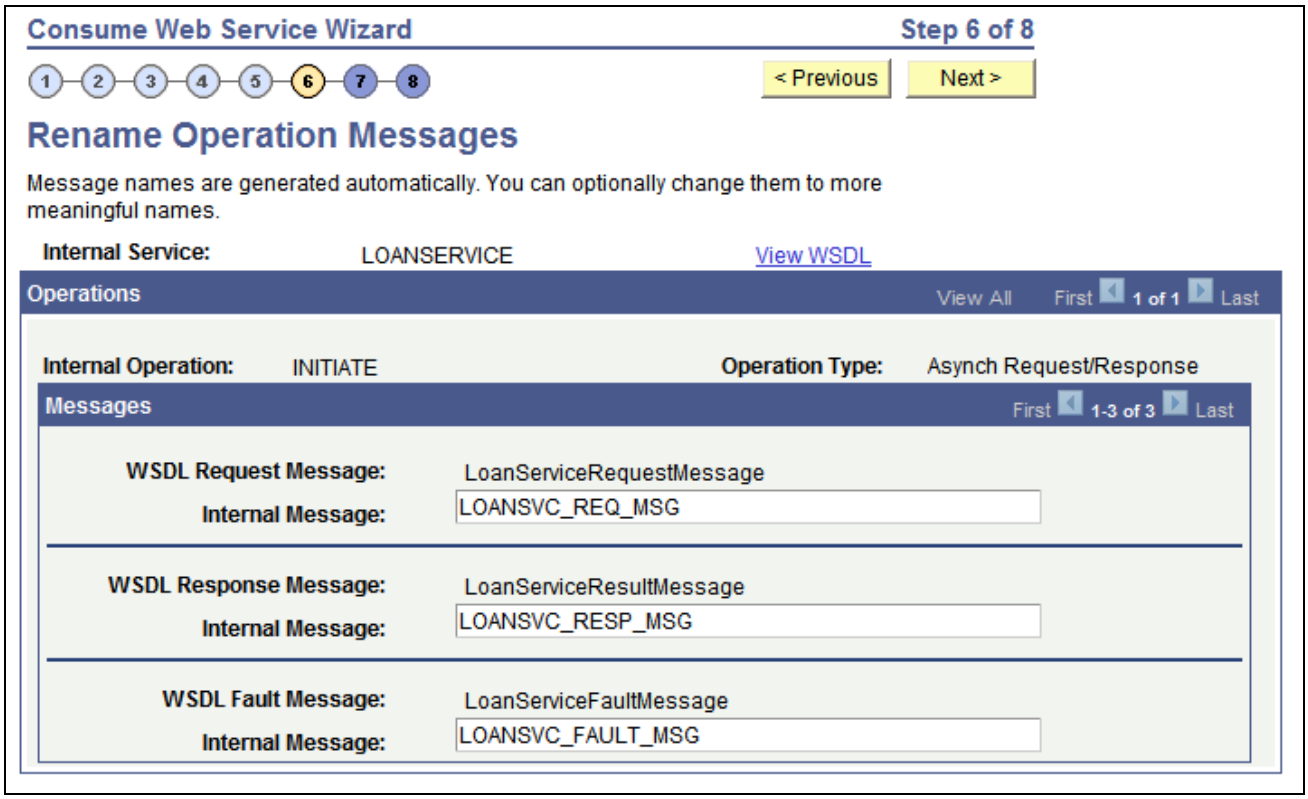


Rename Operation Messages page

When the system creates request, response and fault messages from the consumed service, it provides them with system-generated names. The previous example shows system-generated names appearing for all messages.

Use the page to rename the messages to more meaningful names. The following example shows all messages renamed:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Renamed service operation messages

To rename operation messages:

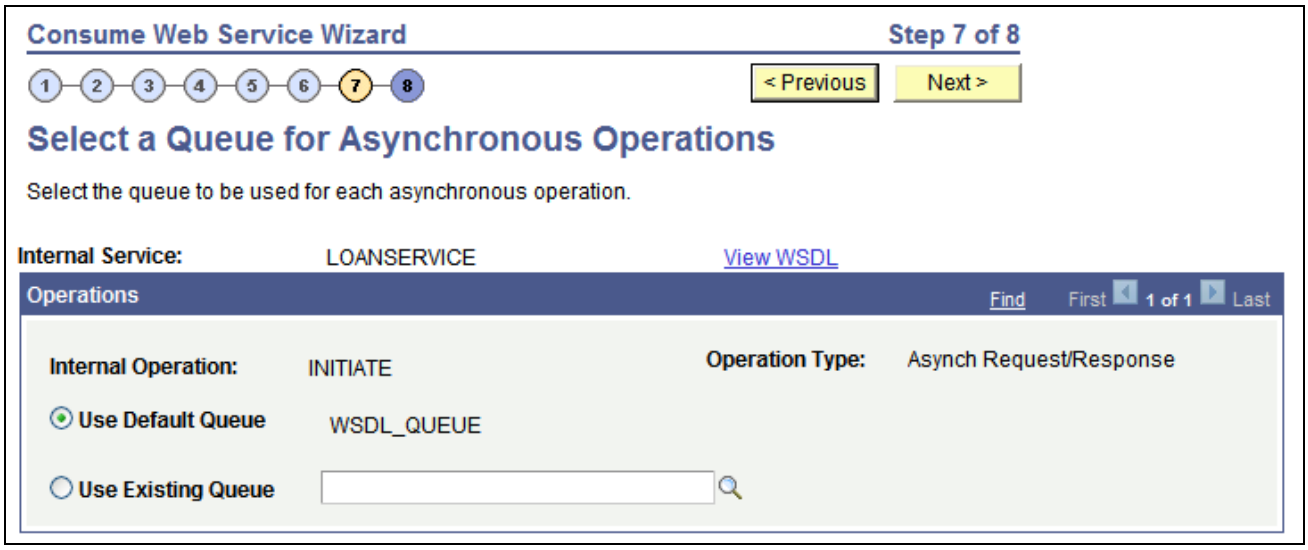
1. Clear the system-generated name from a message name field.
2. Enter a new name in the field.
3. Click the Next button to proceed to the next step in the wizard.

The system checks whether the user-entered message name already exists in the database and displays an error if the name exists.

Step 7: Select a Queue for Asynchronous Operations

The Select a Queue for Asynchronous Operations page (IB_WSDL_IMP_QUEUE) appears only when you are consuming asynchronous one-way and asynchronous request/response operations. The following example shows the page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Select a Queue for Asynchronous Operations page

Note. This page appears only when asynchronous operations are being consumed from the WSDL document.

Use the Select a Queue for Asynchronous Operations page to select a service operation queue for an asynchronous service operation.

PeopleSoft delivers a queue, *WSDL_QUEUE,* to which it assigns asynchronous service operations by default. However, you can select a different queue to use.

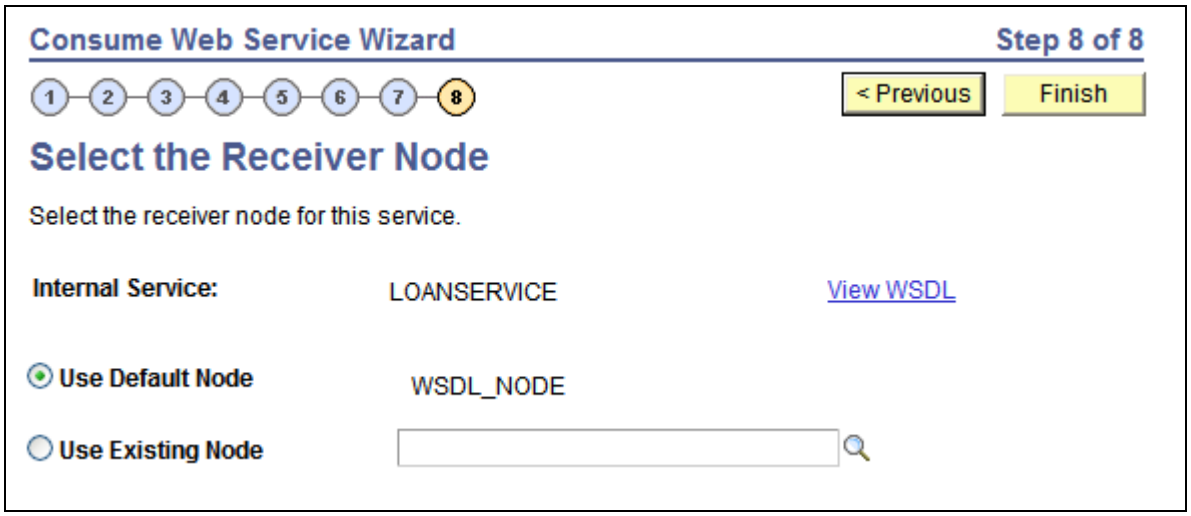
To select a queue for asynchronous operations:

1. Click the Use Existing Queue radio button.
2. From the Use Existing Queue drop-down list box, select the queue to use for the service operation.
3. Click the Next button to proceed to the next step in the wizard.

Step 8: Select the Receiver Node

Use the Select the Receiver Node page (IB_WSDL_IMP_NODE) to select the receiving node for the service. The following example shows the page:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Select the Receiver Node page

PeopleSoft delivers a node, *WSDL_NODE*, that the system uses as the receiving node by default. However, you can select a different receiving node.

If you use the Default node as the receiver node, the system adds connector property overrides to the default node in the generated service operation routing.

Note. You can apply WS-Security to services you consume using the Consume Web Service wizard. The node you select in this step determines how you implement WS-Security for these services.

See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Setting Up Secure Integration Environments," Implementing WS-Security on Services Consumed Using the Consume Web Service Wizard.

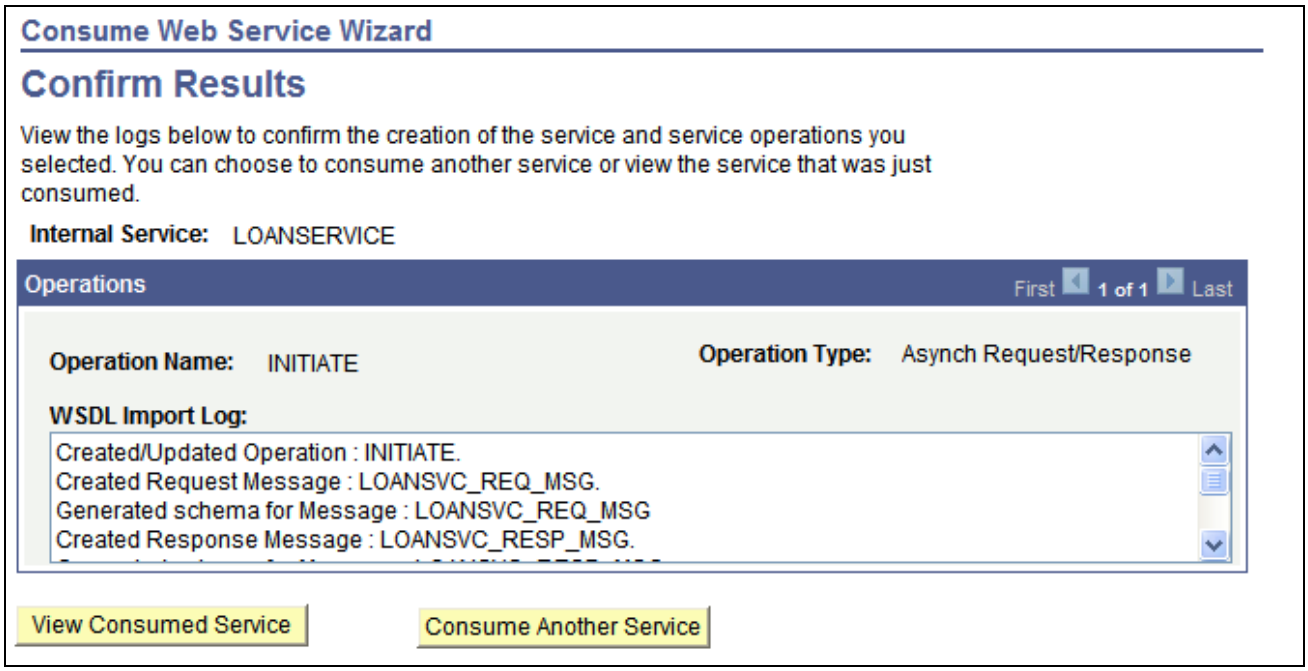
To select a receiving node for a service operation:

1. Click the Use Existing Node radio button.
2. From the Use Existing Node drop-down list box, select the receiving node to use for the service operation.
3. Click the Finish button to proceed to the next step in the wizard.

Confirm and View Results

The final page in the Consume Web Service wizard is the Confirm Results page (IB_WSDL_IMP_RSLTS) shown in the following example:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Confirm Results page

The Confirm Results page provides a WSDL Import Log. The WSDL Import log provides a summary of the WSDL import and lists the integration metadata created. The following example shows the contents of the WSDL Import Log for the example shown in this chapter of consuming a service:

```
Created/Updated Operation : INITIATE.
Created Request Message : LOANSVC_REQ_MSG.
Generated schema for Message : LOANSVC_REQ_MSG
Created Response Message : LOANSVC_RESP_MSG.
Generated schema for Message : LOANSVC_RESP_MSG
Created Fault Message : LOANSVC_FAULT_MSG.
Failed to generate schema for Message : LOANSVC_FAULT_MSG
Created Routing: ~IMPORTED~18003 for Operation: INITIATE
Created Operation Version: V1
```

The Confirm Results page also features the following page elements:

View Consumed Service Click the button to open the consumed service in the Services component.

See Chapter 9, "Managing Services," Accessing and Viewing Service Definitions, page 225.

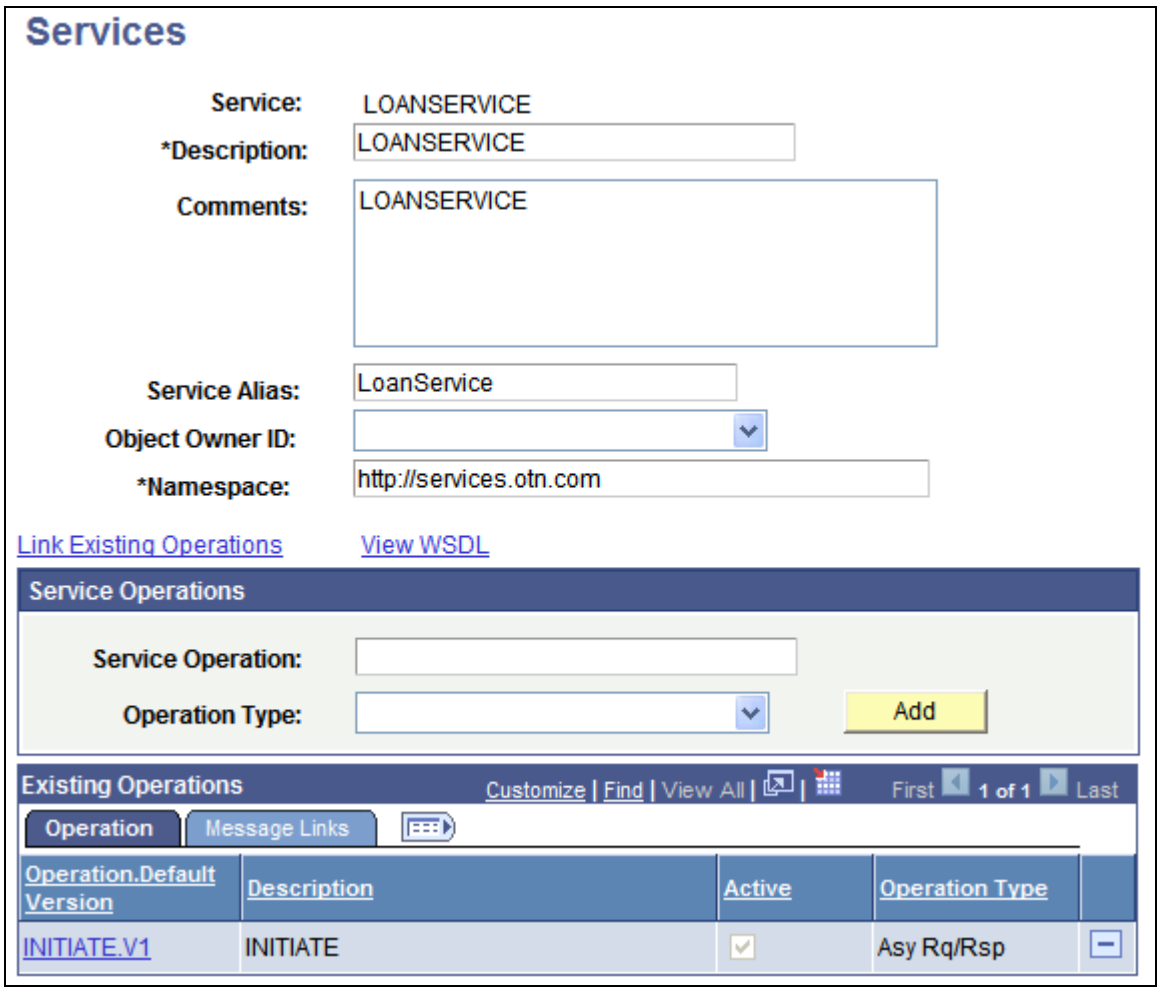
Consume Another Service Click the button to go back to step 1 of the Consume Web Service wizard and consume another service.

Accessing Integration Metadata for Consumed Services

After using the Consume Web Service wizard to consume a service into the PeopleSoft system, use the Service component to access, view and modify the integration metadata created.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following example shows the service definition for the *LOANSERVICE* service created with the Consume Web Service wizard.



Service definition for the service LOANSERVICE

The example shows that when consuming a service, the PeopleSoft system creates active versioned service operations for all operations of the service. In addition, the system saves the consumed WSDL documents for the service operations and you can view the WSDL documents by clicking the View WSDL link.

In the Existing Operations section, click the name of the service operation to open the Service Operations component. Use the Service Operations component to view and modify service operation data and message data, add handlers, and view and modify routing definitions created by the system.

Use one of the following methods to access the Services page and access and view integration metadata for a consumed service:

- On the Consume Web Service Wizard Results page, click the View Consumed Service link.
- Select PeopleTools, Integration Broker, Integration Setup, Services and select the name of the consumed service for which to view data.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 9, "Managing Services," Accessing and Viewing Service Definitions, page 225

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 22

Integrating with BPEL Process-Based Services

This chapter discusses how to:

- Configure the PeopleSoft-delivered BPEL node.
- Consume BPEL process services.
- Provide PeopleSoft services to BPEL processes.

Understanding Integrating with BPEL Processes

This chapter provides an overview of integrating with Business Process Execution Language (BPEL) processes, lists prerequisites for integrating with BPEL processes, and discusses consuming and providing BPEL processes. This chapter also features code examples to help illustrate consuming and providing BPEL processes.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Oracle BPEL Process Manager

You can use any BPEL runtime engine for integrations with BPEL processes. The developer version of Oracle BPEL Process Manager is the BPEL engine that is referenced in this chapter.

See Also

http://www.oracle.com/technology/index.html

Oracle JDeveloper Installation Guide

PeopleTools Installation Guide for your database

Oracle BPEL Process Manager Quick Start Guide

Oracle BPEL Process Manager Developer's Guide

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleSoft-Delivered Application Classes for BPEL Integrations

PeopleSoft provides several application classes for launching and controlling BPEL process instances. The following application classes are located in the PT_BPEL application package and are accessible in PeopleSoft Application Designer:

These classes are for use only when you use Oracle BPEL Process Manager as the BPEL runtime engine.

- AsyncFFSend

  This class provides the onSend handler implementation when calling a BPEL web service in an asynchronous fire and forget fashion.

- BPELUtil

  This class provides utility methods for interacting with BPEL processes from PeopleSoft systems.

- IBUtil

  This class provides utility methods to access PeopleSoft Integration Broker metadata.

See Also

PeopleTools 8.52: PeopleCode API Reference, "BPEL Classes"

Monitoring BPEL Process-Based Integrations

PeopleTools provides the following tools and logs for monitoring, tracing, and debugging integrations on PeopleSoft systems:

- Service Operations Monitor.
- Integration gateway logs.
- PeopleSoft application server logs.

In addition, your BPEL runtime engine may provide additional tools for monitoring integrations. For example, Oracle BPEL Process Manager provides the Oracle BPEL Process Manager Console for managing, administering, and debugging processes that are deployed on the BPEL server. In addition, the BPEL server logs may also provide additional details. Check your BPEL runtime engine documentation for additional information about monitoring tools that are provided.

See Also

Chapter 19, "Managing Error Handling, Logging, Tracing, and Debugging," page 467

PeopleTools 8.52: Integration Broker Service Operations Monitor PeopleBook

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Securing BPEL Process-Based Integrations

PeopleSoft Integration Broker provides a number of options that you can use to secure integrations. These include securing integrations at one or more of the following levels:

- Web server.
- Gateway level.
- Application server level.
- Node level.
- User level.
- Service operation level.
- And so on.

It is important to fully investigate these security options and implement those that best suit your business needs.

It is also important that you investigate the security options available for the BPEL runtime engine that you are using and to implement those security options that fulfill your security requirements.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Setting Up Secure Integration Environments"

Security Documentation for your BPEL runtime engine

Prerequisites for Integrating with BPEL Processes

For creating integrations with BPEL processes, you must have PeopleSoft Integration Broker configured and running.

Note. This section discusses prerequisite configuration steps on the PeopleSoft system for creating integrations with BPEL processes. Check your BPEL runtime engine software documentation for setup and configuration steps that you must perform on the runtime engine prior to performing integrations.

The following list is a partial checklist of items to configure:

- When configuring the integrationGateway.properties file, be sure to set the ig.isc.serverURL property equal to the name of the machine running the integration engine.
- When configuring the PeopleTools application server, set the PUB/SUB option to *Yes.* This value is required for asynchronous integrations.
- Activate the application server domain by setting the Domain Status to *Active.*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Set the service namespace, the schema namespace, and the target location.
- To load files into PeopleTools from the file system, set PS_FILEDIR and PS_SERVDIR in the system variables on your machine.
- PeopleSoft delivers a node, *BPEL*, specifically for BPEL integrations when you are using Oracle BPEL Process Manager as the runtime engine. If you are using Oracle BPEL Process Manager, you must configure this node. Steps to configure the *BPEL* node are provided elsewhere in this chapter.

The *Understanding Creating and Implementing Integrations* chapter of this PeopleBook contains additional information about creating and implementing integrations on PeopleSoft systems.

See Also

Chapter 4, "Understanding PeopleSoft Integration Broker Metadata," page 35

Appendix D, "Setting PS_FILEDIR and PS_SERVDIR Environment Variables," page 655

Configuring the PeopleSoft-Delivered BPEL Node

PeopleTools delivers a node called *BPEL* for use for BPEL integrations when you are using Oracle BPEL Process Manager as the runtime engine.

Note. You must configure the BPEL node only if you are using Oracle BPEL Process Manager as the BPEL runtime engine.

To configure BPEL node properties:

1. Select PeopleTools, Integration Broker, Integration Setup, Nodes.
2. Search for and open the node *BPEL*.

   The Node Definitions page appears.
3. Modify any other fields on the Node Definitions page as required.

   See *PeopleTools 8.52: PeopleSoft Integration Broker Administration*, "Adding and Configuring Nodes," Defining Node Parameters.
4. Click the Properties link at the bottom of the page.

   The Node Properties page appears.

   Note. To view the complete BPEL property name in any of the Property Name fields, insert your cursor in a field and use your keyboard arrow keys to scroll through and view the complete field name.
5. Set the value of the BPELCONSOLEURL property to the URL of the BPEL console.
6. Set the BPELDOMAIN property to the name of the domain that you are using for the BPEL server.
7. Click the OK button.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Do not set any values for the BPELDOMAINPWD property. This property is reserved for future use.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Consuming BPEL Process–Based Services

This section provides an overview of consuming BPEL process-based services and discusses how to:

- Deploy BPEL processes.
- Import WSDL documents from BPEL processes.
- Consume synchronous BPEL operations.
- Consume asynchronous request/response BPEL operations.
- Consume asynchronous one-way BPEL operations.

In addition, the end of this section provides development considerations for consuming BPEL web services.

Understanding Consuming BPEL Process-Based Services

This section provides overview information about consuming BPEL process-based services.

Development Considerations

When consuming BPEL process-based services, consider that:

- XML content representing the payload must be constructed carefully and exactly.
- The XML namespace in the top-level element of the XML fragment representing the payload is obtained from the WSDL of the BPEL service operation.
- The LaunchSyncBPELProcess and LaunchASyncBPELProcess methods do not specify the actual endpoints to which a message is sent. This endpoint is inferred by PeopleSoft Integration Broker at runtime based on the active routing that is associated with the service operation. An exception is raised if no routing exists or if more than one active routing exists.
- When using the LaunchSyncBPELProcess and LaunchASyncBPELProcess methods, you must configure the routings for the message so that exactly one active routing exists for a message. Failure to supply exactly one routing results in a runtime exception.
- PeopleSoft Integration Broker automatically maps an external service operation to an internal PeopleSoft operation. If you attempt to import an external operation that contains the same name as an internal operation that already exists, PeopleSoft Integration Broker provides the new operation with a unique name and new metadata that maps the internal name to the external name. Make sure to use the unique internal name when calling any of the application class methods in the PT_BPEL application package.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- For the asynchronous request/response operations, you must select the correct callback (response) operation for a given service request. This is achieved during the Convert step in the Consume Web Service wizard.

Deploying BPEL Processes and Importing WSDL Documents

The first two steps for consuming any type of BPEL service are to deploy the BPEL process and import the generated WSDL document into the PeopleSoft system.

The following two sections discuss performing these tasks and must be completed before consuming BPEL operations.

Deploying BPEL Processes

The first step in consuming a BPEL process-based service is to deploy the BPEL process in the BPEL runtime engine, thereby generating a WSDL document. See the documentation for your BPEL runtime engine for information about performing this task.

The PeopleSoft system can consume WSDL documents from a UDDI repository, WSDL URL, WSIL URL, or file.

When deploying a BPEL process, note the URL of the WSDL document. You must provide the document location when consuming the WSDL into the PeopleSoft system.

Consuming WSDL Documents from BPEL Processes

To consume WSDL from a BPEL process into the PeopleSoft system use the Consume Web Service component in the PeopleSoft Pure Internet Architecture.

During the consume WSDL process you are prompted to choose a receiving node. If Oracle BPEL Process Manager is your BPEL runtime engine, it is preferred to select the PeopleTools-delivered *BPEL* node as the receiving node. The *BPEL* node is delivered with BPEL-server related information, such as the BPEL console URL and BPEL domain, already added. However, during consumption you can use the default *WSDL_NODE* or any other external node.

Make a note of the web service operation for the process that you are importing as you will call this operation in PeopleCode.

See Also

Chapter 21, "Consuming Services," page 515

Consuming Synchronous BPEL Operations

This section provides an overview of consuming synchronous BPEL operations and discusses how to:

- Create synchronous BPEL requests.
- Invoke synchronous BPEL operations.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Process synchronous BPEL responses.

Understanding Consuming Synchronous BPEL Operations

Before consuming synchronous BPEL operations, you must deploy a BPEL process and import the generated WSDL document into the PeopleSoft system.

See Chapter 22, "Integrating with BPEL Process-Based Services," Deploying BPEL Processes, page 540 and Chapter 22, "Integrating with BPEL Process-Based Services," Consuming WSDL Documents from BPEL Processes, page 540.

Creating Synchronous BPEL Requests

In this step, you use the PeopleSoft-delivered BPEL application classes to create a request message and initialize it with the message content, or payload. The payload is the content that will be sent as part of the soap body to invoke the BPEL process.

The first step is to create a BPELUtil object.

```
&bpel = create PT_BPEL:BPELUtil();
```

Next, create the request message and specify the operation to invoke. In the following code example, *PROCESS* is the operation to be invoked.

```
&payload = "<?xml version='1.0'?>
 <SyncCalcProcessRequest xmlns='http://xmlns.oracle.com/SyncCalc'>
 <messageid>TestId::0123456789</messageid><op>+</op><input1>9</input1>
 <input2>3</input2></SyncCalcProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.PROCESS, %IntBroker_Request);
&msg.SetXmlDoc(&xml);
```

Invoking Synchronous BPEL Operations

To invoke a synchronous BPEL operation, use the LaunchSyncBPELProcess method of the BPELUtil application class.

The following code example shows how to invoke a synchronous operation.

```
&reply = &bpel.LaunchSyncBPELProcess(&OP_NAME="PROCESS", &msg, "", "");
```

Processing Synchronous BPEL Responses

The following PeopleCode example shows sample shows code for processing a synchronous BPEL response.

```
If All(&reply) Then
   &responseStr = &reply.GenXMLString();
   End-If;
```

Example: Consuming Synchronous BPEL Operations

The following code example demonstrates how to use all of the PeopleCode discussed earlier in this section.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PT_BPEL:*;

Local PT_BPEL:BPELUtil &bpel;
Local string &url,  &transactionId,
&payload, &responseStr, &OP_NAME;
Local Message &msg, &reply;
Local XmlDoc &xml;

/* --- creating a BPELUtil object---*/
&bpel = create PT_BPEL:BPELUtil();

/* --- setting operation name --- */
&OP_NAME="PROCESS";
&transactionId = "TestId::0123456789";

&payload = "<?xml version='1.0'?><SyncCalcProcessRequest xmlns=
'http://xmlns.oracle.com/SyncCalc'><messageid>TestId::0123456789</messageid>
<op>+</op><input1>9</input1><input2>3</input2></SyncCalcProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.PROCESS, %IntBroker_Request);
&msg.SetXmlDoc(&xml);

&reply = &bpel.LaunchSyncBPELProcess(&OP_NAME, &msg, "", "");
If All(&reply) Then
   &responseStr = &reply.GenXMLString();
   End-If;
&url = &bpel.GetSyncBPELProcessInstanceUrl("BPEL", &transactionId);
```

See Also

PeopleTools 8.52: PeopleCode API Reference, "BPEL Classes"

Consuming Asynchronous Request/Response BPEL Operations

This section provides an overview of consuming asynchronous request/response BPEL operations and discusses how to:

- Create an asynchronous request/response BPEL request.
- Invoke an asynchronous request/response BPEL operation.
- Create a handler to process the asynchronous request/response BPEL response.
- Add a handler to a service operation to process the asynchronous request/response BPEL response.

This section also features a comprehensive example of the PeopleCode discussed in this section.

Understanding Consuming Asynchronous Request/Response BPEL Operations

Before consuming asynchronous request/response BPEL operations, you must deploy a BPEL process and import the generated WSDL document into the PeopleSoft system.

When you consume an Asynchronous Request/Response WSDL, you must identify the request and callback operations correctly in the Consume Web Service wizard.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 22, "Integrating with BPEL Process-Based Services," Deploying BPEL Processes, page 540 and Chapter 22, "Integrating with BPEL Process-Based Services," Consuming WSDL Documents from BPEL Processes, page 540.

Understanding Updating Asynchronous Request/Response WSDL for BPEL Callback Operations

If you are using Oracle BPEL Process Manager to generate WSDL for a PeopleSoft system to consume, you must modify the generated WSDL before attempting to consume it in the PeopleSoft system.

Note. This section applies only if you are using Oracle BPEL Process Manager 11g to generate WSDL for consuming in the PeopleSoft system.

If you are using earlier versions of Oracle BPEL Process Manager or another BPEL runtime engine, you may disregard the information presented in this section.

The WSDL created by Oracle BPEL Process Manager for asynchronous request/response integrations does not contain a service definition for the callback operation. The PeopleSoft system does not recognize the callback until the service definition is defined. You must add the following code to the WSDL document prior to consuming it in the PeopleSoft system.

```
<wsdl:service name="asyncrrcalcbpelcallback_client_ep">
   <wsdl:port name="AsyncRRCalcBPELCallback_pt" binding="client:
    AsyncRRCalcBPELCallbackBinding">
      <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       location="http://Client.EndPoint.Set.By.Caller"/>
   </wsdl:port>
</wsdl:service>
```

Creating Asynchronous Request/Response BPEL Requests

In this step, you use the PeopleSoft-delivered BPEL application classes to create a request message and initialize it with the message content, or payload. The payload is the SOAP request (envelope) that will be sent to the BPEL process as a service.

```
&bpel = create PT_BPEL:BPELUtil();
```

Next, create the request message and specify the operation to invoke. In the following code example, *CALCULATE* is the operation to be invoked:

```
&payload = "<?xml version='1.0'?>
<ASyncCalcProcessRequest  xmlns='http://xmlns.oracle.com/ASyncCalc'>
<op>+</op><input1>9</input1><input2>3</input2></ASyncCalcProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.CALCULATE, %IntBroker_Request);
&msg.SetXmlDoc(&xml);
```

Invoking Asynchronous Request/Response BPEL Operations

To invoke an asynchronous BPEL operation, use the LaunchASyncBPELProcess method of the BPELUtil application class.

The following example shows code for invoking an asynchronous operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
&responseStr = &bpel.LaunchASyncBPELProcess(&OP_NAME="CALCULATE", &msg, "", "");
```

Creating Handlers to Process Asynchronous Request/Response BPEL Responses

When you imported the asynchronous request/response BPEL WSDL document into the PeopleSoft system, the system automatically created a request message and response message for each service operation that is contained in the WSDL document.

To process the response, you must create a handler and add it to the service operation definition.

To create a handler for the response to an asynchronous request/response operation, use PeopleSoft Application Designer to extend the PS_PT:Integration:INotificationHandler application class and use the OnResponse method to code the response. When PeopleSoft Integration Broker receives the response for the service operation, Integration Broker will use the handler code to process the response.

The following example shows a sample application class that is designed to handle the response:

```
import PS_PT:Integration:INotificationHandler;

class ASyncTestHandler implements PS_PT:Integration:INotificationHandler
   method ASyncTestHandler();
   method OnNotify(&MSG As Message);
end-class;

/* constructor */
method ASyncTestHandler
end-method;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/

   Local File &MYFILE;
   &MYFILE = GetFile("C:\temp\item.txt", "W", %FilePath_Absolute);

   If &MYFILE.IsOpen Then
      &MYFILE.WriteLine("--- Response Received ---");
      &MYFILE.WriteLine(&MSG.GenXMLString());
      &MYFILE.Close();
   End-If;

end-method;
```

Adding Handlers to Operations to Process Asynchronous Request/Response BPEL Responses

After you create the response handler, you must add it to the service operation definition. To perform this task, use the Service Operations–Handlers page. To access this page, select PeopleTools, Integration Broker, Integration Setup, Service Operations, and click the Handlers tab.

Example: Consuming Asynchronous Request/Response BPEL Operations

The following code example provides a full example of all the PeopleCode discussed earlier for creating the asynchronous request and invoking the service operation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PT_BPEL:*;

Local PT_BPEL:IBUtil &ibutil;
Local PT_BPEL:BPELUtil &bpel;
Local string &url, &domain, &pwd, &opType, &asyncUrl, &transactionId,
&payload, &responseStr, &OP_NAME;
Local number &routings;
Local Message &msg, &reply;
Local XmlDoc &xml;

/* --- creating a BPELUtil object --- */
&bpel = create PT_BPEL:BPELUtil();

/* --- setting operation name --- */
&OP_NAME = "CALCULATE";

&payload = "<?xml version='1.0'?><ASyncCalcProcessRequest xmlns=
'http://xmlns.oracle.com/ASyncCalc'><op>+</op><input1>9</input1>
<input2>3</input2></ASyncCalcProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.CALCULATE, %IntBroker_Request);
&msg.SetXmlDoc(&xml);

&responseStr = &bpel.LaunchASyncBPELProcess(&OP_NAME, &msg, "", "");
/*The Instance URL would be available only after the instance is created on the⇒
 BPEL engine. Programmatically wait for URL if needed*/
&url = &bpel.GetASyncBPELProcessInstanceUrl(&responseStr);
```

See Also

Chapter 16, "Managing Service Operation Handlers," page 347

PeopleTools 8.52: PeopleCode API Reference, "BPEL Classes"

Consuming Asynchronous Fire-and-Forget (One-Way) BPEL Operations

This section provides an overview of consuming asynchronous fire-and-forget BPEL operations and discusses how to:

- Create an asynchronous fire-and-forget BPEL requests.
- Invoke an asynchronous fire-and-forget BPEL operation.
- Add handlers to assign correlation IDs to requests.

Note. BPEL asynchronous fire-and-forget operations correspond to asynchronous one-way operations in PeopleSoft systems.

This section also features a comprehensive example of the PeopleCode discussed in this section.

Understanding Consuming Asynchronous Fire-and-Forget BPEL Operations

Before consuming asynchronous fire-and-forget BPEL operations, you must deploy a BPEL process and import the generated WSDL document into the PeopleSoft system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 22, "Integrating with BPEL Process-Based Services," Deploying BPEL Processes, page 540 and Chapter 22, "Integrating with BPEL Process-Based Services," Consuming WSDL Documents from BPEL Processes, page 540.

Creating Asynchronous Fire–and–Forget BPEL Requests

In this step, you use the PeopleSoft-delivered BPEL application classes to create a request message and initialize it with the message content, or payload. The payload is the SOAP request (envelope) that will be sent to the BPEL process as a service.

The first step is to create a BPELUtil object.

```
&bpel = create PT_BPEL:BPELUtil();
```

Next, create the request message and specify the operation to invoke. In the following code example, *FIREANDFORGET* is the operation to be invoked:

```
&payload = "<?xml version='1.0'?>
<ASyncFFProcessRequest xmlns='http://xmlns.oracle.com/ASyncFF'>
<input>test</input></ASyncFFProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.FIREANDFORGET, %IntBroker_Request);
&msg.SetXmlDoc(&xml);
```

Invoking Asynchronous Fire and Forget BPEL Operations

To invoke an asynchronous BPEL operation, use the LaunchASyncBPELProcess method of the BPELUtil application class.

The following code example shows how to invoke an asynchronous operation:

```
&responseStr = &bpel.LaunchASyncBPELProcess(&OP_NAME, &msg, "", "");
```

Adding Handlers to Assign Correlation IDs to Requests

You can create a handler to add a WSA_MessageID to the SOAP request header before the request is dispatched for the BPEL process.

To create a such a handler use PeopleSoft Application Designer to extend the PS_PT:Integration:ISend application class and use the AsyncFFSend method.

The following code example shows an application class to perform this task:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:ISend;
import PT_BPEL:IBUtil;

class AsyncFFSend implements PS_PT:Integration:ISend
   method AsyncFFSend();
   method OnRequestSend(&MSG As Message) Returns Message;
end-class;

/* constructor */
method AsyncFFSend
end-method;

method OnRequestSend
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:ISend.OnRequestSend +/

   &MSG.IBInfo.WSA_MessageID = &MSG.TransactionId;
   Return &MSG;
end-method;
```

Example: Consuming Asynchronous Fire-and-Forget BPEL Processes

The following code example provides a full example of all the PeopleCode discussed earlier for creating the asynchronous request and invoking the service operation.

```
import PT_BPEL:*;

Local PT_BPEL:BPELUtil &bpel;
Local string &url, &payload, &responseStr, &OP_NAME;
Local Message &msg, &reply;
Local XmlDoc &xml;

/* --- creating a BPELUtil object --- */
&bpel = create PT_BPEL:BPELUtil();

/* --- setting operation name --- */
&OP_NAME = "FIREANDFORGET";

&payload = "<?xml version='1.0'?><ASyncFFProcessRequest xmlns='http:
//xmlns.oracle.com/ASyncFF'><input>test</input></ASyncFFProcessRequest>";

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.FIREANDFORGET, %IntBroker_Request);
&msg.SetXmlDoc(&xml);

&responseStr = &bpel.LaunchASyncBPELProcess(&OP_NAME, &msg, "", "");

Example: Consuming Asynchronous Fire-and-Forget BPEL Processes
/*The Instance URL would be available only after the instance is      */
/*created on the BPEL engine. Programmatically wait for URL if needed */

&url = &bpel.GetASyncBPELProcessInstanceUrl(&responseStr);
```

See Also

Chapter 16, "Managing Service Operation Handlers," page 347

PeopleTools 8.52: PeopleCode API Reference, "BPEL Classes"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Providing PeopleSoft Services to BPEL Processes

This section provides an overview of providing services to BPEL processes and discusses how to:

- Provide PeopleSoft synchronous operations to BPEL processes.
- Provide PeopleSoft asynchronous request/response operations to BPEL processes.

Understanding Providing PeopleSoft Services to BPEL Processes

This section provides overview information for providing services to BPEL processes.

Container Messages and Message Parts

If using container messages and message parts, create rowset-based parts in nonrowset-based containers. Doing so ensures that the XSchema that is generated for messages will not include PeopleTools-specific auditing information (PSCAMA). Typically, you do not want this auditing information to be included in the generated XSchema when integrating with BPEL processes.

Provide Web Service Wizard

When providing services to systems using BPEL, consider the following guidelines:

- The providing web services step using the Provide Web Service wizard described in this section is for synchronous and asynchronous request/response operations and is required only for an external client to be able to consume a PeopleSoft service.
- When providing the service, the WSDL document is exported to the WSDL repository in the PeopleSoft Pure Internet Architecture. Optionally, you can select to export the WSDL to a UDDI repository. PeopleSoft uses relative path URL for schemas referenced in the WSDL documents the system generates.
- After generating the WSDL document, carefully inspect the results using the WSDL Generation Log (the last page of the Provide Web Services wizard).

  The WSDL Generation Log contains the *Generated WSDL URL*. Copy this URL and store it somewhere carefully. You will need this WSDL URL later when calling the PeopleSoft-provided web service operation from the BPEL process.

See Also

Chapter 6, "Managing Messages," Managing Container Messages, page 109

Chapter 20, "Providing Services," page 479

Providing Synchronous PeopleSoft Operations to BPEL Processes

This section discusses how to:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Build synchronous PeopleSoft services.
- Provide synchronous PeopleSoft services as WSDL documents.
- Consume the service in a BPEL process and invoke the BPEL process.

Building Synchronous PeopleSoft Services

This section lists the steps for building synchronous PeopleSoft services. Detailed documentation for each step is provided elsewhere in this PeopleBook. See the end of this section for links to the appropriate documentation.

1. Define request and response messages to be associated with the service operation.

   Use the appropriate XSchema for each message.

   Schema for the messages are needed for generating the WSDL. If rowset-based messages are used in the operation, simply save the message definition to generate the schema. If nonrowset messages are used in the operation, you must manually provide schema. These schema will determine the request and response format during integration.

   Examples of XSchema for request and response messages are provided at the end of this section.

2. Create a handler to process the request message.

   Extend the PS_PT:Integration:IRequestHandler application class using the OnRequest method. The output of the handler will be communicated back as the response to the received request.

   An code example of an OnRequest handler is provided at the end of this section.

3. Create a new service.
4. Add a synchronous operation to the service.
   - Generate an any-to-local routing definition for the operation.
   - Add permissions to the service operation.
   - Add the OnRequest handler that you created in step 2 to the operation by referencing the package name, path, and class ID of the handler that you created in step 2.

See Chapter 7, "Sending and Receiving Messages," page 127; Chapter 16, "Managing Service Operation Handlers," page 347; Chapter 9, "Managing Services," page 223 and Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255.

Providing Synchronous PeopleSoft Services as WSDL Documents

Use the Provide Web Services wizard to export the service and generate a WSDL document.

See Chapter 20, "Providing Services," page 479.

Consuming Services in BPEL Processes and Invoking Services from BPEL Processes

The last step is to consume the PeopleSoft-provided service in a BPEL process and then invoke the consumed service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See the documentation that is provided with your BPEL runtime engine for the steps that are necessary to accomplish this step.

Example 1: Providing Synchronous Operations – Sample Request Message

The following code shows an example of a request message:

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://xmlns.oracle.com/Enterprise/Tools
/schemas/PSFTCALCREQUESTMESSAGE.V1" xmlns="http://xmlns.oracle.com
/Enterprise/Tools/schemas/PSFTCALCREQUESTMESSAGE.V1" xmlns:xsd="http:
//www.w3.org/2001/XMLSchema">
  <xsd:element name="PSFTCalcRequestMessage" type=
    "PSFTCalcRequestMessage_Type Shape"/>
  <xsd:complexType name="PSFTCalcRequestMessage_TypeShape">
    <xsd:sequence>
      <xsd:element name="op" type="xsd:string"/>
      <xsd:element name="input1" type="xsd:decimal"/>
      <xsd:element name="input2" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example 2: Providing Synchronous Operations – Sample Response Message

The following code shows an example of a response message:

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://xmlns.oracle.com/Enterprise/Tools
/schemas/PSFTCALCRESPONSEMESSAGE.V1" xmlns="http://xmlns.oracle.com
/Enterprise/Tools/schemas/PSFTCALCRESPONSEMESSAGE.V1" xmlns:xsd="http:
//www.w3.org/2001/XMLSchema">
  <xsd:element name="PSFTCalcResponseMessage" type=
"PSFTCalcResponseMessage_TypeShape"/>
  <xsd:complexType name="PSFTCalcResponseMessage_TypeShape">
    <xsd:sequence>
      <xsd:element name="result" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example 3: Providing Synchronous Operations – OnRequest Handler

The following code shows an example of creating an OnRequest handler by extending the PS_PT:Integration:IRequestHandler application class:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IRequestHandler;

class InboundSyncRequestHandler implements PS_PT:Integration:IRequestHandler
   method InboundSyncRequestHandler();
   method onRequest(&MSG As Message) Returns Message;
   method onError(&MSG As Message) Returns string;
end-class;

method InboundSyncRequestHandler
end-method;

method onRequest
   /+ &MSG as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnRequest +/

   Local Message &response;
   Local File &MYFILE;
   Local XmlDoc &xml, &inxml;
   Local string &payload, &oper, &input1, &input2;
   Local array of XmlNode &nodes;
   Local XmlNode &node;

   &nodes = CreateArray(&node);
   &inxml = &MSG.GetXmlDoc();
   &nodes = &inxml.GetElementsByTagName("op");
   &oper = &nodes [1].NodeValue;
   &nodes = &inxml.GetElementsByTagName("input1");
   &input1 = &nodes [1].NodeValue;
   &nodes = &inxml.GetElementsByTagName("input2");
   &input2 = &nodes [1].NodeValue;

   &payload = "<?xml version='1.0'?><PSFTCalcResponseMessage xmlns=
     'http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PSFTCALCRESPONSEMESSAGE.V1'><result xmlns=">9</result>
      </PSFTCalcResponseMessage>";

   &xml = CreateXmlDoc(&payload);
   &response = CreateMessage(Operation.PSFTCALCULATE, %IntBroker_Response);
   &response.SetXmlDoc(&xml);

   Return &response;
end-method;

method onError
   /+ &MSG as Message +/
   /+ Returns String +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnError +/

   Local integer &nMsgNumber, &nMsgSetNumber;
   Local string &str;

   &nMsgNumber = &MSG.IBException.MessageNumber;
   &nMsgSetNumber = &MSG.IBException.MessageSetNumber;
   &str = &MSG.IBException.DefaultText;

   Return &str;
end-method;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Providing Asynchronous PeopleSoft Request/Response Operations to BPEL Processes

This section discusses how to:

- Build asynchronous request/response PeopleSoft services.
- Provide an asynchronous request/response PeopleSoft service as a WSDL document.
- Consume the service in a BPEL process and invoke the BPEL process.

Building Asynchronous Request/Response PeopleSoft Services

This section lists the steps for building asynchronous request/response PeopleSoft services. Detailed documentation for each step is provided elsewhere in this PeopleBook. See the end of this section for links to the appropriate documentation.

1. Define request and response messages to be associated with the service operation.

   Use the appropriate XSchema for each message.

   Examples of XSchema for request and response messages are provided at the end of this section.

2. Create a handler to process the request message.

   Extend the PS_PT:Integration:INotificationHandler application class using the OnNotify method. The output of the handler will be communicated back as the response to the received request.

   A code example of an OnNotify handler is provided at the end of this section.

3. Create a new service.

4. Add an Asynch Request/Response operation to the service.

   - Generate an any-to-local routing definition for the operation.
   - Add permissions to the service operation.
   - Add the OnNotify handler that you created in step 2 to the operation by referencing the package name, path, and class ID of the handler.

For Asynchronous Request/Response PeopleSoft services, you must invoke the UpdateConnectorResponseProperty method of the BPELUtil class in OnNotify before sending the asynchronous response to BPEL. This will override the default content-type from text/plain to text/xml as expected by BPEL. If this is not done, BPEL will return an HTTP 415 Unknown Media-Type error.

See Chapter 7, "Sending and Receiving Messages," page 127; Chapter 16, "Managing Service Operation Handlers," page 347; Chapter 9, "Managing Services," page 223 and Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255.

Providing Asynchronous PeopleSoft Services as WSDL Documents

Use the Provide Web Services wizard to export the service and generate a WSDL document.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Chapter 20, "Providing Services," page 479.

Consuming Services in BPEL Processes and Invoking Services from BPEL Processes

The last step is to consume the PeopleSoft-provided service in a BPEL process and then invoke the consumed service.

See the documentation that is provided with your BPEL runtime engine for the steps that are necessary to accomplish this step.

Example 1: Providing Asynchronous Request/Response Operations – Sample Request Message

The following code shows an example of a request message:

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://xmlns.oracle.com/Enterprise/Tools
/schemas/PSFTCALCREQUESTMESSAGE.V1" xmlns="http://xmlns.oracle.com
/Enterprise/Tools/schemas/PSFTCALCREQUESTMESSAGE.V1" xmlns:xsd="http:
//www.w3.org/2001/XMLSchema">
  <xsd:element name="PSFTCalcRequestMessage" type=
"PSFTCalcRequestMessage_TypeShape"/>
  <xsd:complexType name="PSFTCalcRequestMessage_TypeShape">
    <xsd:sequence>
      <xsd:element name="op" type="xsd:string"/>
      <xsd:element name="input1" type="xsd:decimal"/>
      <xsd:element name="input2" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example 2: Providing Asynchronous Request/Response Operations – Sample Response Message

The following code shows an example of a response message:

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://xmlns.oracle.com/Enterprise/Tools
/schemas/PSFTCALCRESPONSEMESSAGE.V1" xmlns="http://xmlns.oracle.com
/Enterprise/Tools/schemas/PSFTCALCRESPONSEMESSAGE.V1" xmlns:xsd="http:
//www.w3.org/2001/XMLSchema">
  <xsd:element name="PSFTCalcResponseMessage" type=
"PSFTCalcResponseMessage_TypeShape"/>
  <xsd:complexType name="PSFTCalcResponseMessage_TypeShape">
    <xsd:sequence>
      <xsd:element name="result" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Example 3: Providing Asynchronous Request/Response Operations – Sample OnNotify Handler

The following code shows an example of creating an OnNotify handler by extending the PS_PT:Integration:INotification application class. Note that you must invoke the UpdateConnectorResponseProperty method of the BPELUtil class before performing the publish.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:INotificationHandler;

class InboundASyncResponseHandler implements PS_PT:Integration:
INotificationHandler
   method OnNotify(&MSG As Message);

end-class;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/

   Local Message &response;
   Local string &payload;
   Local XmlDoc &xml;
   Local File &MYFILE;

&payload = "<?xml version='1.0'?><PSFTCalcResponseMessage xmlns=
'http://xmlns.oracle.com/Enterprise/Tools/schemas
/PSFTCALCRESPONSEMESSAGE.V1'><result xmlns=">9</result></PSFTCalcResponse⇒
Message>";

   &xml = CreateXmlDoc(&payload);
   &response = CreateMessage(Operation.PSFTASYNCCALCULATE,
%IntBroker_Response);
   &response.SetXmlDoc(&xml);
   &response.IBInfo.WSA_MessageID = &MSG.IBInfo.WSA_MessageID;
   &response.IBInfo.WSA_ReplyTo = &MSG.IBInfo.WSA_ReplyTo;
   &bpel.UpdateConnectorResponseProperties(&response);
   %IntBroker.Publish(&response);
end-method;
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 23

Integrating with Oracle Mediator and Oracle ESB-Based Services

This chapter discusses how to:

- Consume and invoke Oracle Mediator/ESB-based services.
- Provide and invoke PeopleSoft services in Oracle Mediator/ESB.

Understanding Integrating with Oracle Mediator and Oracle ESB-Based Services

This chapter discusses how to consume and provide Oracle Mediator and Oracle ESB-based services. This chapter also features code examples to help illustrate consuming and providing Oracle Mediator and Oracle ESB-based services.

Note. The code examples in this chapter are for illustrative purposes only and are not intended to be used in a production environment.

Oracle Mediator and Oracle ESB

Oracle Mediator is a component of Oracle Fusion Middleware 11g. Oracle Mediator separates integration concerns from applications and business logic. It moves data among disparate applications, both within and outside of an enterprise. It provides a messaging infrastructure and uses open standards to connect, transform, and route business documents. Oracle Mediator cannot provide services. Instead, it virtualizes endpoints and mediates between a client and a provider of a service.

In Oracle releases earlier than 11g, Oracle Mediator was named Oracle Enterprise Service Bus (ESB)

PeopleSoft supports integrations Oracle Mediator and Oracle ESB-based services. From a PeopleSoft perspective, the main differences between Oracle Mediator and Oracle ESB are:

- Oracle Mediator supports asynchronous request/response integrations. You cannot perform asynchronous request/response integrations using Oracle ESB.
- PeopleSoft supports integrations with Oracle ESB nested WSDL documents. Note that Oracle Mediator does not support nested WSDL documents.

For more information about Oracle ESB or Oracle Mediator, visit the Oracle Technology Network online and enter the keywords *Oracle ESB* or *Oracle Mediator*.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

http://www.oracle.com/technology/products/index.html

Software Components

You must install the following software components to perform integrations using Oracle ESB or Oracle Mediator:

Software Component	*Description*
PeopleTools	Installs the PeopleSoft application server and the web. Also installs PeopleSoft Integration Broker and the PeopleSoft SOA, including the integration gateway, publication/subscription system, Provide Services Wizard, Consume Services Wizard, and so on.
Oracle Fusion Middleware	Required for integrations with Oracle Mediator 11g.
Oracle SOA Suite	Installs the Oracle services-oriented architecture, including Oracle Enterprise Manager. Required only for Oracle 10.1.3g ESB-based integrations.
Oracle JDeveloper Studio	Oracle JDeveloper Studio 11g features the Oracle BPEL Designer and the Oracle Mediator used to develop BPEL and Mediator projects. Oracle JDeveloper Studio 10.1.3g and earlier releases features the Oracle BPEL Designer and the ESB Designer used to develop BPEL and ESB projects.

Check the My Oracle Support website for the currently certified versions of Oracle SOA and Oracle JDeveloper Studio for use with PeopleTools.

Check PeopleTools and Oracle documentation for installation tips and useful information.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools Installation Guide for your database

Oracle SOA Suite Quick Start Guide

Oracle SOA Suite Installation Guide

Oracle JDeveloper Installation Guide

Oracle BPEL Process Manager Quick Start Guide

Oracle Mediator Quick Start Guide

Oracle Mediator Installation Guide

Oracle Enterprise Service Bus Quick Start Guide

Oracle Enterprise Service Bus Installation Guide

http://www.oracle.com/technology/products/index.html

Securing Oracle Mediator and Oracle ESB-Based Services

PeopleSoft Integration Broker provides a number of options that you can use to secure integrations. These include securing integrations at one or more of the following levels:

- Web server.
- Gateway level.
- Application server level.
- Node level.
- User level.
- Service operation level.
- And so on.

It is important to fully investigate these security options and implement those that best suit your business needs.

It is also important that you investigate the security options available for the Oracle SOA Suite and Oracle JDeveloper and to implement those security options that fulfill your security requirements.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Setting Up Secure Integration Environments"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Prerequisites for Integrating with Oracle Mediator/ESB–Based Services

For creating integrations with Oracle Mediator and Oracle ESB services you must have PeopleSoft Integration Broker configured and running.

Note. This section discusses prerequisite configuration steps on the PeopleSoft system for integrating with Oracle Mediator and Oracle ESB-based services. Check your Oracle documentation for setup and configuration steps that you must perform on Oracle JDeveloper, Oracle BPEL Process Manager, and Oracle Mediator or Oracle ESB.

The following list is a partial checklist of items to configure on the PeopleSoft system:

- In PeopleTools ensure that the application server and web server are installed and running.
- On the Nodes-Node Definitions page, create a node to represent Oracle ESB. Set the Node Type field to *External* for this node.
- When configuring the integrationGateway.properties file, be sure to set the ig.isc.serverURL property equal to the name of the machine running the integration engine.
- When configuring the PeopleTools application server, set the PUB/SUB option to *Yes.* This value is required for asynchronous integrations.
- Activate the application server domain by setting the Domain Status to *Active.*
- Set the service namespace, the schema namespace, and the target location.
- To load files into PeopleTools from the file system, set PS_FILEDIR and PS_SERVDIR in the system variables on your machine.

See Also

Chapter 4, "Understanding PeopleSoft Integration Broker Metadata," page 35

Appendix D, "Setting PS_FILEDIR and PS_SERVDIR Environment Variables," page 655

Consuming and Invoking Oracle Mediator/ESB-Based Services

This section discusses how to:

- Provide Oracle Mediator/ESB-based services for consuming in PeopleSoft.
- Consume Oracle Mediator/ESB-based services.
- Invoke synchronous Oracle Mediator/ESB-based services.
- Invoke asynchronous Oracle Mediator/ESB-based services.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Invoke asynchronous request/request Oracle Mediator services.

Understanding Consuming and Invoking Oracle Mediator/ESB-Based Services

This section provides information about consuming Oracle Mediator/ESB-Based services.

Oracle Mediator/ESB Projects When PeopleSoft is a Consumer

Oracle Mediator/ESB cannot provide a service. It can only mediate between a client and the provider of the service.

Hence, if PeopleSoft is the consumer of a Mediator or ESB service, the Mediator/ESB project will contain a SOAP service (like BPEL) that will implement the business logic. Further there will be a routing service that will route the SOAP request message from PeopleSoft to the underlying web service implementation.

Service Routing Types

There are three types of routing services: synchronous, asynchronous, and asynchronous request/response.

Note. Oracle Mediator provides the ability to consume asynchronous request/response services. You cannot consume these types of services uses Oracle ESB.

The type of routing service determines the type of Mediator or ESB project. A Mediator or ESB project with a synchronous routing service defined is a synchronous Mediator/ESB project; and a Mediator/ESB project with an asynchronous routing service defined is an asynchronous Mediator/ESB project. Mediator projects with an asynchronous request/response service defined is an asynchronous request/response service.

Oracle ESB Nested and Non-Nested WSDL

PeopleSoft systems can consume WSDL documents that are nested or not nested. This section provides additional information on each type.

Note. PeopleSoft supports integrations with Oracle ESB nested WSDL documents. This feature is not available with Oracle Mediator.

Using nested WSDL documents allows you to separate the different elements of a service definition into independent documents that you can consume as needed.

Using nested WSDL helps you to write clearer definitions by enabling you to separate the definitions according to their level of abstraction. It also maximizes the ability to reuse service definitions.

The following example shows a nested WSDL document. The document is separated into three smaller documents: data type definitions, abstract definitions, and specific service bindings.

The following example shows a data type definition WSDL document:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
//data type definitions WSDL//
<schema attributeFormDefault="unqualified"
     elementFormDefault="qualified"
     targetNamespace="http://xmlns.oracle.com/SynchAdd_BPEL"
     xmlns="http://www.w3.org/2001/XMLSchema">
     <element name="SynchAdd_BPELProcessRequest">
          <complexType>
               <sequence>
                    <element name="input1" type="integer"/>
                    <element name="input2" type="integer"/>
               </sequence>
          </complexType>
     </element>
</schema>
```

The following example show an abstract definition WSDL document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="NestedAsyncFFAdd_RS"
     targetNamespace="http://oracle.com/esb/namespaces/NestedAsyncFFAdd_ESB"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://oracle.com/esb/namespaces/NestedAsyncFFAdd_ESB"
     xmlns:inp1="http://xmlns.oracle.com/SynchAdd_BPEL">
     <types>
          <schema xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://xmlns.oracle.com/SynchAdd_BPEL"
schemaLocation="SyncAdd_Req.xsd" />
</schema>
</types>
<message name="SynchAdd_BPELProcessRequest_request">
<part name="SynchAdd_BPELProcessRequest"
element="inp1:SynchAdd_BPELProcessRequest" />
</message>
<portType name="NestedESBAsyncFFAdd_ppt">
<operation name="NestedESBAsyncFFAdd">
<input message="tns:SynchAdd_BPELProcessRequest_request" />
</operation>
</portType>
</definitions>
```

The following example shows a service bindings WSDL document:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace="http://oracle.com/esb/namespaces/
  NestedAsyncFFAdd_ESB/concrete"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://oracle.com/esb/namespaces/NestedAsyncFFAdd_ESB/concrete"
xmlns:ws="http://www.example.com/webservice" xmlns:http="http:
//schemas.xmlsoap.org/wsdl/http/"
xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:esb="http://www.oracle.com/esb/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl="http:
//schemas.xmlsoap.org/wsdl/"
xmlns:import="http://oracle.com/esb/namespaces/NestedAsyncFFAdd_ESB">
<import namespace="http://oracle.com/esb/namespaces/NestedAsyncFFAdd_ESB"
location="http://bng-psft-
0100:8888/esb/slide/ESB_Projects/PS_Consumer_NestedAsyncFFAdd_ESB/NestedAsyncFFAdd_
ESB_NestedAsyncFFAdd_RS.wsdl" />
<binding name="__soap_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt"
type="import:NestedESBAsyncFFAdd_ppt">
<soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="NestedESBAsyncFFAdd">
<soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" style="document"
soapAction="NestedESBAsyncFFAdd" />
<input>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" use="literal" />
</input>
</operation>
</binding>
<binding name="__esb_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt"
type="import:NestedESBAsyncFFAdd_ppt">
<esb:binding />
<operation name="NestedESBAsyncFFAdd">
<esb:operation eventname="
NestedAsyncFFAdd_ESB.NestedAsyncFFAdd_RS.NestedESBAsyncFFAdd" />
<input />
</operation>
</binding>
<service name="ESB_NestedAsyncFFAdd_RS_Service">
<port name="__soap_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt"
binding="tns:__soap_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt">
<soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 location="http://bng-psft-0100:8888/event/NestedAsyncFFAdd_ESB/
 NestedAsyncFFAdd_RS" />
</port>
<port name="__esb_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt"
binding="tns:__esb_NestedAsyncFFAdd_RS_NestedESBAsyncFFAdd_ppt"/>
</service>
<plt:partnerLinkType name="NestedESBAsyncFFAdd_pptLT">
<plt:role name="NestedESBAsyncFFAdd_pptProvider">
<plt:portType name="tns:NestedESBAsyncFFAdd_ppt" />
</plt:role>
</plt:partnerLinkType>
```

Oracle Mediator/ESB WSDL Bindings

A WSDL document binding provides the protocol and data format specification for a particular port type.

Oracle Mediator/ESB WSDL documents can have SOAP bindings or Oracle Mediator/ESB native bindings.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Note. Oracle Mediator/ESB WSDL documents to be consumed by PeopleSoft systems must be of the SOAP type. SOAP binding adhere to the SOAP protocol. Choosing the Oracle Mediator/ESB native binding type may result in unpredictable behavior in PeopleSoft systems.

Asynchronous Request/Response Callback Operations

The WSDL created by Oracle Mediator for asynchronous request/response integrations does not contain a service definition for the callback operation. The PeopleSoft system does not recognize the callback until the service definition is defined.

As a result, save the WSDL document to the file system and add the following section to the WSDL document. Then consume the WSDL document in the Consume Web Services Wizard, selecting the File option when specifying the WSDL source.

```
<wsdl:service name="asyncrrcalcbpelcallback_client_ep">
   <wsdl:port name="AsyncRRCalcBPELCallback_pt" binding="client:
    AsyncRRCalcBPELCallbackBinding">
      <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       location="http://Client.EndPoint.Set.By.Caller"/>
   </wsdl:port>
</wsdl:service>
```

PeopleSoft Integration Broker Consume Services Wizard

To consume Oracle Mediator/ESB-based services use the PeopleSoft Integration Broker Consume Services Wizard. The wizard enables you to consume WSDL provided by integration partners. In turn, the wizard creates PeopleSoft service, service operation, message, and routing definitions for the WSDL, thereby enabling you to manage and invoke the service in PeopleSoft.

See Chapter 21, "Consuming Services," Understanding the Consume Web Service Wizard, page 515.

Providing Oracle Mediator/ESB–Based Services for Consuming in PeopleSoft

This section lists the general steps for providing Oracle Mediator/ESB–based services for consuming in PeopleSoft systems. All steps are performed in Oracle Mediator/ESB except where otherwise noted:

1. Create the Mediator/ESB project.

   Remember that the type of routing service (synchronous, asynchronous, or asynchronous request/response) you associate to the project determines the project type (synchronous, asynchronous, asynchronous request/response).

2. Register the project with the Oracle Mediator/ESB server.

   Use the Oracle Mediator/ESB Console to verify that you successfully registered the project.

3. On the machine on which Oracle Mediator/ESB is deployed, verify that the virtual host and port values are set.

   By default the virtual host is set to *localhost.* You must set the virtual host to the IP address or host name.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. On the Definitions tab, note the WSDL URL of the routing service.

Note. Write down the WSDL URL of the routing service. You must enter this URL in the PeopleSoft system to consume the service.

Consuming Oracle Mediator/ESB-Based Services

To consume WSDL from a Mediator/ESB-based service into the PeopleSoft system, use the Consume Web Service component in the PeopleSoft Pure Internet Architecture.

As you use the Consume Services Wizard keep the following points in mind:

- The source of the WSDL is a WSDL URL. Enter the URL of the routing service that you noted on the Definitions tab in Oracle Mediator/ESB.
- When prompted to select service ports select the service(s) who's name start with "SOAP." This ensures that the PeopleSoft system uses the correct binding type to create the routing metadata.
- Make a note of the service operations that you consume. You need these names when you create PeopleCode to create the request message and invoke the service.
- When prompted to select a receiving node select the Use Exiting Node option. Next, enter or search for the external node that you created to represent the Mediator/ESB system.
- When consuming asynchronous services you can use the default queue option or select a different queue.

See Also

Chapter 21, "Consuming Services," page 515

Invoking Synchronous Oracle Mediator/ESB-Based Services

This section discusses how to:

- Manage routing definitions for invoking synchronous Oracle Mediator/ESB-based services.
- Set message and error logging for invoking synchronous Oracle Mediator/ESB-based services.
- Invoke a synchronous Oracle Mediator/ESB-based service.

Prerequisites for Invoking Synchronous Oracle Mediator/ESB-Based Services

Before you can invoke a synchronous Oracle Mediator/ESB-based service, you must first consume it. Consuming Oracle Mediator/ESB-based services is described elsewhere in this section.

See Chapter 23, "Integrating with Oracle Mediator and Oracle ESB-Based Services," Consuming Oracle Mediator/ESB-Based Services, page 563.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Managing Routing Definitions for Invoking Synchronous Oracle Mediator/ESB-Based Services

The PeopleSoft Integration Broker Consume Services Wizard creates an outbound routing definition for the consumed service.

Before you invoke a synchronous Oracle Mediator/ESB-based service in the PeopleSoft system, ensure the following for the routing definition:

On the Routings-Connector Properties page (PeopleTools, Integration Broker, Integration Setup, Routings. Click the Connector Properties tab) and perform the following tasks:

- Ensure that the Connector ID field is set to *HTTPTARGET*.
- Ensure that the value in the Primary URL field is set to the endpoint URL.
- Set any other HTTP target connector properties as desired.

Setting Message and Error Logging for Invoking Synchronous Oracle Mediator/ESB-Based Services

To capture the maximum amount to message and error logging and information when you invoke the service, set the following options:

- On the Routings- Routing Definitions page (PeopleTools, Integration Broker, Integration Setup, Routings), set the value in the Log Detail field to *Header and Detail.*
- In the integrationGateway.properties file, set the ig.log.level property equal to *5*.

Invoking a Synchronous Oracle Mediator/ESB-Based Service

The following sample PeopleCode demonstrates one way to invoke a synchronous Mediator/ESB-based service:

```
Local string &payload, &responseStr;
Local Message &msg, &reply;
Local XmlDoc &xml;

/* --- set input request message--- */
&payload = "<?xml version='1.0' encoding='UTF-
8'?><SynchAdd_BPELProcessRequest
xmlns='http://xmlns.oracle.com/SynchAdd_BPEL'><input1>1234</input1><input2>12
34</input2></SynchAdd_BPELProcessRequest>";
MessageBox(0, "Request Message", 0, 0, &payload);

&xml = CreateXmlDoc(&payload);
&msg = CreateMessage(Operation.ESBSYNCADD, %IntBroker_Request);
&msg.SetXmlDoc(&xml);

&reply = %IntBroker.SyncRequest(&msg);

   If All(&reply) Then
     &responseStr = &reply.GenXMLString();
     MessageBox(0, "Request Message", 0, 0, (&responseStr));
   Else
     WinMessage("Error. No reply (or NULL) from LaunchSyncBPELProcess");
End-If;
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

On execution the system displays the request message that the system sends to Oracle Mediator/ESB. If the invocation is successful, the system displays another box that shows the response from Oracle Mediator/ESB.

Use the PeopleSoft Integration Broker message logs to view the flow of the messages. You can also use the Integration Broker Service Operations Monitor to view information about the outbound transaction.

Use the Oracle Mediator/ESB Console to check for the Mediator/ESB instance and for the successful invocation of the Mediator/ESB service.

See Chapter 19, "Managing Error Handling, Logging, Tracing, and Debugging," Managing Integration Gateway Message and Error Logging, page 470 and *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Synchronous Service Operations."

Invoking Asynchronous Oracle Mediator/ESB-Based Services

This section discusses how to:

- Create acknowledgements for invoking asynchronous Oracle Mediator/ESB-based services.
- Manage routing definitions for invoking asynchronous Oracle Mediator/ESB-based services.
- Set message and error logging for invoking asynchronous Oracle Mediator/ESB-based services.
- Invoke an asynchronous Oracle Mediator/ESB-based service.

Understanding Invoking Asynchronous Oracle Mediator/ESB-Based Services

An asynchronous service is also referred to as a fire-and-forget service.

Prerequisites for Invoking Asynchronous Oracle Mediator/ESB-Based Services

Before you can invoke an asynchronous Oracle Mediator/ESB-based service, you must first consume it. Consuming Oracle Mediator/ESB-based services is described elsewhere in this section.

See Chapter 23, "Integrating with Oracle Mediator and Oracle ESB-Based Services," Consuming Oracle Mediator/ESB-Based Services, page 563.

Creating Acknowledgments for Invoking Asynchronous Oracle Mediator/ESB-Based Services

To handle acknowledgement data for an asynchronous service operation, create an OnReceive handler that uses the OnAckReceive method that is implemented as an application class. After doing so, you must register the application class with the PeopleSoft service operation.

The following code example shows sample code to create the handler:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PS_PT:Integration:IReceiver;

class AsynchFF_AckReceive implements PS_PT:Integration:IReceiver
   method AsynchFF_AckReceive();
   method OnAckReceive(&MSG As Message) Returns integer;
end-class;

/* constructor */
method AsynchFF_AckReceive
end-method;

method OnAckReceive
   /+ &MSG as Message +/
   /+ Returns Integer +/
   /+ Extends/implements PS_PT:Integration:IReceiver.OnAckReceive +/
   /* Variable Declaration */

If &MSG.IsStructure Then

   /* if message is rowset-based */
   Local string &str = &MSG.GenXMLString();

Else
   /* if message is nonrowset-based */
   Local XmlDoc &xmldoc = &MSG.GetXmlDoc();
   Local string &str1 = &xmldoc.GenXmlString();
   Local File &MYFILE;

   &MYFILE = GetFile("C:\Temp\ESB\PS_Consume\ESBasynchffack.txt", "W",
%FilePath_Absolute);
   If &MYFILE.IsOpen Then
      &MYFILE.WriteString(&str1);
      &MYFILE.WriteLine("");
      &MYFILE.WriteString(String(%Operation_Done));
      &MYFILE.Close();
     End-If;
   End-If;

   Return (%Operation__Done);
end-method;
```

To create an acknowledgement for consumed asynchronous Oracle Mediator/ESB-based services:

1. Create an OnReceive handler that uses the OnAckReceive method that is implemented as an application class.

   See Chapter 16, "Managing Service Operation Handlers," Implementing Handlers Using Application Classes, page 353.

2. Open the service operation definitions for the consumed service operation.

3. Click the Handler tab.

   The Service Operations-Handlers page appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Add a handler with the following characteristics:

Name	Enter any name. For example: *AckReceive.*
Type	From the drop-down list select *OnReceive.*
Implementation	From the drop-down list select *Application Class.*
Status	From the status drop-down list select *Active.*

5. Click the Details link.

   The Handler Details page appears.

6. At a minimum enter the following details about the application class:

Package Name	Enter the package name that contains the class that you want to specify.
Path	Enter *:*(a colon).
Class ID	Enter the name of the application class that contains the method that you want to specify.
Method	From the drop-down list select *OnAckReceive.*

Managing Routing Definitions for Invoking Asynchronous Oracle Mediator/ESB-Based Services

The PeopleSoft Integration Broker Consume Services Wizard creates an outbound routing definition for the consumed service.

Before proceeding to invoke the service, ensure the following on the routing definition:

- On the Routings-Routing Definitions page (PeopleTools, Integration Broker, Integration Setup, Routings), in the OnReceive Handler field, enter or select *OnAckReceive.*

  This is the acknowledgement.

- On the Routings-Connector Properties page (PeopleTools, Integration Broker, Integration Setup, Routings. Click the Connector Properties tab).

  - Ensure that the Connector ID field is set to *HTTPTARGET.*
  - Ensure that the value in the Primary URL field is set to the endpoint URL.
  - Set any other HTTP target connector properties as desired.

Setting Message and Error Logging for Invoking Asynchronous Oracle Mediator/ESB-Based Services

To capture the maximum amount of message and error logging and information when you invoke the service, in the integrationGateway.properties file, set the ig.log.level property equal to *5.*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

A setting of *5* is the default value.

Invoking an Asynchronous Oracle Mediator/ESB-Based Service

The following sample PeopleCode demonstrates one way to invoke the asynchronous Oracle Mediator/ESB-based service:

```
Local string &payload, &responseStr;
Local Message &msg, &reply;
Local XmlDoc &xml;

/* --- setting the input request message --- */

&payload = "<?xml version='1.0' encoding='UTF-8'?><SynchAdd_BPELProcessRequest
xmlns='http://xmlns.oracle.com/SynchAdd_BPEL'><input1>6789</input1><input2>6789
</input2></SynchAdd_BPELProcessRequest>";
MessageBox(0, "Request Message", 0, 0, &payload);

&xml = CreateXmlDoc(&payload);

&msg = CreateMessage(Operation.ESBASYNCFFADD, %IntBroker_Request);

&msg.SetXmlDoc(&xml);

%IntBroker.Publish(&msg);
&responseStr = &msg.TransactionId;

MessageBox(0, "Message Transaction ID-Response from LaunchAsyncBPELProcess", 0, 0,
  &responseStr);

MessageBox(0, "Acknowledgement from ESB", 0, 0, "For Ack Msg from ESB ,
  go look at the file C:\temp\ESB\PS_Consume\ESBasynchffack.txt");
```

When you execute the PeopleCode the PeopleSoft system the following information in consecutive message boxes:

1. The request message that it sends to Oracle Mediator/ESB.
2. The transaction ID for the asynchronous request.
3. The file location of the acknowledgement it received from Oracle Mediator/ESB

You can use the PeopleSoft Integration Broker message logs to view the flow of the messages. You can also use the Integration Broker Service Operations Monitor to view information about the outbound transaction.

You can also use the Oracle Mediator/ESB Console to check for the message instance and for the successful invocation of the Mediator/ESB service.

See Chapter 19, "Managing Error Handling, Logging, Tracing, and Debugging," Managing Integration Gateway Message and Error Logging, page 470 and *PeopleTools 8.52: Integration Broker Service Operations Monitor*, "Monitoring Asynchronous Service Operations."

Invoking Asynchronous Request/Response Oracle Mediator-Based Services

This section discusses invoking asynchronous request/response Oracle Mediator-based services.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Prerequisites for Invoking Asynchronous Request/Response Oracle Mediator-Based Services

Before you can invoke an asynchronous Oracle Mediator/ESB-based service, you must first consume it. Consuming Oracle Mediator/ESB-based services is described elsewhere in this section.

See Chapter 23, "Integrating with Oracle Mediator and Oracle ESB-Based Services," Consuming Oracle Mediator/ESB-Based Services, page 563.

Invoking an Asynchronous Request/Response Oracle Mediator-Based Service

To invoke an asynchronous request/response Oracle Mediator-based service, you must use an OnResponse handler. The following code shows a sample OnResponse handler:

```
import PS_PT:Integration:INotificationHandler;
import PT_BPEL:BPELUtil;

class Esb_AsyncRR implements PS_PT:Integration:INotificationHandler
   method Esb_AsyncRR();
   method OnResponse(&MSG As Message);
end-class;

/* constructor */
method Esb_AsyncRR
end-method;

method OnResponse
   /+ &MSG as Message +/

   Local Message &request, &response;
   Local File &MYFILE;
   Local XmlDoc &xml, &inxml;
   Local string &payload, &oper, &input1, &input2, &result_st;
   Local integer &result_no;
   Local array of XmlNode &nodes;
   Local XmlNode &node;
   Local PT_BPEL:BPELUtil &obj = create PT_BPEL:BPELUtil();
   &nodes = CreateArray(&node);
   &request = &MSG;
   &inxml = &request.GetXmlDoc();
   &nodes = &inxml.GetElementsByTagName("input1");
   &input1 = &nodes [1].NodeValue;
   &nodes = &inxml.GetElementsByTagName("input2");
   &input2 = &nodes [1].NodeValue;
   &result_no = Integer(&input1) * Integer(&input2);
   &result_st = String(&result_no);

   &payload = "<?xml version='1.0'?><PSFTCalcResponseMessage xmlns='
   http://xmlns.oracle.com/Enterprise/Tools/schemas/PSFTCALCRESPONSEMESSAGE.V1'>
   <result>" | &result_st | "</result></PSFTCalcResponseMessage>";
   &xml = CreateXmlDoc(&payload);
   &response = CreateMessage(Operation.ESBASYNCRR, %IntBroker_Response);
   &response.SetXmlDoc(&xml);
   &response.IBInfo.WSA_MessageID = &request.IBInfo.WSA_MessageID;
   &response.IBInfo.WSA_ReplyTo = &request.IBInfo.WSA_ReplyTo;

   &obj.UpdateConnectorResponseProperties(&response);
   %IntBroker.Publish(&response);


end-method;
```

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB

This section discusses how to:

- Provide PeopleSoft services to Oracle Mediator/ESB
- Invoke PeopleSoft services in Oracle Mediator/ESB

Understanding Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB

This section provides an overview of providing and invoking PeopleSoft services in Oracle Mediator/ESB.

Oracle Mediator/ESB Projects When PeopleSoft is a Provider

To invoke a PeopleSoft service in Oracle Mediator/ESB, you must create a project in Oracle Mediator/ESB.

See *Oracle Mediator Quick Start Guide*

See *Oracle Enterprise Service Bus Quick Start Guide*

PeopleSoft Integration Broker Provide Services Wizard

To provide PeopleSoft services to Oracle Mediator/ESB you use the PeopleSoft Integration Broker Provide Services Wizard. The wizard enables you to generate WSDL documents based on services you create in PeopleSoft Integration Broker. You then provide the generated WSDL to Oracle Mediator/ESB.

Prerequisites for Providing and Invoking PeopleSoft Services in Oracle Mediator/ESB

You must create the PeopleSoft services you wish to provide, prior to using generating WSDL and providing the WSDL to Oracle Mediator/ESB.

Other prerequisites may exist in Oracle JDeveloper or Oracle Mediator/ESB. Refer to the Oracle documentation for more information.

Providing PeopleSoft Services

Use the PeopleSoft Integration Broker Provide Web Service Wizard to generate a WSDL document for the service you wish to provide to Oracle Mediator/ESB.

The wizard provides you the option to publish the WSDL to a UDDI repository that you define, or to the WSDL repository in PeopleSoft Integration Broker. You can choose either publishing option.

The last page is the WSDL Generation Log which provides the name of the services and URL for each WSDL document generated. You will use this information when you invoke the services in Oracle Mediator/ESB.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

Chapter 20, "Providing Services," page 479

Invoking PeopleSoft Services in Oracle Mediator/ESB

This section describes how to invoke synchronous and asynchronous services in Oracle ESB.

Creating a Mediator/ESB Project to Invoke PeopleSoft Services

After you use the PeopleSoft Integration Broker Provide Web Service wizard to generate WSDL, the next step is to invoke the services in Oracle Mediator/ESB, To do so, you must create a Mediator/ESB project type in Oracle JDeveloper.

See *Oracle Mediator Quick Start Guide*

See *Oracle Enterprise Service Bus Quick Start Guide*

The following are general steps to follow to configure the Oracle Mediator/ESB project after you initially create it. Refer to the Oracle documentation for detailed information and instructions:

1. Modify the project so that the SOAP service points to the PeopleSoft WSDL URL generated when you provided the service in PeopleSoft,
2. Use the Make option to build the project.
3. Use the Register with Mediator/ESB option to register the project with Oracle Mediator/ESB.

   This command allows you to register the services created using Oracle JDeveloper to Oracle Mediator or the Oracle Enterprise Service Bus.

A message appears that indicates you successfully registered the services with the Oracle Mediator/ESB server.

To view, monitor or make runtime adjustments to the Mediator/ESB configuration, use the Oracle Mediator/ESB Console.

When you have successfully registered the services on the Oracle Mediator/ESB Server, you can view them in the Services Panel in Oracle JDeveloper. Click the Mediator/ESB system to view the Mediator/ESB system flow in the Configuration Area on the right side.

Invoking a Service in Oracle

As of the printing date of this PeopleBook, Oracle Mediator/ESB does not feature a test page. To test and invoke Mediator/ESB web services, use the Oracle SOA Suite's Oracle Enterprise Manager.

See *Oracle Enterprise Manager documentation.*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 24

Using the Inbound File Loader Utility

This chapter provides an overview of the Inbound File Loader utility processing and discusses how to:

- Set up processing rules.
- Initiate file processing.
- Test inbound flat file processing.

Understanding the Inbound File Loader Utility

When external systems send inbound transactions consisting of flat files, you can use the Inbound File Loader utility to translate the incoming files into service operations and process them.

The Inbound File Loader utility provides two options for translating and processing files:

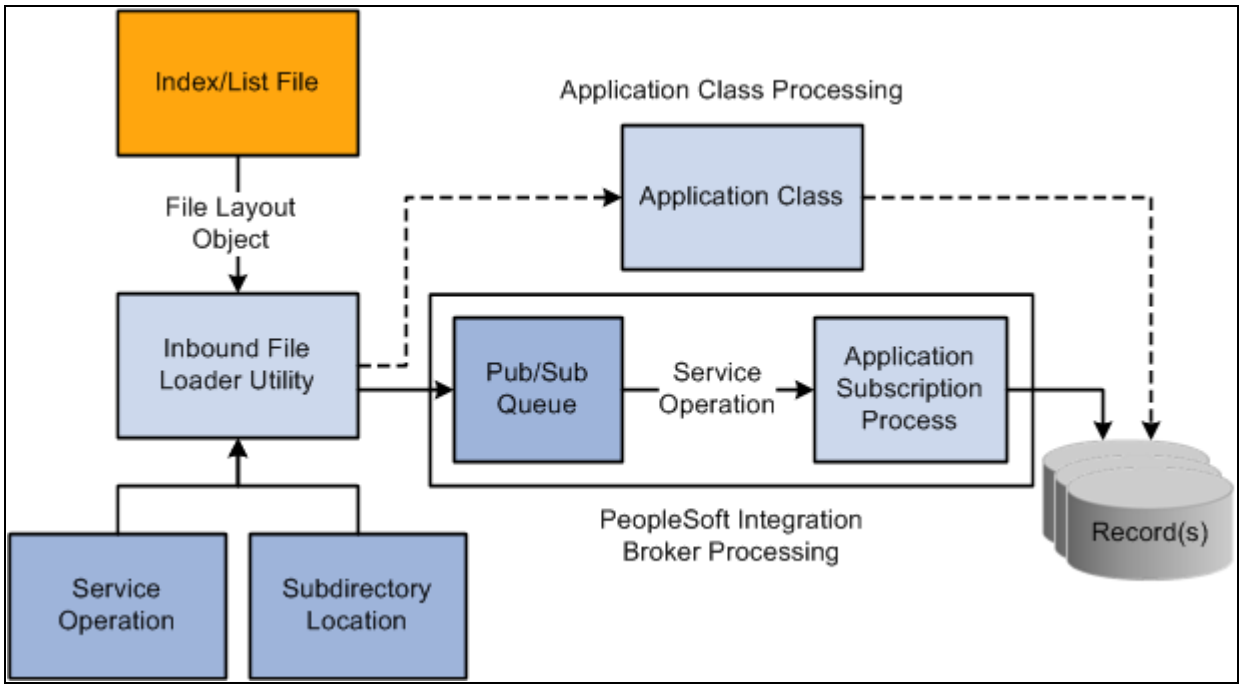
- Use PeopleSoft Integration Broker to convert file data into service operations and publish them locally. Then, subscribe to the service operations and insert the data into the tables.
- Write an application class that will read the contents of files and insert the data directly into the tables

Note. You can only use one processing method at any given time. Application class processing always overrides PeopleSoft Integration Broker processing.

File Processing

This section discusses the processing flow for the Inbound File Loader utility.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Inbound File Loader utility processing flow

The flow for inbound file processing using the Inbound File Loader utility is:

1. The utility receives a flat file in the form of a file layout object from an external system.

   The flat file consists of either:

   - A data file that contains the relevant data.
   - An index file that contains pointers to the data.

     Each index file lists the names of a set of data files to be processed. Each line of the index file must be a plain text file that contains only one field: a file name with full directory path information.

   These files contain the application data, which is in one of the following formats: fixed record, Comma Separated Values (CSV), or XML.

   Note. The wildcards "*" and "?" may only be used in the filename and not in the directory path.

2. The utility reads the file that is submitted for processing:

   - If the file is an index file, the Inbound File Loader utility loads the list of data files that are associated with each index file to be processed into a parameter table.
   - If it is a single data file, the utility inserts the single data file into a parameter table.

     Note. If additional fields in the file layout are not in the message definition, the additional fields are ignored during the copying of the flat file data to the message and are not included in the message.

3. The utility loops through the list of data files to be processed and reads each data file.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. The utility uses either PeopleSoft Integration Broker or application class processing logic to read the file contents and process the data.

   - PeopleSoft Integration Broker processing.

     When using Integration Broker processing logic, the Inbound File Loader utility copies the rowsets of the data files into the message, publishes the service operation locally, and then the receiving system receives the service operation and initiates normal inbound data processing.

   - Application class processing.

     You can build an application class to read the contents of the inbound file as rowsets and implement the necessary processing logic to write to the underlying tables.

5. To add file archiving, delete logic to prevent files from processing again, and so on, when defining processing rules, you can optionally specify an application engine program name and section. If specified, the utility calls the program and section as a final step to the inbound file process.

Understanding Development Activities

This section discusses development activities for using the Inbound File Loader utility and describes:

- General development activities
- Development activities for PeopleSoft Integration Broker processing.
- Creating file layout definitions.
- Development activities for application class processing.

General Development Activities

This section discusses general development activities for using the Inbound File Loader utility using either PeopleSoft Integration Broker processing or application class processing.

Determining the Format for Inbound Data

Determine the necessary format of the inbound data. If there is an industry standard, use it for your file definition. If there is no industry standard, create a file layout object.

Development Activities for PeopleSoft Integration Broker Processing

This section discusses development activities for using the PeopleSoft Integration Broker processing in conjunction with the Inbound File Loader utility.

1. Create a message definition in the PeopleSoft Pure Internet Architecture.

   The structure of the message definition must be similar to the structure of the file layout definition that you created.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. Create a service.
3. Create an asynchronous one-way service operation for the service.
4. Create a local-to-local routing definition for the service operation.
5. Create an OnNotification handler and call the functional library IB_FILE_SUBCODE.

   PeopleTools delivers this functional library and it can be used for any generic notification.

   The following example shows code an OnNotification handler calling the functional library:

```
import PS_PT:Integration:INotificationHandler;

class QUSubscribe implements PS_PT:Integration:INotificationHandler
   method QUSubscribe();
   method OnNotify(&_MSG As Message);
end-class;

Declare Function Subscribe PeopleCode FUNCLIB_IBFILE.IB_FILE_SUBCODE FieldFormula;

/* constructor */
method QUSubscribe
end-method;

method OnNotify
   /+ &_MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   /* Variable Declaration */

   Local Message &MSG;
   Local Rowset &MSG_ROWSET;

   &MSG = &_MSG;

   Subscribe(&MSG);

end-method;
```

6. Define processing rules in the Inbound File Loader Rules page in the PeopleSoft Pure Internet Architecture.
7. Initiate flat file processing using the Inbound File Processing page.
8. Test the inbound flat file processing.

Creating File Layout Definitions

When you use the Inbound File Loader utility, you use file layout definitions to read and write data from flat files. Use the following guidelines when creating file layout definitions:

- Create a file layout definition with the same structure as the message definition to support the vendor file format.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- The hierarchical structure of the data in the file layout definition must match that of the message definition. For example, suppose a message has three levels:

  - Level 0, containing record A.
  - Level 1, containing records B and C.
  - Level 2, containing record D.

  All file layouts that are associated with this message must also have record A in level 0, record B and C in level 1, and record D in level 2.

  Note. The file layout does not need to contain the exact same fields as the message definition

- For every record in a file layout definition, add a new file field, AUDIT_ACTN, as the first field in the record (except when the field already exists in the application table).
- You can associate more than one file layout to a single message. For example, vendor A may have a different number of fields than vendor B, so you may have two file layouts: one for vendor A and one for vendor B.
- Specify the file ID uniquely to include a row in a file, which is necessary in mapping the data to its proper record. Include start and end points when dealing with more than one record in a file layout.
- Each record in the file layout has a file record ID attribute. Do not confuse this with the file layout ID. The file layout ID determines whether a new layout is encountered for multiple file layout processing.

Development Activities for Application Class Processing

This section discusses development activities for application class processing using the Inbound File Loader utility. This section discusses how to:

1. Create an application class.
2. Specify processing rules.

Creating Application Classes

This section discusses creating an application class for processing flat files in conjunction with the Inbound File Loader utility.

The application class you create must implement the IProcessFile interface. The signature of the interface is shown here:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
interface IProcessFile
   /* Any initialization is done in this function*/
method Init (&fldDefName As string, &msgName As string) Returns boolean;

/* Contains processing logic that stores the Rowset data into the respective⇒
 tables */
method Process(&fldDefName As string, &msgName As string, &rs As Rowset) Returns⇒
 boolean;

/* This method shall contain logic for cleanup operations */
method Finish(&fldDefName As string, &msgName As string) Returns boolean;
end-interface;
```

The application class you create must implement the following three methods:

- Init
- Process
- Finish

If the Replace Data check box is selected in the Inbound File Loader Rule page, the Init method will be called. The Finish method is the last method to be invoked by the utility. Any post-processing clean up code can be implemented with this function.

The logic in the Process method stores the file contents in staging tables. You can add logic in the Finish method to move the data from staging tables to the actual transaction tables as a final process.

The Init, Process and Finish methods must return a boolean value of *True* for successful completion of the file processing. If methods Init and Finish are not used, return a default value of *True.*

The following example shows an application class implementing the IProcessFile interface:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
import PTIB:Interfaces:IProcessFile;

class InboundFileProcess implements PTIB:Interfaces:IProcessFile

method Init(&fldDefName As string, &msgName As string) Returns boolean;

method Process(&fldDefName As string, &msgName As string, &rs As Rowset) Returns⇒
 boolean;

method Finish(&fldDefName As string, &msgName As string) Returns boolean;

end-class;

method Init
   /+ &fldDefName as String, +/
   /+ &msgName as String +/
   /+ Returns Boolean +/
/+ Extends/implements PTIB:Interfaces:IProcessFile.Init +/

//This function will be called when the Replace Data flag is
//enabled
//add initialization code, such as cleaning up the table before
//reading in the data from the file



   Return True;
end-method;

method Process
   /+ &fldDefName as String, +/
   /+ &msgName as String, +/
   /+ &rs as Rowset +/
   /+ Returns Boolean +/
/+ Extends/implements PTIB:Interfaces:IProcessFile.Process +/

//Add the code that inserts/updates/delete data in the table

   Return True;
end-method;

method Finish
   /+ &fldDefName as String, +/
   /+ &msgName as String +/
   /+ Returns Boolean +/
   /+ Extends/implements PTIB:Interfaces:IProcessFile.Finish +/

 //This function will be called when the Replace Data flag is
//enabled
// Clean up logic goes here (if any)

   Return True;
end-method;
```

Specifying Processing Rules

After you create an application class, you must access the Inbound File Loader Rules page and specify the following information:

- Root Package ID.
- Path.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Class name.

Prerequisites for Using the Inbound File Loader Utility

The prerequisites for using the Inbound File Loader utility are:

- PeopleSoft Integration Broker must be configured and running.
- PeopleSoft Process Scheduler must be configured in PSAdmin.
- Create a file definition layout as described previously in this chapter.

  See Chapter 24, "Using the Inbound File Loader Utility," Creating File Layout Definitions, page 576.
- If using PeopleSoft Integration Broker processing, complete the development activities for PeopleSoft Integration Broker processing described previously in this chapter.

  See Chapter 24, "Using the Inbound File Loader Utility," Development Activities for PeopleSoft Integration Broker Processing, page 575.
- If using application class processing develop an application class that implements the IProcessFile interface and specify the processing rules as described previously in this chapter.

  See Chapter 24, "Using the Inbound File Loader Utility," Development Activities for Application Class Processing, page 577.

Setting Up Inbound File Loader Processing Rules

This section discusses how to set up inbound flat file processing rules.

Understanding Setting Up Inbound File Loader Processing Rules

The Inbound File Loader utility uses information you define in the Inbound File Loader Rules page to determine the file layout and message combination, as well as other file attributes necessary for processing files.

Setting Up Inbound File Loader Processing Rules

Use the Inbound File Loader Rules page (PSIBINFILERULE) to specify the file layout and message to process, as well as define the parameters for processing. To access the page, select PeopleTools, Integration Broker, File Utilities, Inbound File Loader Rules. The following examples shows the page:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Inbound File Loader Rules page

Note. You can process multiple inbound flat files at one time. By specifying an inbound index file as part of the Inbound File Loader utility parameters. The system reads all input files within the index file and uses the associated file layout object and message to convert the data. Similarly, specify a wildcard in the filename in the inbound file rule component, but make sure that all files that meet the wildcard criteria correspond to the file layout and message mapping that are defined.

File Identifier Displays the inbound file that you are associating with the rule.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Inbound File Enter the index file name or the data file name of the inbound file to process. Specify the full path information. The PeopleCode program uses the *%filepath_absolute* variable when opening the file.

File Type In the File Type section, select the type of inbound file. The options are:

- Data File
- Index File

File Layout ID (Optional.) Enter a file layout ID to associate with the file. The file layout ID is used in a multiple file layout processing. This identifier indicates that the data that follows belongs in a new file layout.

Status From the Status drop-down list, select whether this inbound file rule is active. The valid options are:

- Active.
- Inactive. (Default.)

Replace Data Check the box to indicate that the inbound file processing is a destructive load process.

A destructive load process involves replacing the contents of the tables with new data from the file being processed. In a destructive load process, the service operations must be subscribed in the same order as they are published to ensure transactional integrity.

If this check box is selected, the utility publishes a header message. The header message is a trigger in the subscription process to initialize tables before receiving the data messages. The subscription PeopleCode logic must check for the header message and perform any cleanup operation .

The utility then publishes data messages containing the data followed by a Trailer Message. The trailer message is used as a trigger in the subscription process to indicate that all the data messages have been received.

If the Replace Data check box is not selected, only data messages are published.

When used in the context of application class processing, if the Replace Data flag is selected, the Init() and Finish() methods of the specified application class are invoked.

Publish From Select the name of the publishing node.

Use this option to simulate an inbound asynchronous transaction from an external node.

While using this feature, you must create an inbound asynchronous transaction on the node from where the message is published.

Root Package ID Select the root package ID of the application class.

Path Select the qualifying path of the application class.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Class Name	Select the application class name that contains the file processing logic.
Program Name and**Section**	Select a PeopleSoft Application Engine program and section to invoke when the utility finishes processing data.
Definition Name	Specify the file layout definition for the file(s) being processed. If the File Layout ID field is blank, this field should contain only one entry. If the File Layout ID field is not blank, this scroll area must contain an entry for each file layout definition name that is specified in the inbound file.
Service Operation	For every row in this scroll area, specify a service operation. The utility uses the message(s) defined within the service operation to copy the file rowsets into message rowsets

Note. Use the wildcards "*" and "?" for the file name but not for the directory path. The file layout and service operation must be valid for all files that meet the wildcard criteria.

Initiating File Processing

This section discusses how to initiate inbound flat file processing.

Understanding Initiating File Processing

The Inbound File Processing page runs the Application Engine process PTIB_INFILE that initiates the file-to-message processing. The file-to-message processing function reads the file rowset and publishes it as a message.

If an index file exists when the inbound conversion process runs, the application engine program loads the list of files to be converted into a parameter table and completes a commit. The application engine program uses the list of files within the parameter table to restart the processing if a particular flat file fails. If a single data file is provided, then the rowset processing immediately begins.

The file publish process goes through each of the rowsets of the file layout and copies them into the message row sets.

If the audit action (AUDIT_ACTN) exists in the file, it is copied to the PSCAMA record. If the audit action does not exist in the file, the publishing process uses the default value that is specified in the file layout field property.

The Inbound File Loader utility publishes a new message when one of the following exists:

- The size of the data in the service operation exceeds the value of the Maximum Message Size field set in the PeopleTools Options page.

  To view the value in the field, select PeopleTools, Utilities, Administration, PeopleTools Options.

- A new file layout is detected.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

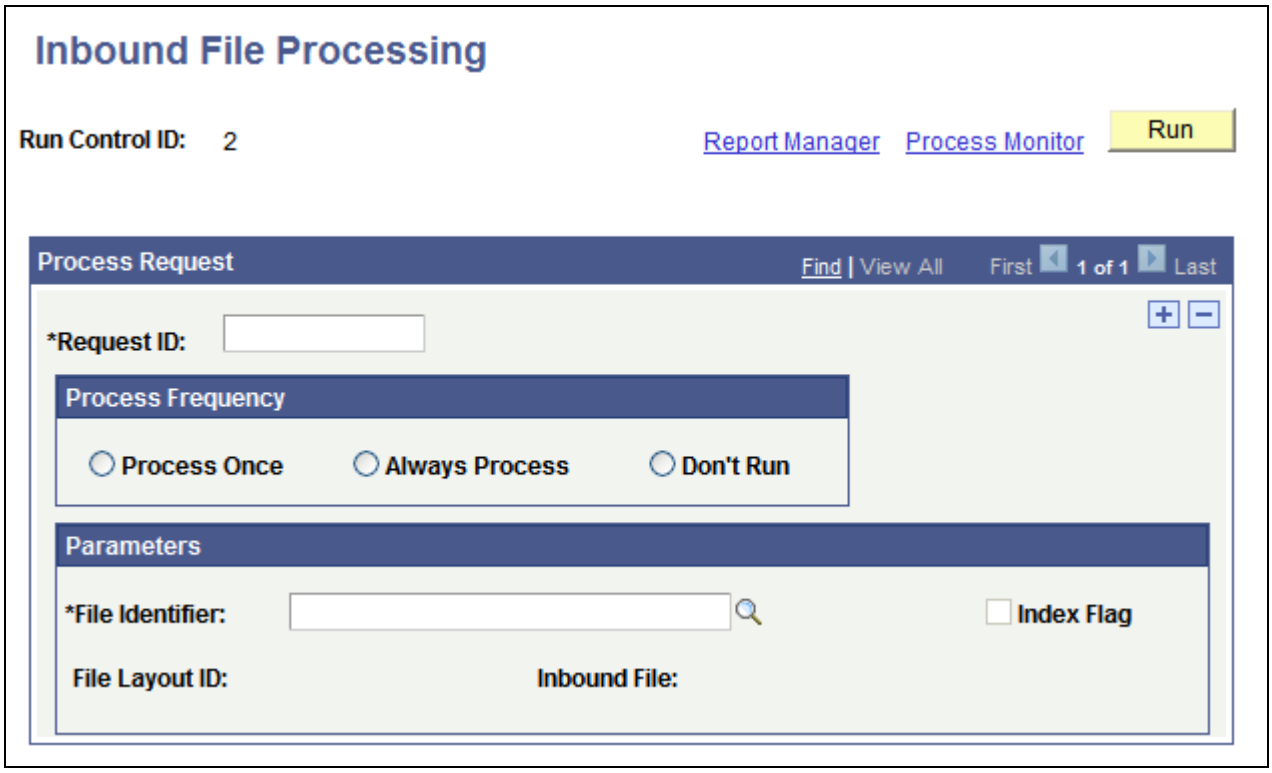
- End of file is reached.

The application engine program completes a commit every time a message is published from a file. After conversion, the flat file remains in the parameter table with a status of *Processed.*

Note. The file layout should exactly match the message layout (excluding the PSCAMA record) and should use the same character set as that used by the file: either American National Standards Institute or Unicode.

Initiating Inbound Flat File Processing

Use the Inbound File Processing page (PSIBFILERUNCNTL) to initiate flat file processing. Select PeopleTools, Integration Broker, File Utilities, Inbound File Processing. The Inbound File Processing page, shown in the following example, appears:



Inbound File Processing page

Request ID Enter a unique identifier to track the request.

Process Frequency Select the frequency for processing the file. The options are:

- Process Once.
- Always.
- Don't Run.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

File Identifier	Select or enter the name of the file identifier that you set up in the Inbound File Loader Rules. page. The file identifier is tied to the publish rules.
Index Flag	A read-only field that indicates if the file being processed is an index file or a data file. When checked, the Inbound File Loader utility is processing an index file.
File Layout ID	A read-only field that displays the file layout ID associated with the file in the Inbound File Loader Rules page.
Inbound File	A read-only field that displays the name of the file being processed.

Testing Inbound Flat File Processing

To test inbound files:

1. Create a sample flat file, or ask the third-party vendor for a sample flat file.
2. Set up processing rules for the test.

   Select PeopleTools, Integration Broker, File Utilities, Inbound File Loader Rules to access the Inbound File Loader Rules page.

   See Chapter 24, "Using the Inbound File Loader Utility," Setting Up Inbound File Loader Processing Rules, page 580.
3. Initiate file processing.

   Select PeopleTools, Integration Broker, File Utilities, Inbound File Processing to access the Inbound File Processing page.

   See Chapter 24, "Using the Inbound File Loader Utility," Initiating Inbound Flat File Processing, page 584.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Verify the inbound processing created a message that contains the sample flat file data.

    a. If you used application class processing, verify if the production or staging tables are loaded with the correct field values. For production tables, look in the PeopleSoft application pages. For staging tables, use either the PeopleSoft application pages or run a query by using PeopleSoft Query.

    b. If you used PeopleSoft Integration Broker for file processing, use the Service Operations Monitor to ensure the inbound file processing created a service operation that contains the sample flat file data.

        Verify that the standard inbound notification process received the message and processed it into the application tables.

        Determine whether the values become the inherited values (if you used the inherited value feature in file layout).

        Validate that the production or staging tables loaded with the correct field values. For production tables, look in the PeopleSoft application pages. For staging tables, use either the PeopleSoft application pages or run a query by using PeopleSoft Query.

        Ensure that the date formats conform.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Chapter 25

Copying Integration Metadata between PeopleSoft Databases

This chapter discusses how to:

- Copy integration metadata between PeopleSoft databases.
- Use data mover scripts to copy message schemas and WSDL.
- Converting message schemas and WSDL to managed objects.
- Manage nodes copied between databases and upgraded from earlier PeopleTools releases.

Copying Integration Metadata Between PeopleSoft Databases

This section provides an overview of copying metadata between PeopleSoft 8.52 databases and discusses data dependences and relationships when copying data.

Understanding Copying Integration Metadata Between PeopleSoft Databases

You can use PeopleSoft Application Designer's Project Copy functionality to copy integration metadata between PeopleSoft databases.

Message schema and WSDL became managed objects beginning with the PeopleTools 8.50 release. As a result, you can use Project Copy to copy message schemas and WSDL documents between PeopleTools 8.50 and higher databases. However, to copy schema and WSDL between PeopleTools 8.50 and higher databases and PeopleTools 8.49 or PeopleTools 8.48 databases, you must use the provided data mover scripts. These data mover scripts are discussed elsewhere in this chapter.

See Chapter 25, "Copying Integration Metadata between PeopleSoft Databases," Using Data Mover Scripts to Copy Message Schema and WSDL Data, page 590.

See Also

PeopleTools 8.52: PeopleSoft Application Designer Lifecycle Management Guide, "Copying Projects and Definitions"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding Data Dependencies and Relationships for Copying Data

When copying data between databases, you must be aware of data dependencies and relationships to ensure that no errors occur and to lessen the chance of encountering orphaned data.

Note. References to 'Project Copy' in the following table are references to the Project Copy feature in PeopleSoft Application Designer.

Object Name	*Comments*
Services.	You can use Project Copy to copy services between databases. WSDL documents that exist for a service are not automatically copied with a service. You must include them in the copy project.
WSDL documents.	To copy WSDL to another PeopleTools 8.52 system, use Project Copy. To copy WSDL to earlier releases of PeopleTools, you must use data mover scripts to copy the data to the target database.
Service operations.	A service operation is tied to a service. If you copy a service operation in a project, the target database must already contain the service to which the service operation is tied in the database. If it does not, you must include that service in the copy project. Service operations cannot exist in a database without at least one service operation version - the default version. So when copying a service operation between databases, you need to be aware what the default service operation is and that you may possibly have to copy it to the target database as well. In addition, keep in mind that the relationship between services and service operations is stored as part of the Service object. So for example, if a service contains three operations and you delete one of the three operations, you must include the service in the project. After the operation delete, the service is now linked to the remaining two operations and so it has changed and would need to be copied to the target database. It is not be enough to only copy the deleted operation in a Delete project; Deleting just the service operation doesn't delete the link to the service. If you don't follow this recommendation, you'll have orphaned data.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Object Name	*Comments*
Service operation versions.	A service operation version refers to a specific service operation. If you copy a service operation version, the target database must already contain the service operation. If it does not, you must include that service operation in the copy project. In addition, service operation versions refer to messages. If you copy a service operation version, the messages that are referenced for that service operation version must exist on the target database. If they do not, you must include them in the copy project. If WSDL documents have been generated for a service operation version, they are not automatically copied during the Project Copy process. Further, once you have copied a service operation version to the target database, it may appear that WSDL documents exist for a service operation version, when they do not. To avoid this situation, after you copy a service operation version to the target database, open the service definition to which the service operation belongs. If the View WSDL link appears, and when you click it WSDL appears, go back to the source database and export the generated WSDL documents to the target database. Another option is to delete the WSDL documents associated with a service operation before the Project Copy, and regenerate them on the target database.
Service operation handlers.	A service operation handler refers to a specific service operation. If you copy a service operation handler, the target database must already contain the service operation to which the handler refers. If it does not, you must include that service operation in the copy project.
Service operation routings.	Routing names are keys in the system. If you copy a routing, the sending and receiving nodes must defined on the target database. If they are not defined on the target database, you must include them in your copy project. Routings reference a specific service operation version. If you copy a routing, the target database must already contain the service operation version to which the routing refers. If it does not, you must include that service operation version in the project. Routings also reference nodes. If you copy a routing, the target database must already contain the nodes being referenced. An exception to this is the local default node. During project copy, any routing referencing the local default node will be modified to reference the default local node of the target system. If the system detects a duplicate routing definition during the project copy process, the routing definition in the project being copied overwrites the routing definition in the database. This internal check occurs only when you use the Copy from File option during project copy. To identify duplicate routings when using the Database to Database option during project copy, or to detect and delete any duplicate routings in the database outside the project copy process, run the duplicate routings check using the Delete Duplicate Routings section of the Service Administration–Routings page. See Chapter 17, "Managing Service Operation Routing Definitions," Deleting Duplicate Routing Definitions, page 413.
Messages.	Container messages and message parts must have message schemas to function properly. You should also move message schemas along with your messages.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Object Name	Comments
Service operation queues.	NA
Message schemas.	To copy message schemas to another PeopleTools 8.52 system, use Project Copy. To copy message schemas to earlier releases of PeopleTools, you must use data mover scripts to copy the data to the target database. You should copy message schema along with all messages you copy from one database to another.
Documents	When you include a document in a copy project, the system inserts the document and all related definitions into the project. To copy a document message type, when you specify the message, you must also include the document as a related definition.
Integration Groups	NA

See Also

Chapter 25, "Copying Integration Metadata between PeopleSoft Databases," Using Data Mover Scripts to Copy Message Schema and WSDL Data, page 590

Using Data Mover Scripts to Copy Message Schema and WSDL Data

The following table lists the data mover scripts that PeopleSoft provides to move message schema and WSDL documents between PeopleTools 8.52 databases and PeopleTools 8.48 databases or PeopleTools 8.49 databases. These scripts are located in the <PS_HOME>\scripts directory.

Object	Script Name	Description
Message schema.	PSIBMSGSCHEMA_EXP.DMS	Export a message schema from a PeopleTools 8.52 database.
Message schema.	PSIBMSGSCHEMA_IMP.DMS	Import a message schema into a PeopleTools 8.52 database.
WSDL document.	PSIBWSDL_EXP.DMS	Export a WSDL document from a PeopleTools 8.52 database.
WSDL document.	PSIBWSDL_IMP.DMS	Import a WSDL document into a PeopleTools 8.52 database.

The WSDL data mover scripts move WSDL by WSDL name, not service name. Therefore it is possible to select specific WSDL for importing/exporting for a given service.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

You may encounter errors while moving large WSDL documents and schemas from a Microsoft SQL, Oracle or Informix platform to a Sybase or DB2 (UNIX or OS/390) platform, because of size restrictions in Sybase and DB2. The maximum size of WSDL documents from Microsoft SQL, Oracle or Informix platforms to DB2 or Sybase platforms is described in the following table:

Platform	*Maximum WSDL Size (Bytes)*
DB2 – UNIX	32700
DB2 – OS/390	31744
Sybase	32000

Converting WSDL Documents and Message Schemas to Managed Objects

This section discusses how to:

- Use the Metadata WSDL/Schema Convert page.
- Convert WSDL documents and message schemas to managed objects.
- Convert message schemas to managed objects.
- Delete data from the deprecated data repository.

Understanding Converting WSDL Documents and Message Schema to Managed Objects

This section discusses converting WSDL documents and XML message schema to managed objects. It also discusses the deprecated data repository.

WSDL Documents and Message Schema as Managed Objects

Beginning in PeopleTools 8.50, WSDL documents and XML message schema became managed objects.

After you copy WSDL and schema from earlier releases, PeopleSoft Integration Broker enables you to convert the data to managed objects. (To copy WSDL and XML message schema from earlier versions of PeopleTools, use the data mover scripts described previously in this chapter.)

See Chapter 25, "Copying Integration Metadata between PeopleSoft Databases," Using Data Mover Scripts to Copy Message Schema and WSDL Data, page 590.

In addition, during the PeopleTools upgrade process the system automatically attempts to convert WSDL and schema metadata into managed objects. However, if the system is unable to convert the copied or upgraded data, you can convert it in the PeopleTools 8.52 system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Deprecated Data Repository

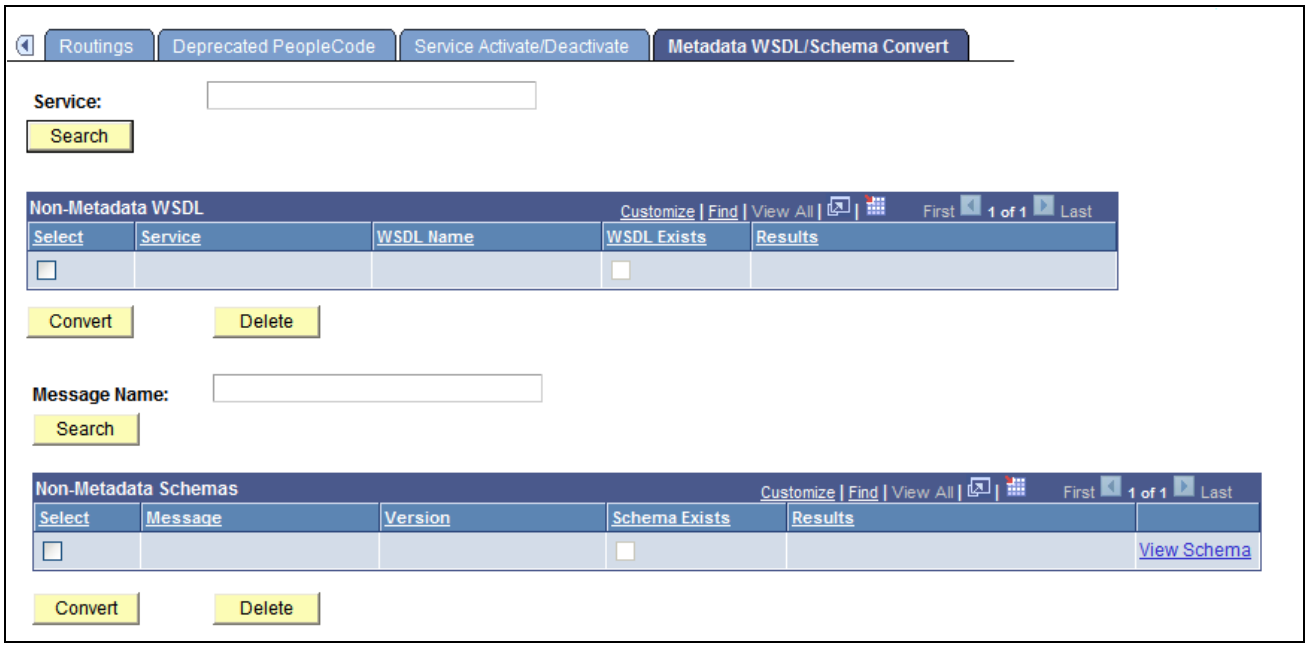
WSDL documents and XML message schema data that you copy from earlier PeopleTools releases to a PeopleTools 8.52 database is stored in a deprecated data repository in the database. WSDL and schema that do not property convert to managed objects during the upgrade process are also stored in this repository.

After you convert WSDL and schema to managed objects, the system moves the data to the metadata repository.

You can leave data that you do not convert in the deprecated data repository, or you can delete if from the system.

Using the Metadata Convert/Schema Convert Page

Use the Service Administration – Metadata Convert/Schema Convert page (IB_META_CONV) to convert WSDL documents and message schemas to managed objects. To access the page select PeopleTools, Integration Broker, Service Utilities, Service Administration and click the Metadata WSDL/Schema Convert tab. The following graphic shows the Service Administration – Metadata Convert/Schema Convert page:



Metadata WSDL/Schema Convert page

Use the top portion of the page to convert WSDL documents to managed objects. Use the bottom portion of the page to convert XML message schemas to managed objects. The page enables you to work with one WSDL document or message schema at a time, or you can work with all data in the deprecated data repository at once.

When working with message schema, you can also use the provided View Schema link on the page to view schemas before converting them.

This page also enables you to delete data from the deprecated data repository that you do not want to convert to managed objects or that you no longer need.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

The following sections describe how to accomplish these tasks in greater detail.

Converting WSDL Documents to Managed Objects

This section discusses how to convert WSDL documents to managed objects.

To convert a WSDL document to a managed object:

1. Access the Service Administration – Metadata Convert/Schema Convert page (PeopleTools, Integration Broker, Service Utilities, Service Administration, Metadata Convert/Schema Convert).
2. Select the WSDL document(s) to convert:
   - To select a specific service for which to convert WSDL, in the Service field, enter the service name.
   - To select from all services in the deprecated data repository, click the Search button under the Service field.

   Search results appear in the Non-Metadata WSDL grid.
3. In the Non-Metadata WSDL grid, check the box next to each service name for which to convert WSDL.

   If a check appears in the WSDL Exists box, WSDL already exists as a managed object for the service. You can choose to use the existing WSDL, or select the box in the Select column to convert the WSDL again and overwrite the existing WSDL.
4. Click the Convert button under the Non-Metadata WSDL grid.
5. The status of the conversion for each service you selected appears in the Results field.

Converting Message Schemas to Managed Objects

This section discusses how to convert XML message schema to managed objects.

To convert message schemas to managed objects:

1. Access the Service Administration – Metadata Convert/Schema Convert page (PeopleTools, Integration Broker, Service Utilities, Service Administration, Metadata Convert/Schema Convert).
2. Select the XML message schema(s) to convert:
   - To select a specific schema to convert, in the Message Name field, enter the message name.
   - To select from all schema in the deprecated data repository, click the Search button under the Message Name field.

   Search results appear in the Non-Metadata Schemas grid.
3. In the Non-Metadata Schemas grid, check the box next to each message name that contains the schema to convert.

   If a check appears in the Schema Exists box, XML message schema already exists as a managed object for the message. You can choose to use the existing managed object schema, or select the box in the Select column to convert the schema again and overwrite the existing schema.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Click the Convert button under the Non-Metadata Schemas grid.
5. The status of the conversion for each message schema that you converted appears in the Results field.

If you are unable to convert a schema to a managed object, rebuild the schema:

- For rowset-based messages, open the message in the Messages-Message Definition page and save the message. Upon Save, PeopleSoft Integration Broker automatically builds the schema for the message.
- For nonrowset-based messages use one of the following options to rebuild the schema:
  - Access the schema using the View Schema link on the Metadata Convert/Schema Convert page and copy it to the Messages-Schema page and correct and save the schema.
  - Access the schema using the View Schema link on the Metadata Convert/Schema Convert page and copy it into an XML editor and correct the schema. Then upload it into the system using the Messages-Schema page.
  - Upload or create a new schema using the Messages-Schema page.

See Also

Chapter 6, "Managing Messages," Managing XML Message Schemas for Rowset-Based Messages, page 97

Chapter 6, "Managing Messages," Adding XML Message Schemas to Nonrowset-Based Messages, page 100

Deleting Data from the Deprecated Data Repository

This section discusses how to delete WSDL documents and XML message schemas from the deprecated data repository.

To delete data from the deprecated data repository:

1. Access the Service Administration – Metadata Convert/Schema Convert page (PeopleTools, Integration Broker, Service Utilities, Service Administration, Metadata Convert/Schema Convert).
2. Select the WSDL document(s) or XML message schema(s) to delete:
   - To select a specific WSDL document to delete, in the Service field, enter the service name.
   - To select from all services in the deprecated data repository, click the Search button under the Service field.
   - To select a specific schema to delete, in the Message Name field, enter the message name.
   - To select from all schema in the deprecated data repository, click the Search button under the Message Name.

   Search results appear in the appropriate WSDL or schema grid.
3. In the WSDL or schema grid, check the Select box next to the data to delete.
4. Click the Delete button under the grid in which the data to delete appears.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Managing Nodes Copied Between Databases and Upgraded from Earlier PeopleTools Releases

The user ID on any nodes that you copy to a database using Project Copy or that you upgrade from earlier PeopleTools releases must be:

- Valid in the target database.
- Assigned to the permission lists of any service operation that you intend to use.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Appendix A

Integration Scenarios

This appendix provides an overview of the basic integration scenarios you can implement using PeopleSoft Integration Broker and discusses how to:

- Integrate with PeopleSoft Integration Broker systems.
- Integrate with PeopleSoft Integration Broker systems through a firewall.
- Integrate with PeopleSoft Integration Broker systems by using hubs.
- Integrate with third-party systems.
- Integrate with third-party systems by using remote gateways.
- Integrate with PeopleTools 8.47 and earlier PeopleTools 8.4x systems.
- Integrate with PeopleTools 8.1x systems.

Understanding Integration Setup

An integration engine is automatically installed as part of your PeopleSoft application, and an integration gateway is installed as part of the PeopleSoft Pure Internet Architecture. However, there's no requirement that your integration gateway be on the same machine as the integration engine.

In general, the high-level tasks that you perform to configure any of the integration scenarios are:

- Define a local integration gateway.
- Define a remote integration gateway.
- Set integration gateway properties.
- Set up a local node.
- Set up a remote node.
- Create service operations with inbound and outbound routing definitions.

You may not need to perform all of these tasks. For example, if you don't need to communicate through a firewall, you probably don't need to define a remote gateway.

Application messaging, the precursor to PeopleSoft Integration Broker, employed content-based routing—each message had to provide its own routing information, which was defined in the message header. With PeopleSoft Integration Broker, you define the routing information separately. You can apply multiple routings to a message and change the routings independent of the message definition.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Defining Local and Remote Integration Gateways

On each PeopleSoft Integration Broker system in your configuration, you must specify a local integration gateway. The local gateway is the application's first point of contact with other PeopleSoft applications, third-party systems, PeopleSoft Integration Broker hubs, and remote gateways. You must define exactly one local gateway for each integration engine, but a single installed gateway can serve multiple engines. The web server where the integration gateway resides can be any machine on which you've installed the PeopleSoft Pure Internet Architecture.

To define the local gateway, use the Gateways component (IB_GATEWAY) to:

- Add the gateway that the application will use to communicate with other systems.
- Specify the uniform resource locator (URL) of the gateway's PeopleSoft listening connector.

  This is the connector that receives messages from an integration engine (including the default local node) or another integration gateway.
- Register the target connectors that are delivered with PeopleSoft Integration Broker.

  These target connectors are automatically installed during the PeopleSoft Pure Internet Architecture installation process. When you subsequently define local and remote nodes, you specify—from the list of installed target connectors—which connector the local gateway should use to send messages to each node.

Note. The remote gateway default connector setting in the integrationGateway.properties file, ig.connector.defaultremoteconnector, determines through which connector the gateway routes messages that are bound for other gateways. By default, this property is set to the HTTP target connector, HTTPTARGET. Never change this setting unless you develop a custom connector to handle this routing.

An integration gateway also includes a set of listening connectors, which are likewise installed with the PeopleSoft Pure Internet Architecture. You don't to need to specify these connectors directly; third-party systems send messages to the gateway by specifying the URL of an appropriate listening connector.

Setting Integration Gateway Properties

You set integration gateway properties by using the gateway's primary configuration file, called integrationGateway.properties. You must set an Oracle Jolt connect string in this file to enable communication with each PeopleSoft Integration Broker node that will be involved in an integration that uses this gateway.

The following example shows the required Oracle Jolt connect string setting:

```
ig.isc.<nodename>.serverURL=//<machine_name>:<jolt_port>
ig.isc.<nodename>.userid=<database_user_id>
ig.isc.<nodename>.password=<database_password>
ig.isc.<nodename>.toolsRel=<peopletools_release_version>
```

Note. You can also configure a default Oracle Jolt connect string to specify the target node to use when a message arrives at the gateway but doesn't specify a node name matching any of the existing entries. The default Oracle Jolt connect string settings are identical to the others, except that they don't include a node name. You can specify any PeopleSoft Integration Broker node as the default node, including any of the existing entries.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring Local and Remote Nodes

Use the Nodes component (IB_NODE) to configure local and remote nodes.

When you configure nodes, use the Nodes - Connectors page to specify the gateway and target connector to use to send messages to each node. Use the information in the following table as a guide for choosing the appropriate information for the configuration scenarios that are described in this appendix:

Scenario	*System*	*Node Definition*	*Connector*
PeopleSoft as a web service provider.	NA	NA.	Not applicable (NA)*
PeopleSoft Integration Broker to PeopleSoft Integration Broker.	Both applications.	Default local.	NA*
PeopleSoft Integration Broker to PeopleSoft Integration Broker.	Both applications.	Remote.	PSFTTARGET (PeopleSoft target connector)
PeopleSoft Integration Broker to PeopleSoft systems.	Both applications.	Default local.	NA*
PeopleSoft Integration Broker to PeopleSoft systems.	Both applications.	Remote.	PSFTTARGET (PeopleSoft target connector)
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a remote gateway.	Both applications.	Default local.	NA*
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a remote gateway.	Both applications.	Remote.	PSFTTARGET
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a hub.	Both applications.	Default local.	NA*
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a hub.	Both applications.	Remote.	PSFTTARGET

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Scenario	*System*	*Node Definition*	*Connector*
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a hub.	PeopleSoft Integration Broker hub.	Default local.	NA*
PeopleSoft Integration Broker to PeopleSoft Integration Broker by using a hub.	PeopleSoft Integration Broker hub.	Remote	PSFTTARGET
PeopleSoft Integration Broker to a third party.	PeopleSoft Integration Broker.	Default local.	NA*
PeopleSoft Integration Broker to a third party.	PeopleSoft Integration Broker.	Remote.	Third-party connector, as appropriate: • HTTPTARGET (HTTP target connector) • JMSTARGET (Java Message Service [JMS] target connector) target connector) • FTPTARGET (File Transfer Protocol [FTP] target connector) • SMTPTARGET (Simple Mail Transfer Protocol [SMTP] target connector)
PeopleSoft Integration Broker to a third party.	Third-party system.	NA.	NA
PeopleSoft Integration Broker to a third party by using a remote gateway.	PeopleSoft Integration Broker.	Default local.	NA*

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Scenario	*System*	*Node Definition*	*Connector*
PeopleSoft Integration Broker to a third party by using a remote gateway.	PeopleSoft Integration Broker.	Remote.	Third-party connector, as appropriate: • HTTPTARGET • JMSTARGET • FTPTARGET • SMTPTARGET
PeopleSoft Integration Broker to a third party by using a remote gateway.	Third-party system.	NA.	NA
PeopleSoft Integration Broker to PeopleSoft Integration Broker on PeopleTools 8.47 and earlier PeopleTools 8.4x systems.	Both applications.	Default local.	NA
PeopleSoft Integration Broker to PeopleSoft Integration Broker on PeopleTools 8.47 and earlier PeopleTools 8.4x systems.	Both applications.	Remote.	PSFTTARGET (PeopleSoft target connector)
PeopleSoft Integration Broker to PeopleTools 8.1x systems.	PeopleSoft Integration Broker.	Default local.	NA*
PeopleSoft Integration Broker to PeopleTools 8.1x systems.	PeopleSoft Integration Broker.	Remote.	PSFT81TARGET (PeopleSoft 8.1 target connector)
PeopleSoft Integration Broker to PeopleTools 8.1x systems.	PeopleSoft 8.1x system.	NA. Set up message nodes, message channels, messages, and so on.	NA

* The default connector is PSFTTARGET, but it is not used.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

See Also

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Using the integrationGateway.properties File

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

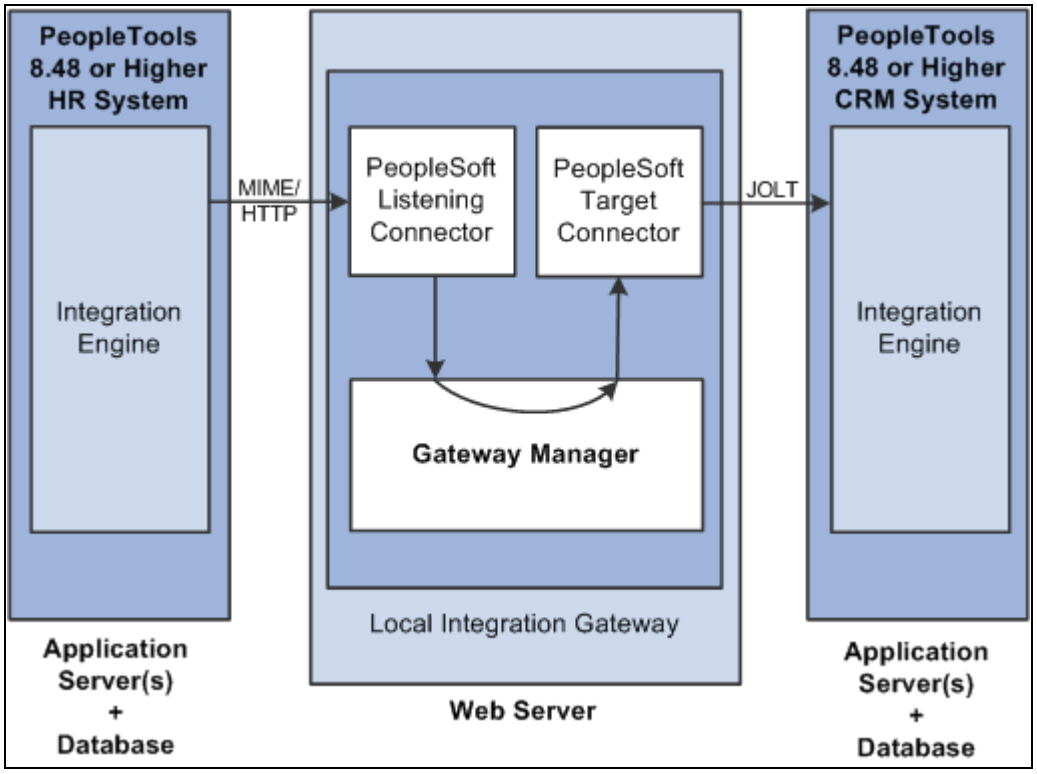
PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Integrating with PeopleSoft Integration Broker Systems

This section provides an overview of this scenario and discusses how to configure the system for this scenario.

Understanding This Scenario

This diagram shows a PeopleSoft Human Resources system communicating with a PeopleSoft Customer Relationship Management (PeopleSoft CRM) system and shows the configuration and interaction of PeopleSoft Integration Broker components:



Integrations with PeopleSoft Integration Broker systems

This communication can be synchronous or asynchronous.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the System for This Scenario

This section describes the source and destination system configuration tasks, based on the scenario shown in the previous diagram. In this example, the PeopleSoft Human Resources system is the source system and the PeopleSoft CRM system is the destination system.

This section discusses how to configure:

- The integration gateway.
- The PeopleSoft Human Resources system.
- The PeopleSoft CRM system.

Configuring the Integration Gateway

The only required property that you must set for the local gateway is the Oracle Jolt connect strings that enable the gateway to find the PeopleSoft CRM system. Set this property in the integrationGateway.properties file.

Configuring the PeopleSoft Human Resources System

Perform the following tasks on the PeopleSoft Human Resources system:

1. Define the local integration gateway for the PeopleSoft Human Resources system by using the Gateways component.

   Any integration gateway that you've installed and configured to find the PeopleSoft CRM system can serve this role. Specify the gateway's PeopleSoft listening connector as the gateway's URL.

2. Configure the default local node definition that represents the PeopleSoft system by using the Nodes component.

   This node is delivered predefined on the system.

3. Define a remote node to represent the PeopleSoft CRM system.

   Because the PeopleSoft CRM system uses PeopleSoft Integration Broker, specify the local gateway for the PeopleSoft Human Resources system and its PeopleSoft target connector on the Node Definitions - Connectors page.

4. Define a service operation that specifies the request message, the service operation handler definition and routing definition.

   The routing is a point-to-point routing where the PeopleSoft CRM node is the receiving node and the PeopleSoft HR system is the sending node.

Configuring the PeopleSoft CRM System

Perform the following tasks on the PeopleSoft CRM system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

1. Define the local integration gateway for the PeopleSoft CRM system by using the Gateways component.

   Any integration gateway that you've installed and configured can serve this role, including the local gateway for the PeopleSoft Human Resources system. Specify the gateway's PeopleSoft listening connector as the gateway's URL.

2. Configure the default local node definition that represents the PeopleSoft CRM system by using the Nodes component.

   This node is delivered predefined on the system.

3. Define a remote node to represent the PeopleSoft Human Resources system.

   Because the PeopleSoft Human Resources system uses PeopleSoft Integration Broker, specify the local gateway for the PeopleSoft CRM system and its PeopleSoft target connector on the Node Definitions - Connectors page.

4. Define a service operation that specifies the request message, the service operation handler definition and routing definition.

   The routing is a point-to-point routing where the PeopleSoft CRM node is the receiving node and the PeopleSoft HR system is the sending node.

5.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Integrating with PeopleSoft Integration Broker Systems Through Firewalls

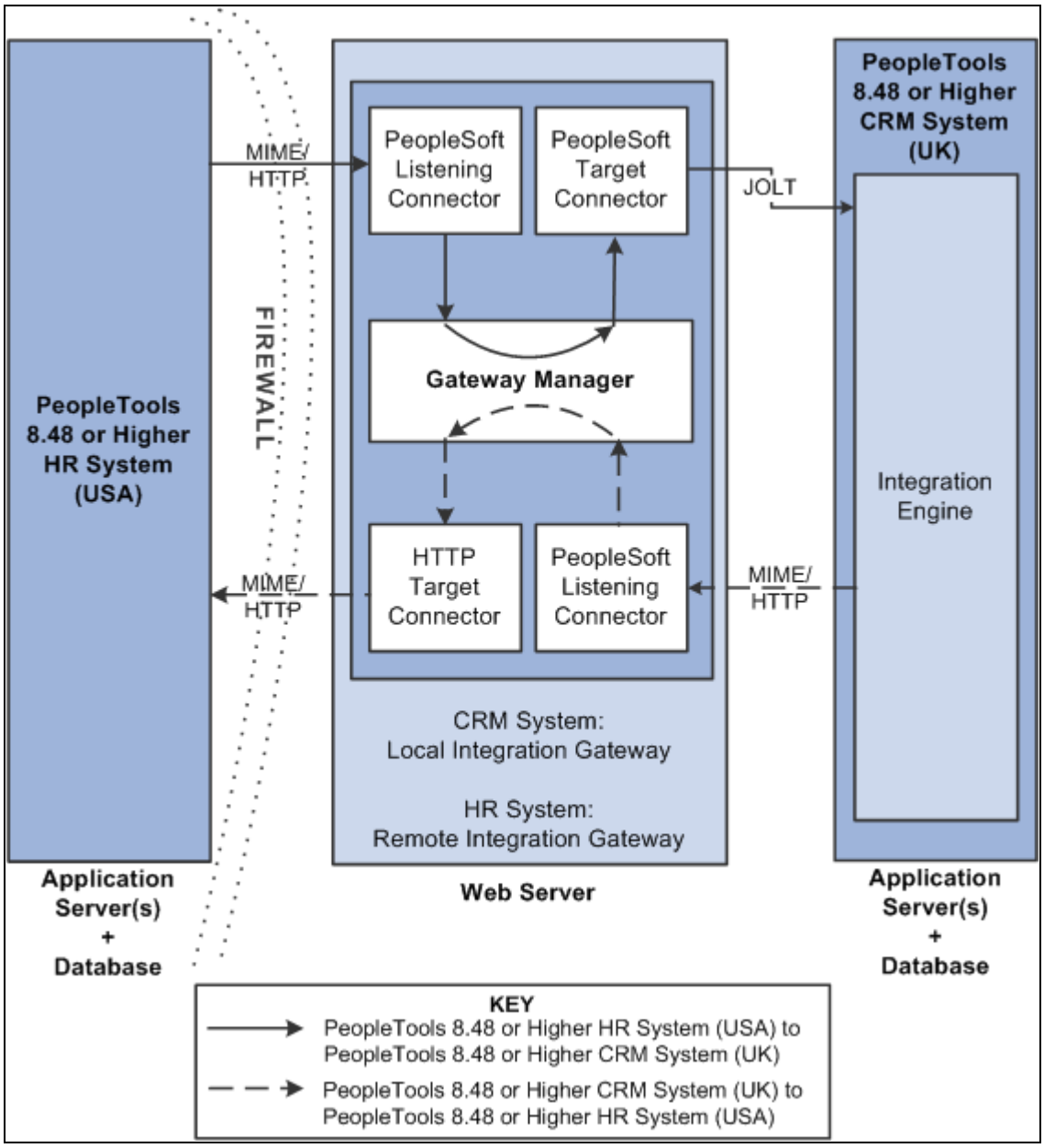
This section provides an overview of this scenario and discusses how to configure the system for this scenario.

Understanding This Scenario

Use a remote gateway configuration when connections with an integration participant are not possible through the internet. This type of implementation enables you to communicate with wide area networks (WANs) and local area networks (LANs) where a firewall is present.

This diagram shows the configuration of PeopleSoft Integration Broker components for integrations with other PeopleSoft Integration Broker systems by using a remote gateway:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Integrations with PeopleSoft Integration Broker systems by using firewalls.

For this configuration scenario, one PeopleSoft application and one integration gateway reside on each side of the firewall. The integration gateway can reside on the same physical machine on which you have installed the PeopleSoft application, or it can reside on its own machine.

In this configuration scenario, PeopleSoft Integration Broker uses the default remote gateway connector, the HTTP target connector, on the local gateway to send messages to the PeopleSoft listening connector on the remote gateway. Routing all messages through the local gateway enables each PeopleSoft Integration Broker system to keep its own centralized log of all integration messages.

Because this example shows two-way communication and assumes that the same service operation is being exchanged, the PeopleSoft Human Resources (USA) system and the PeopleSoft CRM (UK) system are source systems when they send messages, and they're destination systems when they receive messages.

Keep in mind the following as you review these configuration tasks:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- You should use a single integration gateway for all applications that reside on the same side of a firewall.
- The local integration gateway for one application is the remote integration gateway for the other application.

Configuring the System for This Scenario

This section describes the configuration tasks for each of the components that are shown in the previous diagram.

This section discusses how to configure:

- The PeopleSoft Human Resources (USA) system.
- The PeopleSoft Human Resources (USA) integration gateway.
- The PeopleSoft CRM (UK) system.
- The PeopleSoft CRM (UK) integration gateway.

Configuring the PeopleSoft Human Resources (USA) System

On the PeopleSoft Human Resources (USA) system:

1. Define a local integration gateway.

   Use the Gateways component to define the local PeopleSoft Human Resources (USA) gateway.

2. Define a remote integration gateway.

   The remote integration gateway for the PeopleSoft Human Resources (USA) system is the PeopleSoft CRM (UK) gateway. Use the Gateways component to define a new gateway, and for the gateway URL, specify the PeopleSoft listening connector of the PeopleSoft CRM (UK) gateway.

3. Define the default local node.

   Use the Nodes component to define the default local node, which represents the PeopleSoft Human Resources (USA) system.

4. Define a remote node.

   The remote node that you define represents the PeopleSoft CRM (UK) system. When you set up the remote node, specify the PeopleSoft CRM (remote) integration gateway and the PeopleSoft target connector on that gateway.

5. For the outbound integration, define a service operation that specifies the request and response messages, the service operation handler definition, and the outbound routing definition.

   The outbound routing is a point-to-point routing where the PeopleSoft HR node is the sending node and the PeopleSoft CRM node is the receiving node.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

6. For the inbound integration, on the same service operation create a routing definition where the PeopleSoft HR system is the receiving node and the PeopleSoft CRM system is the sending node.

Note. The external alias you specify on the inbound routing definition must match the external alias on the outbound routing definition you created in the previous step.

Configuring the PeopleSoft Human Resources (USA) Integration Gateway

The only required integration gateway property that you must set for the PeopleSoft Human Resources (USA) integration gateway is the Oracle Jolt connect strings that enable communication with the integration engine on the PeopleSoft Human Resources (USA) system. Set this property in the integrationGateway.properties file.

Configuring the PeopleSoft CRM (UK) System

On the PeopleSoft CRM (UK) system:

1. Define a local integration gateway.

   Use the Gateways component to define the local PeopleSoft CRM (UK) gateway.

2. Define a remote integration gateway.

   The remote integration gateway for the PeopleSoft CRM (UK) system is the PeopleSoft Human Resources (USA) gateway. Use the Gateways component to define a new gateway, and for the gateway URL, specify the PeopleSoft listening connector of the PeopleSoft Human Resources (USA) gateway.

3. Define the default local node.

   Use the Nodes component to define the default local node, which represents the PeopleSoft CRM (UK) system.

4. Define a remote node.

   The remote node that you define represents the PeopleSoft Human Resources (USA) system. When you set up the remote node, specify the PeopleSoft Human Resources (remote) integration gateway and the PeopleSoft target connector on that gateway.

5. For the outbound integration, define a service operation that specifies the request and response messages, the service operation handler definition, and the outbound routing definition.

   The outbound routing is a point-to-point routing where the PeopleSoft HR node is the receiving node and the PeopleSoft CRM node is the sending node.

6. For the inbound integration, on the same service operation create a routing definition where the PeopleSoft HR system is the sending node and the PeopleSoft CRM system is the receiving node.

Note. The external alias you specify on the inbound routing definition must match the external alias on the outbound routing definition you created in the previous step.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the PeopleSoft CRM (UK) Integration Gateway

The only required integration gateway property that you must set for the PeopleSoft CRM (UK) integration gateway is the Oracle Jolt connect strings that enable communication with the integration engine on the PeopleSoft CRM (UK) system. Set this property in the integrationGateway.properties file.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Integrating with PeopleSoft Integration Broker Systems by Using Hubs

This section provides overviews of this scenario and hub routing types and discusses how to configure:

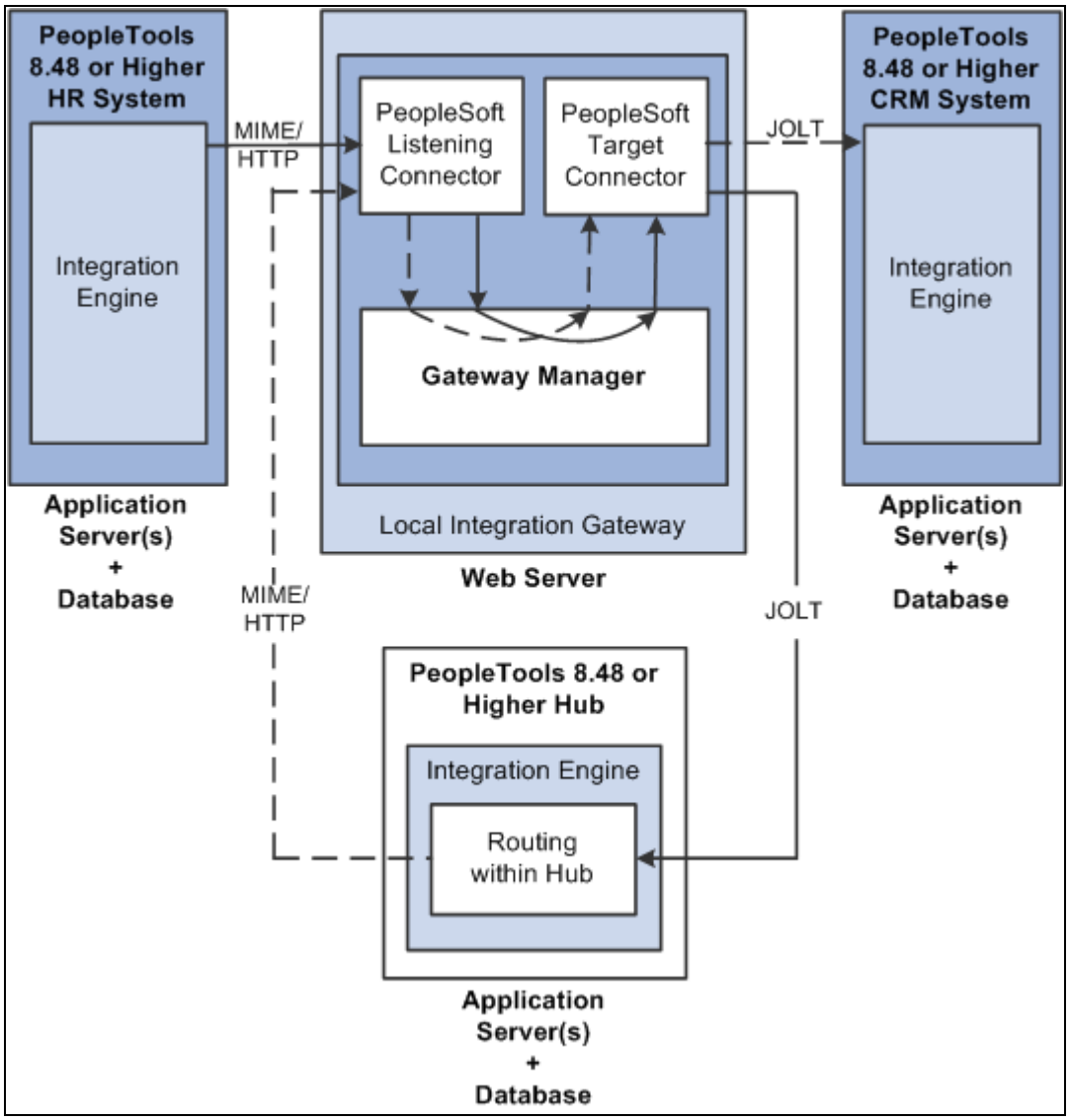
- Generic routing hubs.
- Sender-specified routing hubs.

Understanding This Scenario

A PeopleSoft Integration Broker hub configuration includes an integration engine that houses routing rules and transformations. All integrations are routed through the hub, which enables you to centralize routing rules and offload the transformation process.

This diagram shows a one-way hub configuration scenario that involves a PeopleSoft Human resources system and a PeopleSoft CRM system:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Integrations with PeopleSoft Integration Broker systems using a hub.

In this scenario, all of the routing rules and transformations are located on the hub.

To implement integrations between the two systems without a hub, you must set up a complete set of complementary routing rules and transformations on each node.

Understanding Hub Routing Types

There are two hub routing types: generic routing and sender-specified routing. The configuration steps for a hub vary, depending on which routing type you choose.

With generic routing, all transactions from the participating systems are sent directly to a hub for routing and transformation.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

With sender-specified routing, a destination node name is passed as a parameter to a PeopleCode Publish or SyncRequest method, such as %intBrokerPublish or %intBroker.SynchRequest, to explicitly route the outbound transactions to the necessary node. Using sender-specified routing requires that you define the explicit destination nodes on the sending system. However, you can configure the system so that PeopleSoft Integration Broker passes these outbound transactions to the hub for possible rerouting and transformation.

Note. You must use sender-specific routing when you're using PublishXMLDoc to asynchronously publish an XML object.

Regardless of which hub routing you use, you must configure each PeopleSoft application's integration engine, the integration gateway, and the PeopleSoft Integration Broker hub. A PeopleSoft Integration Broker hub can be an installed PeopleSoft application, or it can have only a stand-alone PeopleTools database installed, which includes the integration engine.

Configuring Generic-Routing Hubs

By using the elements in the previous diagram as an example, this section provides an overview of how to configure a generic-routing hub. In the scenario, the PeopleSoft Human Resources system, the PeopleSoft CRM system, and the hub must all point to the same integration gateway and use the same gateway URL.

This section discusses how to configure:

- The PeopleSoft Human Resources system.
- The PeopleSoft CRM system.
- The PeopleSoft hub.
- The integration gateway.
- The generic-routing hub.

Configuring the PeopleSoft Human Resources System

On the PeopleSoft Human Resources system:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Nodes component to set up the local node, which represents the PeopleSoft Human Resources system.

3. Set up a remote node.

   Use the Nodes component to set up the remote node, which represents the hub system.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Create a service operation.

   Use the Service Operations component to set up a service operation. Define the request message, service operation handler definition, and routing definition.

   Create a point-to-point routing definition where the PeopleSoft HR system is the sending node and the hub system is the receiving node.

Configuring the PeopleSoft CRM System

On the PeopleSoft CRM system:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Node Definition component to set up the local node, which represents the PeopleSoft CRM system.

3. Set up a remote node.

   Use the Node Definition component to set up the remote node, which represents the hub system.

4. Create a service operation.

   Use the Service Operations component to set up a service operation.

   Create an inbound point-to-point routing definition where the sending node is the hub system and the receiving node is the PeopleSoft CRM system.

Configuring the PeopleSoft Hub

On the PeopleSoft hub:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Node Definition component to set up the local node, which represents the hub system.

3. Set up remote nodes.

   Set up two remote nodes: one that represents the PeopleSoft Human Resources system and another that represents the PeopleSoft CRM system.

4. Create a service operation.

   Use the Service Operations component to set up a service operation.

   Create an outbound point-to-point routing definition where the sending node is the PeopleSoft HR system and the receiving node is the PeopleSoft CRM system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the Integration Gateway

You must set integration gateway properties for the local gateway. The only required properties are the Oracle Jolt connect string properties that enable communication with the integration engines on the PeopleSoft Human Resources, PeopleSoft CRM, and PeopleSoft hub systems. Set these properties in the integrationGateway.properties file.

Configuring the Generic-Routing Hub

For all messages going through the hub, you must set up a service operation and routing on the hub. By using the systems in the diagram as an example, the following table shows the node, service operation, and routing configurations that are required for generic routing through a hub:

Item to Configure	PeopleSoft Human Resources System	Integration Broker Hub	PeopleSoft CRM System
Local nodes	Rename the default local node to represent the PeopleSoft Human Resources system.	Rename the default local node to represent the hub.	Rename the default local node to represent the PeopleSoft CRM system.
Remote nodes	Define a remote node to represent the hub system.	Define remote nodes to represent the PeopleSoft Human Resources and CRM systems.	Define a remote node to represent the hub.
Service operations and routings.	Define outbound routing to the hub system.	Create a service operation that contains an outbound point-to-point routing definition where the sending node is the PeopleSoft HR system and the receiving node is the PeopleSoft CRM system.	Define inbound routing from the hub system.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring Sender-Specified Routing Hubs

By using the systems shown in the previous diagram as an example, this section provides an overview of how to configure a sender-specific routing hub. The PeopleSoft Human Resources system is the sending system and the PeopleSoft CRM system is the receiving system. In this scenario, the sending system, the receiving system, and the hub must all point to the same gateway and use the same gateway URL.

This section discusses how to configure:

- The PeopleSoft Human Resources (sending) system.
- PeopleSoft CRM (receiving) system.
- The PeopleSoft hub.
- The integration gateway.
- The sender-specified routing hub.

Configuring PeopleSoft Human Resources (Sending) System

On the PeopleSoft Human Resources system:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Nodes component to set up the local node, which represents the PeopleSoft Human Resources system.

3. Set up remote nodes.

   Set up two remote nodes: one for the receiving system (PeopleSoft CRM in the example) and one for the hub. When setting up the PeopleSoft CRM remote node, on the Nodes-Node Definitions page in the Hub Node field, enter the node name of the hub system.

4. Create a service operation.

   Use the Service Operations component to create a service operation that contains an outbound point-to-point routing definition where the sending node is the PeopleSoft HR system and the receiving node is the PeopleSoft CRM system.

Configuring PeopleSoft CRM (Receiving System)

On the PeopleSoft CRM system:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Nodes component to set up the local node, which represents the PeopleSoft CRM system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

3. Set up a remote node.

   Use the Nodes component to set up a node that represents the hub.

4. Create a service operation.

   Use the Service Operations component to create a service operation that contains an inbound point-to-point routing definition where the sending node is the hub system and the receiving node is the PeopleSoft CRM system.

5. Create a service operation.

   Use the Service Operations component to create a service operation that contains an inbound point-to-point routing definition where the sending node is the hub system and the receiving node is the PeopleSoft CRM system.

Configuring the PeopleSoft Hub

On the PeopleSoft hub:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Set up a local node.

   Use the Nodes component to set up the local node, which represents the hub system.

3. Set up remote nodes.

   Use the Nodes component to set up two remote nodes: one for the PeopleSoft Human Resources system and one for the PeopleSoft CRM system.

4. Create a service operation.

   Use the Service Operations component to create a service operation that contains a point-to-point routing where the sending node is the PeopleSoft HR node and the receiving node is the CRM node.

Configuring the Integration Gateway

The only required integration gateway properties for the local integration gateway are the Oracle Jolt connect string properties that enable communication with the integration engines on the target PeopleSoft Integration Broker systems. Set these properties in the integrationGateway.properties file.

Configuring the Sender-Specified Routing Hub

For all messages going through the hub, you must set up transactions and relationships on the hub. By using the systems in the previous diagram as example, the following table shows the node, transaction, and relationship configurations that are required for sender-specified routing through a hub from the PeopleSoft Human Resources system:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Item to Configure	*PeopleSoft Human Resources System*	*PeopleSoft Integration Broker Hub*	*PeopleSoft CRM System*
Local nodes	Rename the default local node to represent the PeopleSoft Human Resources system.	Rename the default local node to represent the hub.	Rename the default local node to represent the PeopleSoft CRM system.
Remote nodes	Define remote nodes to represent the PeopleSoft CRM and hub systems.	Define remote nodes to represent the PeopleSoft Human Resources and CRM systems.	Define a remote node to represent the hub.
Service operations and routings.	Create a service operation that contains an outbound point-to-point routing definition that specifies the receiving node is the PeopleSoft CRM system.	Create a service operation that contains a point-to-point routing where the sending node is the PeopleSoft HR node and the receiving node is the CRM node.	Create a service operation that contains an inbound point-to-point routing definition where the sending node is the hub system and the receiving node is the PeopleSoft CRM system.

All messages to the PeopleSoft CRM node are the result of publish statements, which include these target node parameters:

- msg.Publish(Node.CRM)
- SyncRequest(Node.CRM)
- %intBroker.publish(&MyDoc, Message.MyMessage, Node.CRM)
- %intBroker.SyncRequest(&MyDoc, Message.MyMessage, Node.CRM)

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Integrating with Third-Party Systems

This section provides an overview of this scenario and discusses how to configure the system for this scenario.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Understanding This Scenario

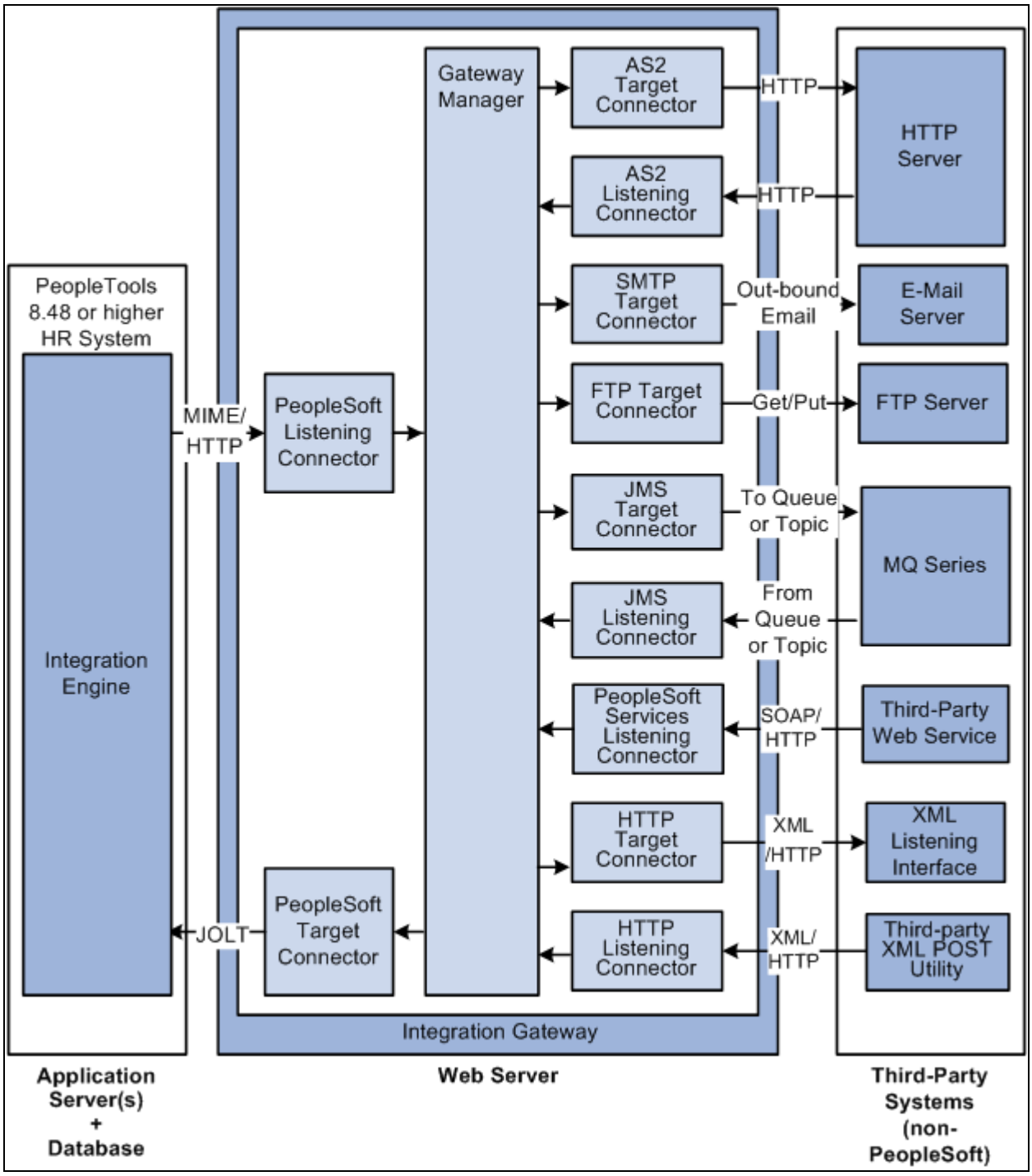
For communications with third-party systems, messages can go through local or remote gateways.

Sending a message to a third-party system is the same as sending a message to a PeopleSoft Integration Broker node, except that the target connector that you select depends on the third-party system with which you are communicating.

Messages from third-party systems can enter the gateway through any of the listening connectors that are delivered with PeopleSoft Integration Broker or through a listening connector that you build. You cannot use the PeopleSoft listening connector for integrations with third-party systems, because it can accept messages only in the PeopleSoft internal format.

This diagram shows the connectors that a PeopleSoft system can use to communicate with a third-party system and how the PeopleSoft system can communicate with third-party systems:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Integrations with third-party systems

Configuring the System for This Scenario

This section discusses how to configure:

- The PeopleSoft Human Resources system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- The PeopleSoft Human Resources integration gateway.

Configuring the PeopleSoft Human Resources System

On the PeopleSoft Human Resources system:

1. Define a local integration gateway.

   Use the Gateways component to set up a local integration gateway for sending messages.

2. Create a service operation.

   Use the Service Operations component to create a service operation that contains an inbound point-to-point routing definition where the sending node is the hub system and the receiving node is the PeopleSoft CRM system.

3. Set up a remote node.

   Set up a remote node that represents the third-party system. When you define this node, you select the appropriate connector (for example, JMS target connector, SMTP target connector, and so forth) for communicating with the third-party system.

4. Create a service operation with an inbound routing definition where the sending node is the third-party system and the receiving node is the PeopleSoft HR system.

5. In the service operation definition you created in the previous step, create an outbound routing definition where the sending node is the PeopleSoft HR system and the receiving node is the third-party system.

Configuring the PeopleSoft Human Resources Integration Gateway

The only required integration gateway properties for the local integration gateway are the default Oracle Jolt connect string properties that enable communication with integration engines on the PeopleSoft Human Resources system. Set these properties in the integrationGateway.properties file.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Integrating with Third-Party Systems by Using Remote Gateways

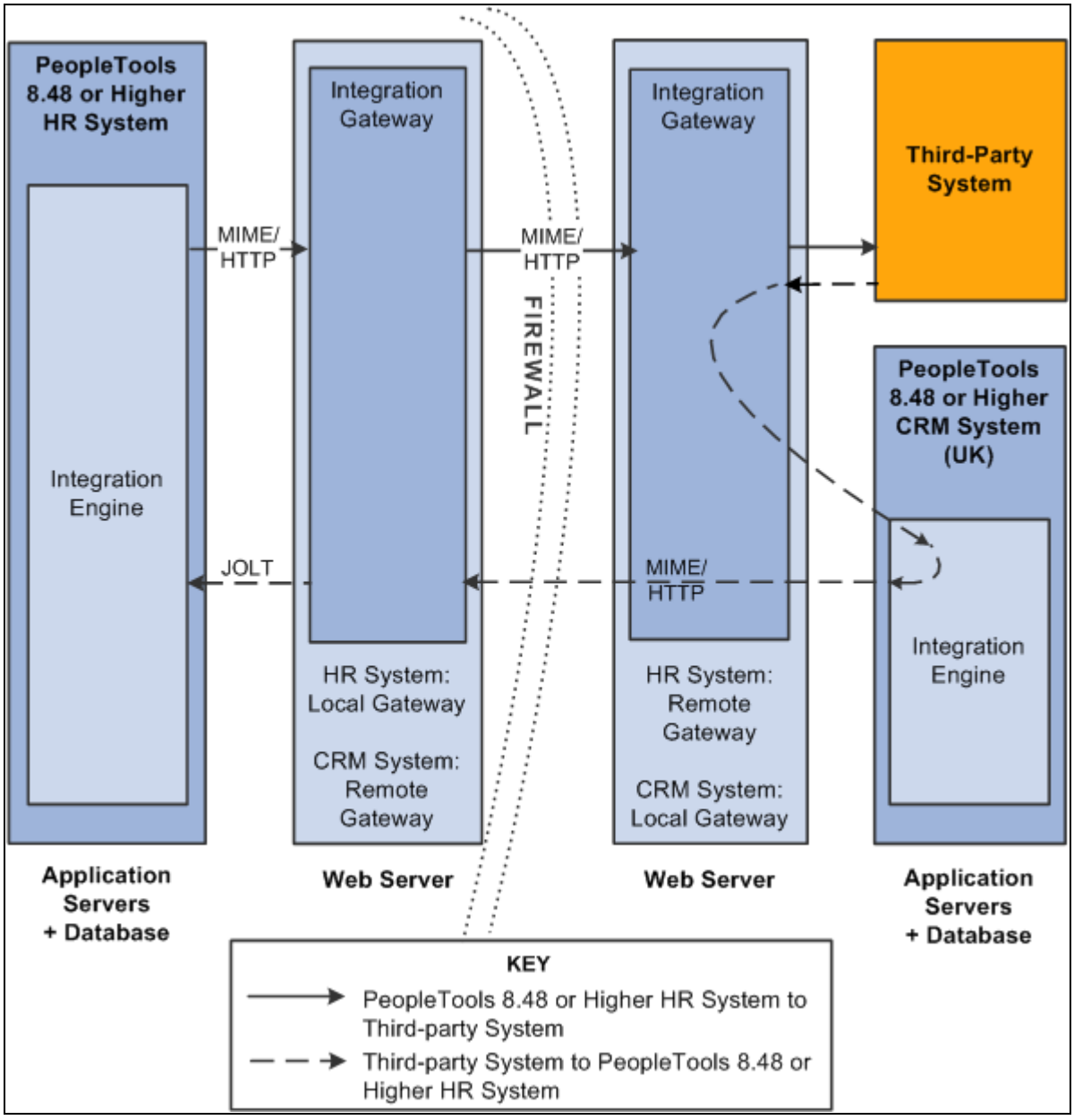
This section provides an overview of this scenario and discusses how to:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Send messages to third-party systems.
- Receive messages from third-party systems.

Understanding This Scenario

This diagram shows a high level view of how to integrate with third-party systems by using remote gateways:

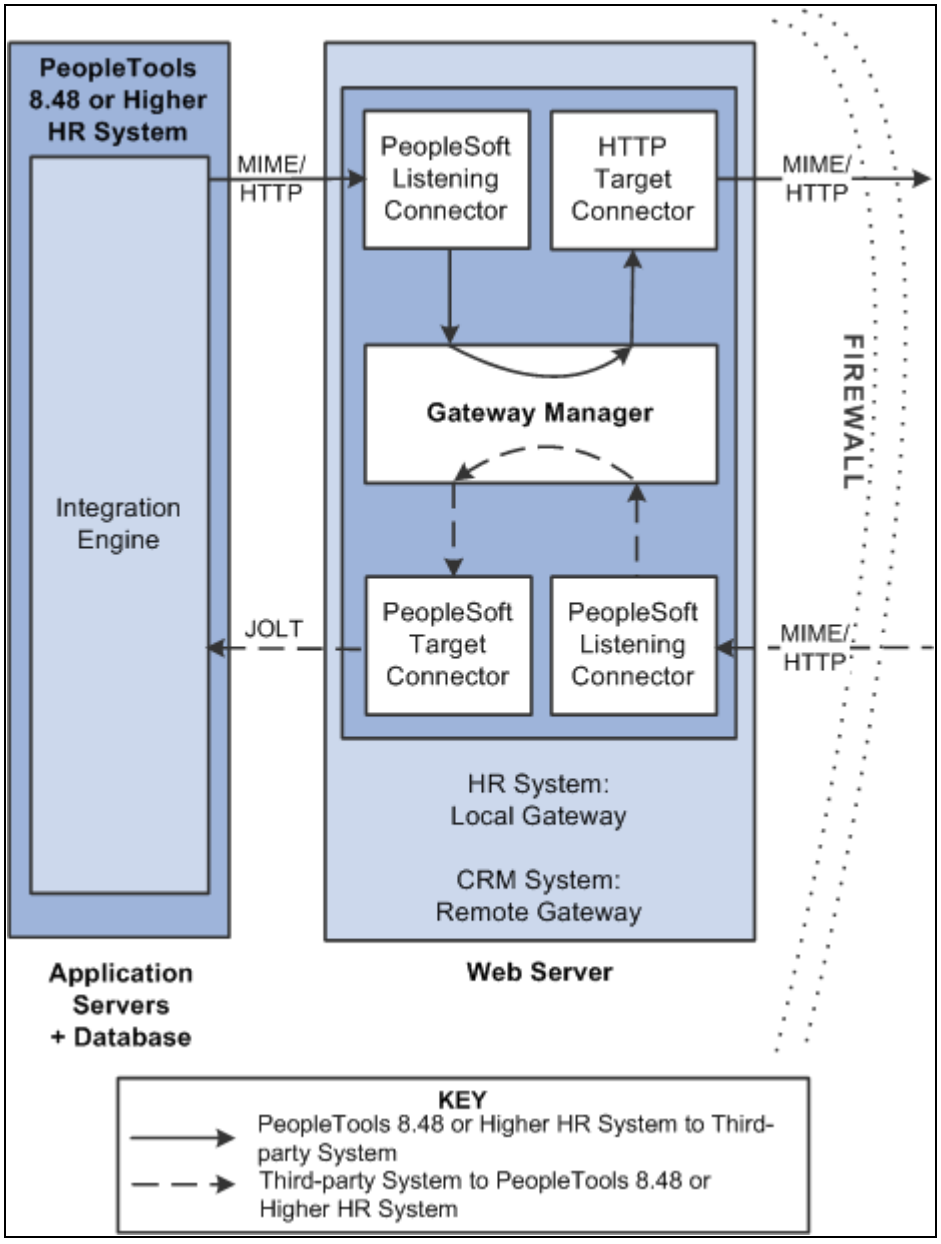


High-level diagram of integrations with third-party systems using remote gateways.

Integrations from one integration gateway to another can only be sent when they originate from a PeopleSoft system. The only exception to this is when integrations are performed through a custom connector that makes use of the IBRequest methods needed to handle this routing.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

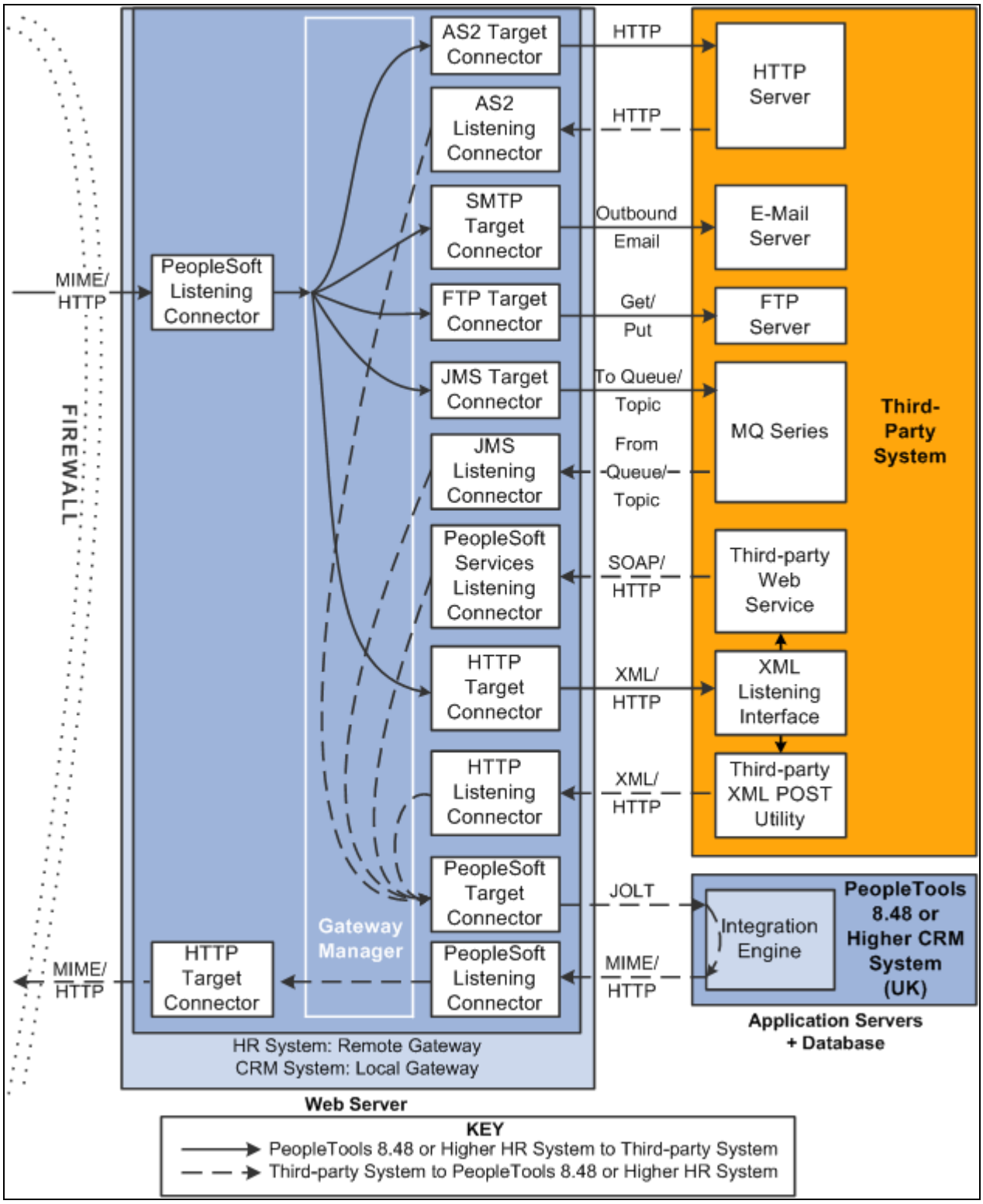
The following diagram provides a more detailed view of the configuration on the PeopleSoft system:



Configuration on the PeopleSoft system for integrations with third-party systems using remote gateways.

The following diagram provides a more detailed view of the configuration on the remote PeopleSoft system and the third-party system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Configuration on the remote PeopleSoft system and third-party system for integrations using remote gateways

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Sending Messages to Third-Party Systems

This section discusses how to configure:

- The PeopleSoft Human Resources system.
- The PeopleSoft Human Resources integration gateway.

The setup for this scenario is similar to the configuration for integrations with Integration Broker systems. The only difference is the target connector you use.

However, instead of using the PeopleSoft target connector for the remote node you must select the target connector based on the third-party system with which the PeopleSoft system is communicating. The target connector you select must reside on the remote gateway.

In the previous diagram, the PeopleSoft Human Resources system is the source system and the selected target connector is shown on the other side of the firewall, on the remote integration gateway.

As you review the configuration tasks for this scenario, keep in mind the following points:

- PeopleSoft recommends using a single gateway for all applications that reside on one side of a firewall.
- The local gateway for one PeopleSoft application can be the remote gateway for the other PeopleSoft application.

Configuring the PeopleSoft Human Resources System

On the PeopleSoft Human Resources system:

1. Define a local integration gateway.

   Use the Gateways component to define the local PeopleSoft Human Resources (USA) gateway.

2. Define a remote integration gateway on the local system.

   Use the Gateways component to define the gateway for the PeopleSoft Human Resources (USA) system (which is the PeopleSoft CRM [UK] gateway) and to specify the URL of the PeopleSoft CRM (UK) gateway.

3. Set up a local node.

   Use the Nodes component to set up the local node, which represents the PeopleSoft Human Resources (USA) system.

4. Set up a remote node.

   Use the Nodes component to set up the remote node, which represents the third-party system. When you define the remote node, use the Node Definitions-Connectors page to specify the gateway ID on the remote integration gateway. In addition, select the appropriate target connector, for example JMS target connector, SMTP target connector, POP 3 target connector, and so forth.

5. Create a service operation that includes an outbound routing definition where the sending node is the PeopleSoft HR system and the receiving node is the third-party system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the PeopleSoft Human Resources Integration Gateway

The only required integration gateway properties are the Oracle Jolt connect string properties that enable communication with integration engines on PeopleSoft Integration Broker systems. Set these properties in the integrationGateway.properties file.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Receiving Messages from Third-Party Systems

The previous diagram shows a second configuration scenario where a third-party system is performing an inbound HTTP Post to a PeopleSoft Human Resources/USA system via a UK gateway. In this scenario, the message goes through the PeopleSoft CRM/UK system only to get routing information, before it is sent to the remote integration gateway (the gateway on the USA side of the firewall. Therefore, in this scenario, the PeopleSoft CRM/UK system serves as a hub.

This section discusses how to configure:

- The PeopleSoft Human Resources (USA) system.
- The PeopleSoft Human Resources integration gateway.
- The PeopleSoft CRM (UK) system and hub.
- The PeopleSoft CRM (UK) integration gateway.
- The third-party system and PeopleSoft system.

Message Flow

In this scenario, a message originating from a third-party system is posted to the HTTP listening connector, JMS listening connector or a custom-built listening connector on the PeopleSoft CRM/UK integration gateway. Since the message does not contain the required routing information for the remote gateway, the listening connector hands it off to the PeopleSoft target connector. The PeopleSoft target connector sends the message to the default PeopleSoft node (the PeopleSoft CRM/UK) as determined by the default Jolt settings in the integrationGateway.properties file.

When the message reaches Integration Broker on the PeopleSoft CRM/UK system, the system applies transaction information to reroute the message to the remote gateway (the gateway on the USA side of the firewall), and thereby serves as a hub.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Message Processing on the Remote Gateway

Whenever you publish a message bound for a remote gateway, PeopleSoft Integration Broker reads it, determines that the target connector is not on its local gateway, places the remote gateway URL inside the IBInfo message wrapper and posts it to the PeopleSoft listening connector on the local gateway. The local gateway manager finds a remote gateway URL in the message wrapper and routes it to the remote gateway default connector, the HTTP target connector. The HTTP target connector on the local gateway then posts the message to the remote gateway URL (the PeopleSoft listening connector on the remote gateway) in MIME format, and removes the URL from the IBInfo message wrapper. On arrival at the remote gateway, the message is processed like any other incoming PeopleSoft message.

An exception to this message flow is if on the UK integration gateway side you created and loaded a custom listening connector that allows for the required routing information to be populated in the IBInfo message wrapper. The message would no longer need to be sent via the hub.

Keep in mind the following points:

- PeopleSoft recommends that you use a single gateway for all applications that reside on one side of a firewall.
- The local gateway for one PeopleSoft application can be a remote gateway for another PeopleSoft application.
- A message coming from a third-party system (local gateway or remote gateway) system can enter the integration gateway from any of the delivered listening connectors or from a custom-built listening connector. It cannot, however, use the PeopleSoft listening connector. PeopleSoft has designed the PeopleSoft listening connector to accept messages in the PeopleSoft internal message format only. Note that the diagram shows the message entering the integration gateway via the HTTP listening connector.

Configuring the PeopleSoft Human Resources (USA) System

On the PeopleSoft Human Resources system:

1. Set up a local node.

   Use the Node Definition component to set up the local node, which represents the Human Resources system.

2. Set up a remote node.

   The remote node that you set up represents the PeopleSoft CRM system. When you set up the remote node, specify the PeopleSoft target connector.

3. Create a service operation that contains an outbound routing definition where the sending node is the PeopleSoft CRM system and the receiving node is the PeopleSoft HR system.

Configuring the PeopleSoft Human Resources Integration Gateway

The only required integration gateway properties are the Oracle Jolt connect string properties that enable communication with the PeopleSoft Human Resources system. Set these properties in the integrationGateway.properties file.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the PeopleSoft CRM (UK) System and Hub

On the PeopleSoft CRM (UK) system:

- Define a local integration gateway.

  Use the Gateways component to set up the local gateway for the sending system.

- Define a remote integration gateway.

  The remote gateway for the PeopleSoft CRM (UK) system is the PeopleSoft Human Resources (USA) gateway.

- Set up a local node.

  Use the Node Definition component to set up the local node, which represents the PeopleSoft CRM system.

- Set up remote nodes.

  Use the Node Definition component to define two remote nodes: one remote node that represents the third-party system and another to represent the PeopleSoft Human Resources (USA) system. When you define the remote node that represents the third-party system, you specify the HTTP target connector, HTTPTARGET. When you define the remote node that represents the PeopleSoft Human Resources (USA) system, set it to use the PeopleSoft target connector on the remote (USA) gateway.

- Create a service operation that contains an outbound routing definition where the sending node is the third-party system and the receiving node is the PeopleSoft HR system.

- In the service operation definition that you created in the previous step, create an inbound routing definition if the third-party system will be sending you requests.

Configuring the PeopleSoft CRM (UK) Integration Gateway

The only required properties are the Oracle Jolt connect string properties that enable communication with integration engines on the PeopleSoft CRM systems. Set these properties in the integrationGateway.properties file.

Configuring the Third-Party System and PeopleSoft System

Because the PeopleSoft CRM (UK) system serves as a hub in this scenario, you must set up transactions and relationships for all messages from the third-party system that are routed through it. By using the systems in the diagram as an example, the following table shows the required node, transaction and relationship configurations:

Item to Configure	*PeopleSoft Human Resources System*	*PeopleSoft CRM System (Hub)*
Local nodes	Rename the default local node to represent the PeopleSoft Human Resources system.	Rename the default local node to represent the PeopleSoft CRM system.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Item to Configure	*PeopleSoft Human Resources System*	*PeopleSoft CRM System (Hub)*
Remote nodes	Define a remote node to represent the PeopleSoft CRM system.	Define remote nodes to represent the third-party system and the PeopleSoft Human Resources system.
Service operations and routing definitions	Define a service operation with an inbound routing definition where the PeopleSoft CRM system is the sending node and the PeopleSoft HR system is the receiving node.	Define a service operation with an outbound routing definition where the sending node is the third-party system and the receiving node is the PeopleSoft HR system.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

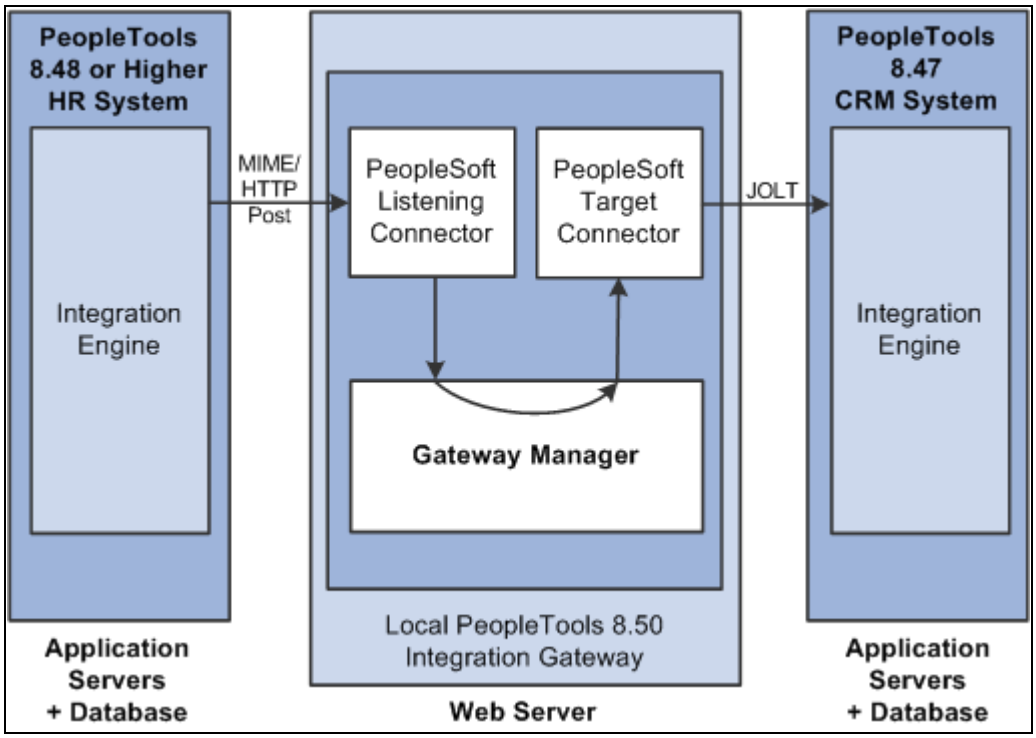
Integrating with PeopleTools 8.47 and Earlier PeopleTools 8.4x Systems

This section provides and overview of this scenario and discusses how to configure the system for this scenario.

Understanding This Scenario

This diagram shows the PeopleSoft Integration Broker components and configuration for communications between PeopleSoft Integration Broker system and a PeopleSoft system running on PeopleTools 8.47:

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



Integrations with PeopleTools 8.47 and earlier systems.

While this diagram depicts a PeopleSoft 8.47 system, the information provided in this section applies to any PeopleTools 8.47 or earlier system.

In this example, the PeopleSoft Human Resources is the sending system and the PeopleSoft CRM system is the receiving system.

The remainder of this section highlights the integration tasks by using the systems and components shown in the diagram as examples.

Configuring the System for This Scenario

This section discusses how to configure:

- The PeopleSoft Human Resources system.
- The PeopleSoft CRM system.

Configuring the PeopleSoft Human Resources System

In this scenario, the PeopleSoft Human Resource system is running on PeopleTools 8.52.

On the PeopleSoft Human Resources system:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

- Define a local integration gateway.

  Use the Gateways component to set up a local gateway for the PeopleSoft Human Resources system.

  The only required properties are the Oracle Jolt connect string properties that enable communication with the PeopleSoft Human Resources systems. Set these properties in the integrationGateway.properties file

- Set up a local node.

  Use the Node Definition component to set up the local node, which represents the PeopleSoft 8.52 Human Resources system.

- Set up a remote node.

  The remote node that you set up represents the PeopleSoft 8.47 or earlier Human Resources system. When you set up the remote node, specify the PeopleSoft target connector (PSFTTARGET) on the Connectors tab.

  Note. If you have upgraded from a PeopleSoft system, all nodes that existed for the system have been preserved as remote nodes in the PeopleSoft Integration Broker system. However, you must then associate each of these nodes to the PeopleSoft target connector for the remote note.

- Create a service operation with an inbound routing definition.

  Use the Service Operations component to create a service operation that contains an inbound routing definition where the receiving node is the PeopleSoft 8.52 system and the sending node is the PeopleSoft 8.47 or earlier system.

- Set up an outbound routing definition.

  In the service operation definition that you created in the previous step, create and outbound routing definition where the sending node is the PeopleSoft 8.52 system and the receiving node is the 8.47 or earlier system.

Configuring the PeopleSoft 8.47 or Earlier Human Resources System

In this scenario, the PeopleSoft CRM system is running on PeopleTools 8.47.

On the PeopleSoft CRM system:

1. Define the local integration gateway for the PeopleSoft CRM system by using the Gateways component. Specify the gateway's PeopleSoft listening connector as the gateway's URL.
2. Set up a local node.

   Use the Node Definition component to set up the local node, which represents the PeopleSoft CRM system.
3. Set up a remote node.

   The remote node that you set up represents the PeopleSoft 8.52 system. When you set up the remote node, specify the PeopleSoft target connector (PSFTTARGET) on the Connectors tab.

   Because the PeopleSoft Human Resources system uses PeopleSoft Integration Broker, specify the local gateway for the PeopleSoft CRM system and its PeopleSoft target connector on the Node Definitions - Connectors page.

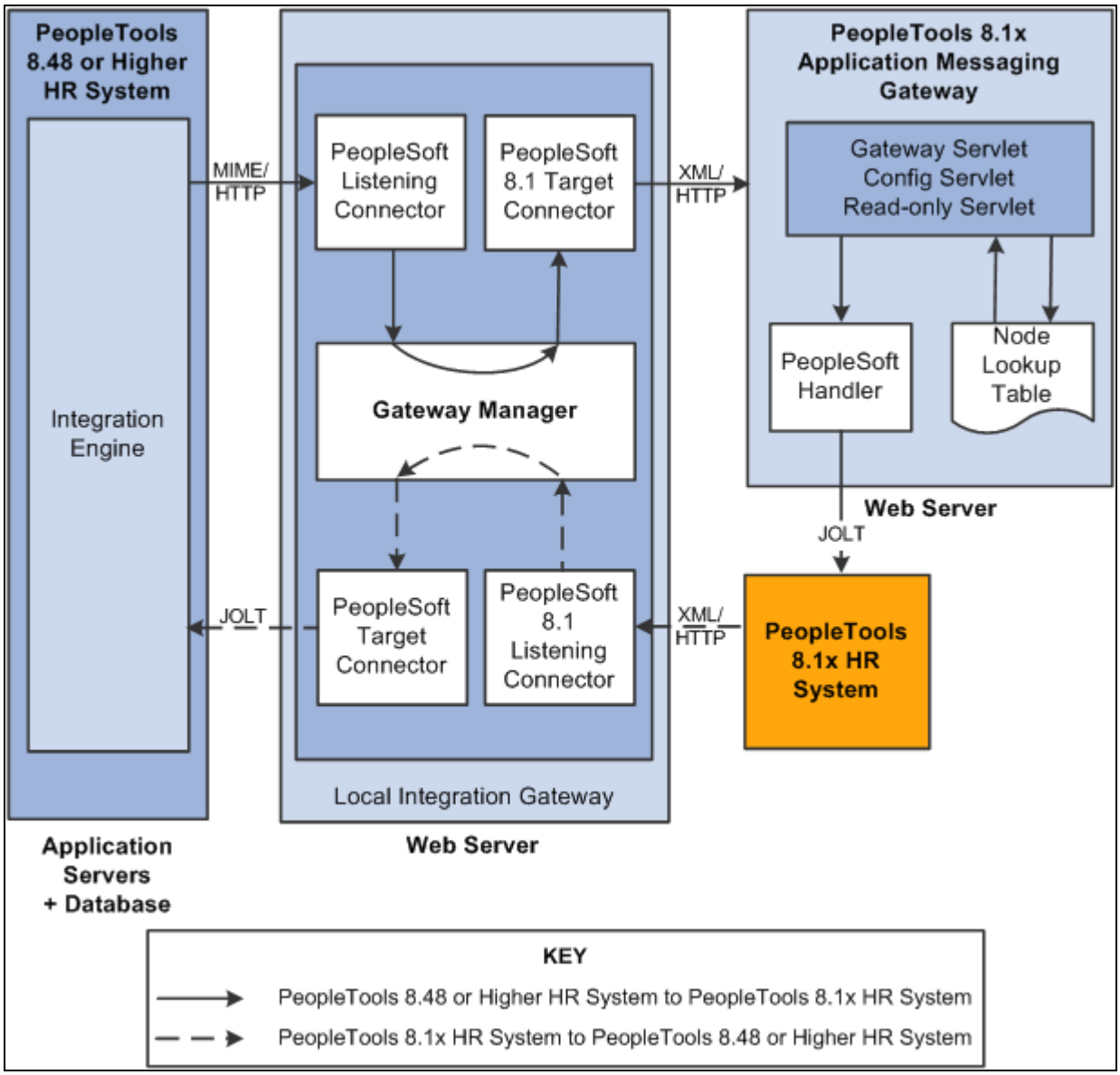
 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Integrating with PeopleTools 8.1x Systems

This section provides an overview of this scenario and discusses how to configure the system for this scenario.

Understanding This Scenario

This diagram shows the PeopleSoft Integration Broker components and configuration for communications between PeopleSoft Integration Broker systems and PeopleSoft systems running on PeopleTools 8.1x:



Integrations with PeopleSoft 8.1x systems

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

In this scenario, you must configure the PeopleSoft Integration Broker system, the integration gateway, and the PeopleTools 8.1x system. The remainder of this section highlights these tasks by using the systems and components shown in the diagram as examples.

Configuring the System for This Scenario

This section discusses how to configure:

- The PeopleSoft Human Resources system.
- The PeopleSoft Human Resources integration gateway.
- The PeopleTools 8.1x Human Resources system.

Configuring the PeopleSoft Human Resources System

On the PeopleSoft Human Resources system:

- Define a local integration gateway.

  Use the Gateways component to set up a local gateway for the PeopleSoft Human Resources system.

- Set up a local node.

  Use the Node Definition component to set up the local node, which represents the PeopleSoft Human Resources system.

- Set up a remote node.

  The remote node that you set up represents the PeopleSoft 8.1x Human Resources system. When you set up the remote node, specify the PeopleSoft 8.1 target connector (PSFT81TARGET) on the Connectors tab.

  Note. If you have upgraded from a PeopleSoft 8.1x system, all nodes that existed for the system have been preserved as remote nodes in the PeopleSoft Integration Broker system. However, you must then associate each of these nodes to the PeopleSoft 8.1 target connector.

- Create a service operation with an inbound routing definition.

  Use the Service Operations component to create a service operation that contains an inbound routing definition where the receiving node is the PeopleSoft 8.52 system and the sending node is the PeopleSoft 8.1x system.

- Set up an outbound routing definition.

  In the service operation definition that you created in the previous step, create and outbound routing definition where the sending node is the PeopleSoft 8.52 system and the receiving node is the 8.1x system.

Configuring the PeopleSoft Human Resources Integration Gateway

You must set integration gateway properties for the local gateway. The only required properties are the Oracle Jolt connect string properties that enable communication with the PeopleSoft Human Resources systems. Set these properties in the integrationGateway.properties file.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Configuring the PeopleSoft 8.1x Human Resources System

On the PeopleSoft 8.1x Human Resources system, locate the PeopleSoft 8.1x Human Resources message node and change the URL (location) to the PeopleSoft listening connector. The format is http://*webserver* /PSIGW.war/PS81ListeningConnector.

See Also

Appendix A, "Integration Scenarios," Understanding Integration Setup, page 597

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Managing Integration Gateways," Administering Integration Gateways

Chapter 10, "Managing Service Operations," Adding Service Operation Definitions, page 255

Chapter 10, "Managing Service Operations," Configuring Service Operation Definitions, page 255

PeopleTools 8.52: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Appendix B

Transformation Example: Integration Between Two PeopleSoft Nodes

This appendix discusses how to:

- Create message definitions.
- Set up codesets.
- Set up transformations.
- Walk through the generated XSL code.
- Test the transformation.

Understanding This Appendix

This appendix presents an in-depth example of how to use transformations to alter the messages sent between two systems.

Note. The code examples in this appendix are for illustrative purposes only and are not intended to be used in a production environment.

Using the Example

The purpose of this appendix is to present a more in depth example of how transformations can be used to alter the messages sent between two systems.

The following example describes an integration between a PeopleSoft Supply Chain Management node (PeopleSoft SCM) and a PeopleSoft Customer Relationship Management node (PeopleSoft CRM). This example demonstrates taking a PeopleSoft SCM purchase order message and transforming it into a similar PeopleSoft CRM purchase order message format.

Two aspects of transformations will be examined in this example:

- How XSL can be used to modify the structure of a message.
- How codesets can be used to map values between messages.

The example will focus on the PeopleSoft SCM system. This is the node where the XSL will be executed to transform the message from the SCM specified format to that of CRM. When the SCM system sends the message to CRM, the message will be in the CRM native format.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Integration Metadata for This Example

The example does not go into detail about setting up the all infrastructure necessary to actually send messages between the two nodes. It assumes that the systems have been configured and the service operations and services have been defined. Please refer to the chapters elsewhere in this PeopleBook for setting up integration metadata.

See Also

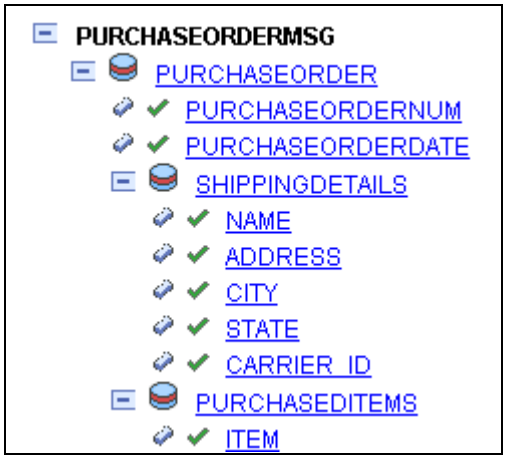
Chapter 4, "Understanding PeopleSoft Integration Broker Metadata," page 35

Creating Message Definitions

This section discusses the structure of the message definitions used in this example.

Message Definition: PeopleSoft SCM Node

The following example shows the format of the purchase order message on the PeopleSoft SCM node.



PURCHASEORDERMSG message definition.

The following is a sample message that corresponds to the message structure:

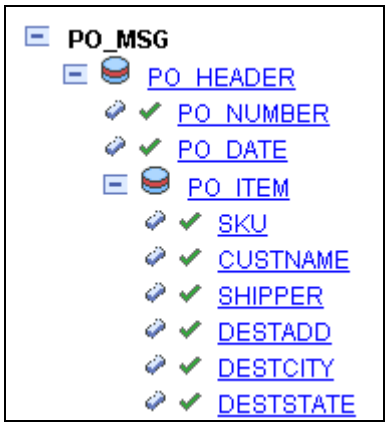
Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<PURCHASEORDERMSG>
  <FieldTypes>
    <PURCHASEORDER class="R">
      <PURCHASEORDERNUM type="CHAR"/>
      <PURCHASEORDERDATE type="DATE"/>
    </PURCHASEORDER>
    <SHIPPINGDETAILS class="R">
      <NAME type="CHAR"/>
      <ADDRESS type="CHAR"/>
      <CITY type="CHAR"/>
      <STATE type="CHAR"/>
      <CARRIER_ID type="CHAR"/>
    </SHIPPINGDETAILS>
    <PURCHASEDITEMS class="R">
      <ITEM type="CHAR"/>
    </PURCHASEDITEMS>
    <PSCAMA class="R">
      <LANGUAGE_CD type="CHAR"/>
      <AUDIT_ACTN type="CHAR"/>
      <BASE_LANGUAGE_CD type="CHAR"/>
      <MSG_SEQ_FLG type="CHAR"/>
      <PROCESS_INSTANCE type="NUMBER"/>
      <PUBLISH_RULE_ID type="CHAR"/>
      <MSGNODENAME type="CHAR"/>
    </PSCAMA>
  </FieldTypes>
  <MsgData>
    <Transaction>
      <PURCHASEORDER class="R">
        <PURCHASEORDERNUM IsChanged="Y">19908</PURCHASEORDERNUM>
        <PURCHASEORDERDATE IsChanged="Y">2006-04-03</PURCHASEORDERDATE>
        <SHIPPINGDETAILS class="R">
          <NAME IsChanged="Y">Smith,Bill</NAME>
          <ADDRESS IsChanged="Y">123 Anywhere St</ADDRESS>
          <CITY IsChanged="Y">Fresno</CITY>
          <STATE IsChanged="Y">CA</STATE>
          <CARRIER_ID IsChanged="Y">USPS</CARRIER_ID>
        </SHIPPINGDETAILS>
        <PURCHASEDITEMS class="R">
          <ITEM IsChanged="Y">AAS5536</ITEM>
        </PURCHASEDITEMS>
        <PURCHASEDITEMS class="R">
          <ITEM IsChanged="Y">POB332Q</ITEM>
        </PURCHASEDITEMS>
      </PURCHASEORDER>
      <PSCAMA class="R">
        <LANGUAGE_CD>ENG</LANGUAGE_CD>
        <AUDIT_ACTN/>
        <BASE_LANGUAGE_CD>ENG</BASE_LANGUAGE_CD>
        <MSG_SEQ_FLG/>
        <PROCESS_INSTANCE>0</PROCESS_INSTANCE>
        <PUBLISH_RULE_ID/>
        <MSGNODENAME/>
      </PSCAMA>
    </Transaction>
  </MsgData>
</PURCHASEORDERMSG>
```

Message Definition: PeopleSoft CRM Node

The following example shows the format of the purchase order on the PeopleSoft CRM node.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



PO_MSG message definition.

This is a sample message that corresponds to the message structure:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>
<PO_MSG>
  <FieldTypes>
    <PO_HEADER class="R">
      <PO_NUMBER type="CHAR"/>
      <PO_DATE type="DATE"/>
    </PO_HEADER>
    <PO_ITEM class="R">
      <SKU type="CHAR"/>
      <CUSTNAME type="CHAR"/>
      <SHIPPER type="CHAR"/>
      <DESTADD type="CHAR"/>
      <DESTCITY type="CHAR"/>
      <DESTSTATE type="CHAR"/>
    </PO_ITEM>
    <PSCAMA class="R">
      <LANGUAGE_CD type="CHAR"/>
      <AUDIT_ACTN type="CHAR"/>
      <BASE_LANGUAGE_CD type="CHAR"/>
      <MSG_SEQ_FLG type="CHAR"/>
      <PROCESS_INSTANCE type="NUMBER"/>
      <PUBLISH_RULE_ID type="CHAR"/>
      <MSGNODENAME type="CHAR"/>
    </PSCAMA>
  </FieldTypes>
  <MsgData>
    <Transaction>
      <PO_HEADER class="R">
        <PO_NUMBER IsChanged="Y">BBN7782</PO_NUMBER>
        <PO_DATE IsChanged="Y">2006-04-15</PO_DATE>
        <PO_ITEM class="R">
          <SKU IsChanged="Y">JN557BB</SKU>
          <CUSTNAME IsChanged="Y">Jones,Mark</CUSTNAME>
          <SHIPPER IsChanged="Y">Federal Express</SHIPPER>
          <DESTADD IsChanged="Y">66  Availer St</DESTADD>
          <DESTCITY IsChanged="Y">Stockton</DESTCITY>
          <DESTSTATE IsChanged="Y">CA</DESTSTATE>
        </PO_ITEM>
      </PO_HEADER>
      <PSCAMA class="R">
        <LANGUAGE_CD>ENG</LANGUAGE_CD>
        <AUDIT_ACTN/>
        <BASE_LANGUAGE_CD>ENG</BASE_LANGUAGE_CD>
        <MSG_SEQ_FLG/>
        <PROCESS_INSTANCE>0</PROCESS_INSTANCE>
        <PUBLISH_RULE_ID/>
        <MSGNODENAME/>
      </PSCAMA>
    </Transaction>
  </MsgData>
</PO_MSG>
```

Setting Up the Codesets

Both the PeopleSoft SCM and PeopleSoft CRM messages contain an entry to capture the shipping agent to be used for the purchase order. For the PeopleSoft SCM message, the element used is CARRIER_ID; for the PeopleSoft CRM message it is SHIPPER. For the purposes of this example, the assumption is that the values for these two elements map differently, according to the following table:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

CARRIER_ID values on SCM System	*SHIPPER Values on CRM System*
FEDEX	Federal Express
UPS	United Parcel Service
USPS	United States Postal Service

An SCM message containing the following XML:

```
<CARRIER_ID>UPS</CARRIER_ID>
```

would logically map to

```
<SHIPPER>United Parcel Service</SHIPPER>
```

in the corresponding CRM message.

In order to enable such mapping in this example, you must create codeset metadata.

Create codeset metadata:

1. Create a codeset group called *SCM_GROUP*. Add the following entries:

Match Name	*Match Value*
CARRIER_ID	FEDEX
CARRIER_ID	UPS
CARRIER_ID	USPS
CARRIER_ID	Blank

   Note that the final match value entry is blank: this will be used for the default value.

2. Create a codeset called *SCM_CODESET*, using the codeset group defined in the prior step. Add the following entries to this codeset:

Match Name	*Match Value*
CARRIER_ID	FEDEX
CARRIER_ID	UPS
CARRIER_ID	USPS
CARRIER_ID	Blank

3. Create a codeset group called *CRM_GROUP*. There is no need to define any entries for this group.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. Add codeset values from the *SCM_GROUP* to the *CRM_GROUP*, using the *SCM_CODESET*. Four entries will need to be defined:
    - Select *CARRIER_ID* and *<blank>*, and add a return name of *SHIPPER* and the value *Unknown.*
    - Select *CARRIER_ID* and *FEDEX*, and add a return name of *SHIPPER* and the value *Federal Express.*
    - Select *CARRIER_ID* and *UPS*, and add a return name of *SHIPPER* and the value *United Parcel Service.*
    - Select *CARRIER_ID* and *USPS*, and add a return name of *SHIPPER* and the value *United States Postal Service.*
5. Go to the node definition for the SCM node and add the codeset group *SCM_CODESET* to the node.
6. Go to the node definition for the CRM node and add the codeset group *CRM_CODESET* to the node.

Setting Up the Transformation

The PeopleSoft SCM and PeopleSoft CRM purchase order messages share what are essentially the same first two elements: a purchase order number and the purchase order date. However, from there the structures differ. The PeopleSoft SCM message has a small section containing shipping information, followed by the list of items purchased. The PeopleSoft CRM message has the shipping information merged with the list of items purchased.

The following XSL transforms the PeopleSoft SCM purchase order message into the PeopleSoft CRM format:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">


  <xsl:template match="PURCHASEORDERMSG">
    <PO_MSG>
      <xsl:apply-templates select="FieldTypes"/>
      <xsl:apply-templates select="MsgData"/>
    </PO_MSG>
  </xsl:template>

  <xsl:template match="MsgData">
    <MsgData>
      <xsl:apply-templates select="Transaction"/>
    </MsgData>
  </xsl:template>

  <xsl:template match="Transaction">
    <Transaction>
      <xsl:apply-templates select="PURCHASEORDER"/>
      <PSCAMA class="R">
        <LANGUAGE_CD>ENG</LANGUAGE_CD>
        <AUDIT_ACTN/>
        <BASE_LANGUAGE_CD>ENG</BASE_LANGUAGE_CD>
        <MSG_SEQ_FLG/>
        <PROCESS_INSTANCE>0</PROCESS_INSTANCE>
        <PUBLISH_RULE_ID/>
        <MSGNODENAME/>
      </PSCAMA>
    </Transaction>
  </xsl:template>

  <xsl:template match="PURCHASEORDER">
    <PO_HEADER class="R">
      <PO_NUMBER IsChanged="Y"><xsl:value-of select="PURCHASEORDERNUM"/>
      </PO_NUMBER>
      <PO_DATE IsChanged="Y"><xsl:value-of select="PURCHASEORDERDATE"/>
      </PO_DATE>
      <xsl:apply-templates select="PURCHASEDITEMS"/>
    </PO_HEADER>
  </xsl:template>

  <xsl:template match="PURCHASEDITEMS">
    <PO_ITEM class="R">
      <SKU IsChanged="Y"><xsl:value-of select="ITEM"/></SKU>
      <CUSTNAME IsChanged="Y">
        <xsl:value-of select="../SHIPPINGDETAILS/NAME"/>
      </CUSTNAME>
      <SHIPPER IsChanged="Y">
         <psft_function name="codeset" codesetname="SCM_CODESET">
            <parm name="CARRIER_ID">
               <xsl:value-of select="../SHIPPINGDETAILS/CARRIER_ID"/>
            </parm>
            <value name="SHIPPER" select="."/>
         </psft_function>
   </SHIPPER>
      <DESTADD IsChanged="Y">
        <xsl:value-of select="../SHIPPINGDETAILS/ADDRESS"/>
      </DESTADD>
      <DESTCITY IsChanged="Y">
        <xsl:value-of select="../SHIPPINGDETAILS/CITY"/>
      </DESTCITY>
```


Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

```
      <DESTSTATE IsChanged="Y">
        <xsl:value-of select="../SHIPPINGDETAILS/STATE"/>
      </DESTSTATE>
    </PO_ITEM>
  </xsl:template>

  <xsl:template match="FieldTypes">
  <FieldTypes>
    <PO_HEADER class="R">
      <PO_NUMBER type="CHAR"/>
      <PO_DATE type="DATE"/>
    </PO_HEADER>
    <PO_ITEM class="R">
      <SKU type="CHAR"/>
      <CUSTNAME type="CHAR"/>
      <SHIPPER type="CHAR"/>
      <DESTADD type="CHAR"/>
      <DESTCITY type="CHAR"/>
      <DESTSTATE type="CHAR"/>
    </PO_ITEM>
    <PSCAMA class="R">
      <LANGUAGE_CD type="CHAR"/>
      <AUDIT_ACTN type="CHAR"/>
      <BASE_LANGUAGE_CD type="CHAR"/>
      <MSG_SEQ_FLG type="CHAR"/>
      <PROCESS_INSTANCE type="NUMBER"/>
      <PUBLISH_RULE_ID type="CHAR"/>
      <MSGNODENAME type="CHAR"/>
    </PSCAMA>
  </FieldTypes>
  </xsl:template>
</xsl:stylesheet>
```

Note that this XSL contains a reference to the psft_function, which will resolve codeset mapping after the transform has been run.

This XSL should be placed into an Application Engine program, and this program associated with the routing definitions for the service operation.

XSL Walkthrough

This section discusses the process flow through the XSL when the PeopleSoft SCM purchase order message is transformed.

XSL is composed of templates. An XSLT template is an instruction of what to do when a particular XML node or condition is encountered. During a transform, the XSL processing engine starts at the root element of the XML message and then attempts to find matching templates in the XSL. When a matching template is found, the instructions in that template are used in the building of the output XML document.

The processing of a transform is actually a two pass process. In the first pass, the XSL is executed against the input XML, and an output XML document is generated. In the second pass, the psft_function calls are resolved and the codeset values are placed into the document.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Transformation Processing: First Pass

This section discusses the first pass of transform processing, during which the XSL code runs against the input XML and the system generates an output XML document. The steps listed below mimic the actions taken by the transformation engine in processing the input XML.

1. The first element in the input message is *PURCHASEORDERMESSAGE*. The transformation engine finds the following matching template in the XSL:

```
<xsl:template match="PURCHASEORDERMSG">
  <PO_MSG>
    <xsl:apply-templates select="FieldTypes"/>
    <xsl:apply-templates select="MsgData"/>
  </PO_MSG>
</xsl:template>
```

   Here we see the root element of the output document being created. This template instructs the transformation engine to:

   - Output the start tag for *PO_MSG*.

     It is important to note that template processing is a recursive process. In the sequence above, the transformation engine outputs the start tag, then applies templates to all FieldType nodes. This is essentially a callout to the template that handles those nodes (and potentially any children). The output of this call is then placed into the output document. Then the transformation engine selects all the MsgData nodes, and applies templates to them. Again, the output from the processing of those nodes is then place into the output document. Finally, the closing PO_MSG tag is written into the output, and the first pass is finished. Of course, documenting a recursive process is not always straightforward. In this example bear in mind that while it is presented as a numbered sequence of steps, the actual process is not sequential.

   - Select the FieldTypes nodes under the current node in the input document, and process them.

   - Select the MsgData nodes under the current node in the input document, and process them.

   - Output the end tag for PO_MSG.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

2. The transform engine selects the FieldTypes node, and finds the following template:

```
<xsl:template match="FieldTypes">
<FieldTypes>
  <PO_HEADER class="R">
    <PO_NUMBER type="CHAR"/>
    <PO_DATE type="DATE"/>
  </PO_HEADER>
  <PO_ITEM class="R">
    <SKU type="CHAR"/>
    <CUSTNAME type="CHAR"/>
    <SHIPPER type="CHAR"/>
    <DESTADD type="CHAR"/>
    <DESTCITY type="CHAR"/>
    <DESTSTATE type="CHAR"/>
  </PO_ITEM>
  <PSCAMA class="R">
    <LANGUAGE_CD type="CHAR"/>
    <AUDIT_ACTN type="CHAR"/>
    <BASE_LANGUAGE_CD type="CHAR"/>
    <MSG_SEQ_FLG type="CHAR"/>
    <PROCESS_INSTANCE type="NUMBER"/>
    <PUBLISH_RULE_ID type="CHAR"/>
    <MSGNODENAME type="CHAR"/>
  </PSCAMA>
</FieldTypes>
</xsl:template>
```

   The interesting thing about this template is that it is basically an instruction to insert static text into the output document. Of course, this makes sense, as the FieldTypes section is dependant upon the message structure, and not the actual data contained within any particular message instance. Also note that there is no further node selection in this template, so after the XML is emitted this particular path through the XML ends.

3. Now that the FieldTypes nodes have been processed, the transform engine processes the MsgData node using the following matching template:

```
<xsl:template match="MsgData">
  <MsgData>
    <xsl:apply-templates select="Transaction"/>
  </MsgData>
</xsl:template>
```

   The template is quite simple. The transformation engine is to put a starting MsgData node in the output document, and then process the Transaction nodes in the input document. Note that the node context has changed: the current node in the input document being processed is the MsgData node. The call to select then implies that all child Transaction nodes under the MsgData are to be selected.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. The Transaction nodes under MsgData are matched by the following template:

```
<xsl:template match="Transaction">
  <Transaction>
    <xsl:apply-templates select="PURCHASEORDER"/>
    <PSCAMA class="R">
      <LANGUAGE_CD>ENG</LANGUAGE_CD>
      <AUDIT_ACTN/>
      <BASE_LANGUAGE_CD>ENG</BASE_LANGUAGE_CD>
      <MSG_SEQ_FLG/>
      <PROCESS_INSTANCE>0</PROCESS_INSTANCE>
      <PUBLISH_RULE_ID/>
      <MSGNODENAME/>
    </PSCAMA>
  </Transaction>
</xsl:template>
```

   A Transaction start tag is written to the output document, and then the PURCHASEORDER nodes are to be handled. Once these nodes have been processed, the PSCAMA information will be written out, along with the closing Transaction tag.

5. The call to handle the PURCHASEORDER node invokes:

```
<xsl:template match="PURCHASEORDER">
  <PO_HEADER class="R">
    <PO_NUMBER IsChanged="Y">
       <xsl:value-of select="PURCHASEORDERNUM"/>
    </PO_NUMBER>
    <PO_DATE IsChanged="Y">
       <xsl:value-of select="PURCHASEORDERDATE"/>
    </PO_DATE>
    <xsl:apply-templates select="PURCHASEDITEMS"/>
  </PO_HEADER>
</xsl:template>
```

   The PO_HEADER start tag is emitted as well as the child PO_NUMBER and PO_DATE elements. The call out to xs:value-of means that node values from the input message are copied to the output message. In this case, the node PO_NUMBER in the output message is given the value from PURCHASEORDERNUM in the input message. PO_DATE is given the value from PURCHASEORDERDATE. Once these two elements have been written out, the transformation engine is then told to process the PURCHASEDITEMS nodes in the input document.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

6. Each PURCHASEDITEMS node in the input message causes the following template to be executed:

```
<xsl:template match="PURCHASEDITEMS">
  <PO_ITEM class="R">
    <SKU IsChanged="Y"><xsl:value-of select="ITEM"/></SKU>
    <CUSTNAME IsChanged="Y">
       <xsl:value-of select="../SHIPPINGDETAILS/NAME"/>
    </CUSTNAME>
    <SHIPPER IsChanged="Y">
       <psft_function name="codeset" codesetname="SCM_CODESET">
          <parm name="CARRIER_ID">
            <xsl:value-of select="../SHIPPINGDETAILS/CARRIER_ID"/>
          </parm>
          <value name="SHIPPER" select="."/>
       </psft_function>
     </SHIPPER>
    <DESTADD IsChanged="Y">
       <xsl:value-of select="../SHIPPINGDETAILS/ADDRESS"/>
    </DESTADD>
    <DESTCITY IsChanged="Y">
       <xsl:value-of select="../SHIPPINGDETAILS/CITY"/>
    </DESTCITY>
    <DESTSTATE IsChanged="Y">
       <xsl:value-of select="../SHIPPINGDETAILS/STATE"/>
     </DESTSTATE>
  </PO_ITEM>
</xsl:template>
```

   This is where the bulk of the building of the output message is performed. For each PURCHASEDITEMS node in the input document, this template will be run once. The template is responsible for building out the PO_ITEM element and children in the output message. As in the template for PURCHASEORDER, this template uses the values from the input message and copies them across to the output message. For example, the SKU element in the output is given the value from the ITEM node in the input. Also note that the SHIPPER node contains a reference to psft-function and the SCM_CODESET. At this stage in the transformation, this text is static except for the value-of call to "../SHIPPINGDETAILS/CARRIER_ID", which will be resolved to "USPS". The actual codeset lookup will not be done at this point; this will occur in the second pass.

7. After the PURCHASEDITEMS template completes, the transformation jumps back to step 5, and outputs the closing PO_HEADER template.

8. Once the PURCHASEORDER template in step 5 completes, the transformation jumps back to step 4, and the Transaction template is completed. At this point the PSCAMA section in the template is written to the output message.

9. After the Transaction template in step 4 completes, the transformation jumps back to step 3, and the MsgData template. At this point the closing MsgData tag is written to the output message.

10. After the MsgData template in step 3 completes, the transformation jumps back to step 1. At this point the closing PO_MSG tag is written out, and the first part of the transformation process ends.

Transformation Processing: Second Pass

This section discusses the second pass of transform processing, during which the psft_function calls are resolved and the codeset values are placed into the document.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Assuming that no errors were encountered in the processing in the first pass a complete XML document will be available, containing the formatted PeopleSoft CRM purchase order message. However, this XML will still contain the reference:

```
<SHIPPER IsChanged="Y">
   <psft_function name="codeset" codesetname="SCM_CODESET">
      <parm name="CARRIER_ID">USPS</parm>
      <value name="SHIPPER" select="."/>
   </psft_function>
</SHIPPER>
```

The second pass walks through the message and resolves all calls to psft_function. In this instance the codeset lookup will be run, and the psft_function node will be replaced with the result. The XML in the output message will then look like:

```
<SHIPPER IsChanged="Y">
   United States Postal Service
</SHIPPER>
```

After the second pass completes, the transform is complete.

Testing the Transformation

The Transform Test utility is very useful when creating message transforms, since it allows one to exercise the transform without actually having service operations being invoked.

This section discusses using the utility to verify the example in this appendix.

To test the transformation:

1. Save to file the example message XML from the Message Definition: PeopleSoft SCM Node section presented earlier in this appendix.
2. Put the XSL from the Setting Up The Transformation section of this appendix in an application engine program.
3. Access the Transformation Test utility.

   Select PeopleTools, Integration Broker, Service Utilities, Transformation Test.

   The Transformation Test Utility page appears.
4. Add a new project.
5. In the *Program Name* field, enter in the name of the application engine program containing the XSL.
6. In the *Source Node Name* field, enter in the node name for the PeopleSoft SCM node.
7. In the *Destination Node Name* field enter in the node name for the CRM node.
8. In the *File Name* field, enter in the fully qualified path name of the file containing the input message.
9. Click the *Transform* button.

   The transformed XML appears in the Message Text text box.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Appendix C

Understanding Migrated Integration Metadata

This appendix discusses:

- Migrated integration metadata.
- Migrated integration PeopleCode.
- Correcting integration PeopleCode that did not successfully migrate.

Understanding Migrated Integration Metadata

The following table summarizes how objects are migrated between PeopleTools 8.47 and earlier releases and PeopleTools 8.48 and higher releases:

PeopleTools 8.4x Objects (PeopleTools 8.47 and Earlier)	***PeopleTools 8.48 and Higher Objects***
Node.	Node.
Channel.	Queue.
Message.	Message.
Node transactions and relationships.	Service, Service Operations, Service Operation Versions, and Routings.
Message and Subscription PeopleCode.	Application classes and service operation handlers.

Note. All objects migrated to PeopleTools 8.48 and higher releases have the Owner ID of the message from which they were created. Any invalid Owner IDs are deleted at the time of upgrade. If no Owner ID exists for an object, you must manually define one in the PeopleTools 8.48 or higher database.

Node Objects

Upgraded node objects are assigned a Default User ID as defined in the upgrade script.

In PeopleTools 8.48 and higher releases security is implemented at the user ID level. The default user ID is used in conjunction with securing service operations.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Channel Objects

Channels have been converted to queues.

The new queue name should be the same as the old channel name.

As part of the upgrade/conversion, any related language data associated with the channel should also be brought over to the newly created queue definition.

Message Objects

Messages are shapes that provide the physical description of the contents of a service operation transaction, and describe the data being sent, including fields, field types, and field lengths.

Unlike PeopleTools 8.47 and earlier releases, beginning with PeopleTools 8.48 messages do not contain any processing logic, such as PeopleCode events or subscriptions. All processing logic is created by extending a set of delivered application classes, and associating those application classes to service operations using service operation handlers.

Node Transaction and Relationship Objects

Node transactions and relationships are migrated to services, service operations and routing definitions.

Service Objects

During the upgrade process, a service definition is created for each distinct message referenced in the node transaction table.

A service inherit most of its properties from the message, including description, long description, Owner ID, language code, and so on.

The service name in PeopleTools 8.48 and higher releases is the same as the message name in the PeopleTools 8.47 or earlier system.

Any related language data associated with the message is inherited by the service.

Service Operation Objects

Service operations have the same name as the service to which they are associated.

Complex transactions like asynchronous or synchronous transactions with transformations, hub transactions or async-to-sync are defined by grouping a set of node transactions and relationships together. As the complex cases are upgraded, the system separates node transactions that were created for these complex cases (for example an asynchronous hub case) from the simple cases (for example, outbound asynchronous requests with no transformation). Asynchronous and synchronous node transactions that do not participate in a relationship (for example, those that are left over after we settled the complex cases) in PeopleTools 8.47 and earlier releases become service operations in PeopleTools 8.48 and higher releases.

Warning! If there is no node transaction on the PeopleTools 8.47 or earlier system, no service operation is created on the PeopleTools 8.48 or higher system during upgrade.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

In cases where a PeopleTools 8.47 or earlier node has both synchronous and asynchronous transactions, the first transaction defined on the node is migrated as a service operation; the second message cannot be created and an error is generated to the output log at the time of upgrade.

For asynchronous-to-synchronous transactions on a PeopleTools 8.47 or earlier system, the following occurs during the upgrade process to PeopleTools 8.48 or higher system:

- The service operation is named after the outbound message name.
- The request message is named after the outbound asynchronous message.
- The response message is named after the asynchronous response message specified in the relationship.

In PeopleTools 8.47 and earlier systems the IsActive property of the Message class was used to check the active property on the message. In PeopleTools 8.48 and higher systems it checks the service operation. In cases where a service operation is not created for a message, the IsActive property returns *False.* Therefore there may be a behavioral change from previous releases. In cases where there is no service operation created, and you want the previous behavior preserved, you must create a service operation and set the operation state to match that which was on the message.

Routing Objects

The upgrade process creates point-to-point routing definitions in the PeopleTools 8.48 or higher system based on node transactions and relationships defined in the PeopleTools 8.47 or earlier system.

Only current relationships in the PeopleTools 8.47 or earlier system are migrated.

The Active/Inactive flag on the transaction in the PeopleTools 8.47 or earlier system determines whether the routing definition is active or inactive in the PeopleTools 8.48 or later system.

The connector information defined on the outbound transaction is used in the routing definition. You must manually update aliases on routing definitions that use custom connectors.

When the system creates routing definitions during the upgrade process, the external message name from the transaction is used as the routing alias, if one was defined. If there is no value in the external message name, messages on nodes that are marked as *External* use the PeopleTools 8.47 or earlier system alias message name. PeopleTools 8.47 and earlier system nodes marked as anything other then *External* use the PeopleTools 8.48 alias format of *serviceoperation.version.*

All routing definitions created during the upgrade process are point-to-point routings, with one exception. In cases where on the PeopleTools 8.47 or earlier system there is an inbound synchronous or asynchronous node transaction on the default local node without a corresponding outbound synchronous or asynchronous node transaction (via relationship) on the default local node, an any-to-local routing definition is created during the upgrade.

Understanding Migrated Integration PeopleCode

The following table summarized how PeopleCode has been migrated between PeopleTools 8.4x releases and PeopleTools 8.48:

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleTools 8.4x PeopleCode (PeopleTools 8.47 and Earlier)	*PeopleTools 8.48 and Higher PeopleCode*
Message and subscription PeopleCode.	Application classes.
PeopleCode events.	Service operation handlers.

All PeopleCode referenced in a message is converted to an application class, which is in turn then referenced by a handler that is created in the service operation generated by the converted message.

Application Classes

PeopleCode defined on messages in PeopleTools 8.47 and earlier releases is migrated into application classes in PeopleTools 8.48 and higher releases.

Application classes have to belong to an application package. The message name becomes the application package name and description. The exception for this naming convention is when an application package already exists on the PeopleTools 8.48 or higher system that matches the message name. In this case the application package is stripped of all underscores and the number *1* is appended to the end of the name.

For every subscription event associated with a message, an application class is created in the PeopleTools 8.48 or higher system with a similar name as the subscription event. Since application class names cannot contain special characters, those found in the subscription event name are simply replaced with an underscore.

Default values for application class names are used for the other message events.

If a PeopleTools 8.47 or earlier message has no PeopleCode defined on it, no application package is generated for it in the PeopleTools 8.48 or higher system.

Application classes that fail to compile are saved and commented out. In these cases, a deprecated handler is created to invoke the old message or subscription event, to behave, runtime-wise, as it did in PeopleTools 8.47 and earlier systems. The user, however, is responsible for correcting any application classes that failed to compile and modifying any service operation definition that makes use of the deprecated handler.

See Appendix C, "Understanding Migrated Integration Metadata," Correcting Integration PeopleCode That Did Not Migrate, page 651.

PeopleCode Methods

The following table describes how PeopleCode events from PeopleTools 8.47 and earlier releases map to PeopleTools 8.48 and higher methods.

PeopleTools 8.4x Integration PeopleCode Events (PeopleTools 8.47 and Earlier)	*PeopleTools 8.48 Integration PeopleCode Base Classes and Methods (PeopleTools 8.48 and Higher)*
OnRequest	IRequestHandler:OnRequest
OnAckReceive	IReceiver:OnAckReceive

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

PeopleTools 8.4x Integration PeopleCode Events (PeopleTools 8.47 and Earlier)	***PeopleTools 8.48 Integration PeopleCode Base Classes and Methods (PeopleTools 8.48 and Higher)***
OnRouteReceive	IRouter: OnRouteReceive
OnRouteSend	IRouter:OnRouteSend
OnSend	ISend:OnRequestSend
Subscription	INotification:OnNotify

Built-In Functions

Many PeopleCode built-in functions have been deprecated for the new PeopleSoft Integration Broker model. Most of the PeopleTools 8.47 and earlier built-in functions have been internally redirected to work with the PeopleTools 8.48 and later equivalent. When compiling PeopleCode that uses the PeopleTools 8.47 and earlier built-in functions, an informational message appears that explains that the given built-in has been deprecated.

Other Migrated Constructs

The following constructs are also migrated during the upgrade process:

- Import statements.
- Function libraries.
- Variables not explicitly declared.

Special Characters

During the upgrade process, any special characters that used in the name of PeopleCode constructs in the PeopleTools 8.47 or earlier PeopleCode, such as slashes ("/"), single quotation marks " ' ' ", and periods (.) are replaced with underscores (_).

Correcting Integration PeopleCode That Did Not Migrate

This section provides an overview of integration PeopleCode that did not migrate and how to correct it.

Understanding Integration PeopleCode That Did Not Migrate

This section discusses the reasons why integration events and subscriptions might not successfully migrate to application classes during the upgrade from PeopleTools 8.47 and earlier systems to PeopleTools 8.48 and higher systems and the *Deprecated* type service operation handler used to manage such PeopleCode.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Reasons Why Integration PeopleCode Does Not Migrate

Integration events and subscriptions might not successfully migrate to application classes for the following reasons:

- Use of global variables.
- Use of component variables.
- Use of constants.
- Use of local functions.
- Inability to determine the return type or the return type is incorrect.

Deprecated PeopleCode Handler

The PeopleSoft system creates a deprecated PeopleCode handler for any integration PeopleCode that cannot be migrated to the PeopleTools 8.48 or higher system.

Deprecated handlers enable you to run PeopleTools 8.47 and earlier PeopleCode (subscription or OnRequest). However, PeopleSoft recommends that you correct the PeopleCode and migrate the code into an *Application Class* type handler for use in the PeopleTools 8.48 or higher system.

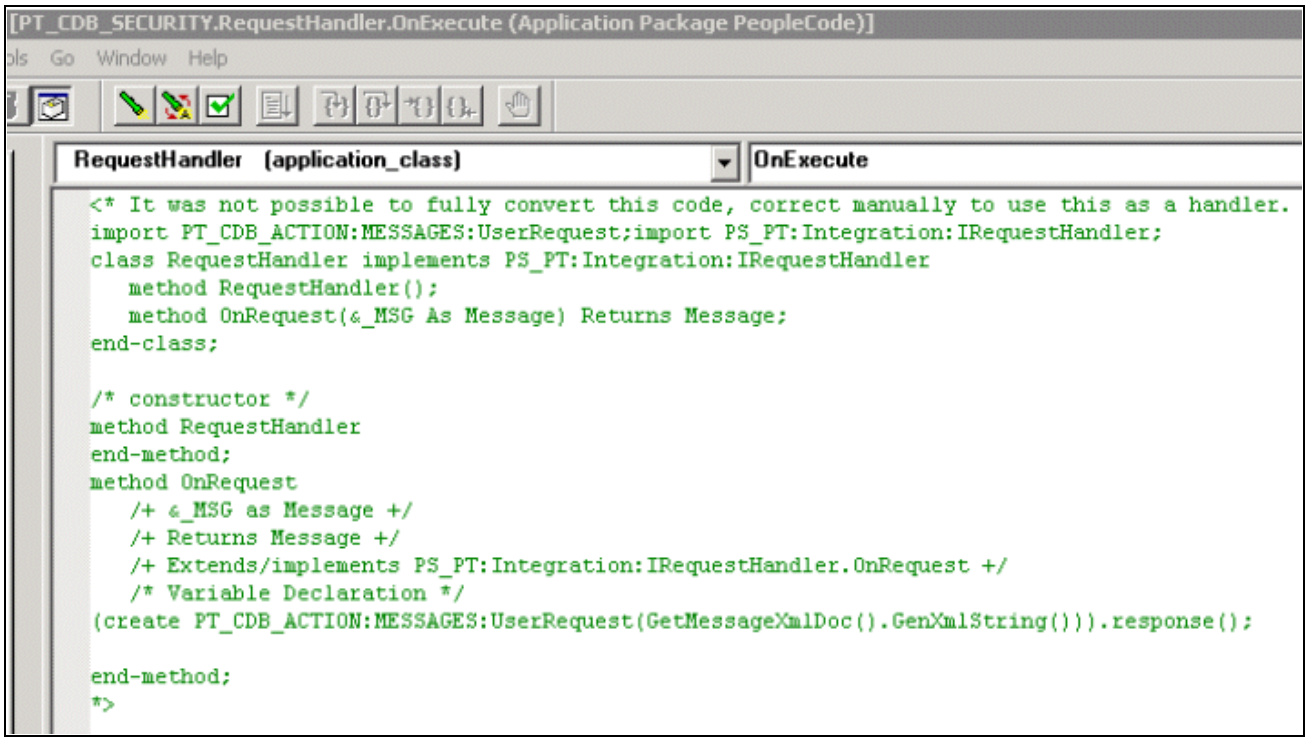
See Appendix C, "Understanding Migrated Integration Metadata," Correcting Non-Migrated Integration PeopleCode, page 652.

Correcting Non-Migrated Integration PeopleCode

This section discusses correcting non-migrated PeopleCode and creating an *Application Class* type handler.

The following example shows an application class created for the PeopleTools 8.47 or earlier message *PT_CDB_SECURITY*. During the upgrade process, the system was unable to migrate the integration PeopleCode defined on the message.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.



A PeopleTools 8.48 application class created for the PeopleTools 8.4x message PT_CDB_SECURITY

To correct non-migrated integration PeopleCode:

1. In PeopleSoft Application Designer, open the migrated application package.

   The package name is generally the same as the message name in the PeopleTools 8.47 or earlier system.

2. Remove the "<*" and "*>" symbols, as well as the comments from the code.

3. Compile the code.

   If the code did not migrate for one of the following reasons, the system displays a meaningful error message to assist you in correcting the problem:

   - Use of component scope variables.
   - Use of global variables or constant variables
   - Incorrect return type for handler.

   If the reason for the failure is due to the use of a local function, modify the PeopleCode to ensure that the function is passing appropriate types for the context in the application package.

4. Correct the PeopleCode and save the changes.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

5. Delete the deprecated handler.

    a. Select PeopleTools, Integration Broker, Service Utilities, Service Administration.. Click the Deprecated PeopleCode tab.

    The Deprecated PeopleCode page appears.

    b. In the Service Operation field, enter the name of the service operation that contains the deprecated handler to delete and click the Search button.

    c. In the Select column, check the box next to the service operation name that contains the deprecated handler to delete.

    d. Click the Delete button.

6. Add an *Application Class* type handler to the service operation definition that references the modified PeopleCode.

    See Chapter 10, "Managing Service Operations," Adding Handlers to Service Operations, page 259.

7. In the message definition that was migrated to the PeopleTools 8.48 or higher system, delete the PeopleCode event or Subscription that is defined on the message.

    Open the PeopleCode editor for the respective event and null out the program that exists there. Several warnings appear when saving the program, but your changes will be committed.

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Appendix D

Setting PS_FILEDIR and PS_SERVDIR Environment Variables

This appendix discusses how to:

- Set the PS_FILEDIR and PS_SERVDIR environment variable in Microsoft Windows environments.
- Set the PS_FILEDIR and PS_SERVDIR environment variable in UNIX environments.

Understanding Setting PS_FILEDIR and PS_SERVDIR Environment Variables

Some features of PeopleSoft Integration Broker enable you or require that you upload, import or consume data from files. For example, the system enables you to upload XML data from files to populate messages or consume WSDL from files.

In these and most other situations where you upload, import, or consume data from files, you must set the PS_FILEDIR environment variable on the machine on which the application server resides before you can successfully perform these actions.

Some features require that you also set the PS_SERVDIR environment variables

The procedure for setting the PS_FILEDIR and PS_SERVDIR environment variables is identical.

Setting PS_FILEDIR and PS_SERVDIR in Microsoft Windows Environments

Use the following procedure to set the PS_FILEDIR or PS_SERVDIR environment variable in Windows.

To set the PS_FILEDIR or PS_SERVDIR environment variable in Microsoft Windows environments:

1. Close any open DOS windows.
2. On your desktop, right-click the My Computer icon and click Properties.

   The System Properties dialog appears.
3. Click the Advanced tab.

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

4. In the Environment Variables section, click the Environment Variables button.

   The Environment Variables dialog box appears.

5. In the User variables for <user name> section, click New.

   A New User Variable dialog box displays.

   a. In the Variable Name field enter *PS_FILEDIR* or *PS_SERVDIR.*

   b. In the Variable Value field, enter *c:\temp.*

   c. Click the OK button to close the dialog box.

6. In the System variables section, click New. The New System Variable dialog box appears.

   a. In the Variable Name field enter *PS_FILEDIR* or *PS_SERVDIR.*

   b. In the Variable Value field, enter *c:\temp,* unless the documentation for the feature with which you are working states that you should enter a different value.

   c. Click the OK button to close the dialog box.

7. Click the OK button again to exit the System Properties dialog box.

Setting PS_FILEDIR and PS_SERVDIR in UNIX Environments

To set the PS_FILEDIR or PS_SERVDIR variable in UNIX use one of the following commands as appropriate for your UNIX environment.

Note that the path you specify in either command is the location from where the system will upload files. The documentation for the feature with which you are working may state a specific path to enter.

To set PS_FILEDIR:

- export PS_FILEDIR = <PS_HOME>/file
- setenv PS_FILEDIR = <PS_HOME>/file

To set PS_SERVDIR:

- export PS_SERVDIR = <PS_HOME>/file
- setenv PS_SERVDIR = <PS_HOME>/file

 Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.

Index

Copyright © 1988, 2013, Oracle and/or its affiliates. All Rights Reserved.