

---

# Portal Solutions 9.1 PeopleBook: Application Integration Framework

---

**September 2009**

Copyright © 1988, 2009, 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

<b>Application Integration Framework Preface .....</b>	<b>vii</b>
Application Integration Framework .....	vii
Common Terms Used in This PeopleBook .....	vii
PeopleBooks and the Online PeopleSoft Library .....	viii

## Chapter 1

<b>Getting Started with Application Integration Framework .....</b>	<b>1</b>
Understanding the Application Integration Framework .....	1
Application Integration Framework Implementation .....	1
Define Value Maps .....	1
Create Application Engine Transform Programs .....	2
Update Service Operation Routing .....	2

## Chapter 2

<b>Understanding Application Integration Framework .....</b>	<b>3</b>
Application Integration Framework .....	3
Use Cases for Application Integration Framework .....	5
Maps .....	6
Domain Value Map .....	6
Cross-Reference Map .....	7
Functions to Populate and Maintain the Cross-Reference and DVMs .....	7
Use Case: Integration Broker Transformation Without AIA Middleware .....	9
Use Case: Integration Broker Point-to-Point Transformation .....	13
Use Case: Integration Broker Transformation in Which a Third Party Uses AIA Middleware .....	17

## Chapter 3

<b>Defining and Populating Value Maps .....</b>	<b>21</b>
Understanding Value Maps .....	21
Defining Value Maps .....	24

Pages Used to Define Value Maps .....	25
Adding a Value Map .....	25
Adding Elements to the Map .....	26
Assigning Domains to the Value Map .....	27
Populating a Domain Value Map .....	28
Pages Used to Populate a Domain Value Map .....	28
Populating a Domain Value Map .....	28

## Chapter 4

<b>Creating Transform Programs and Updating Service Operations .....</b>	<b>29</b>
Understanding Transform Programs .....	29
Creating a Transform Program .....	30
Updating Service Operation Routing .....	31

## Chapter 5

<b>Accessing Maps Using XSLT Extension .....</b>	<b>33</b>
Understanding XSLT Extension Functions .....	33
Cross-Reference Functions .....	33
xref:populateXRefRow .....	33
xref:populateXrefRowNVP .....	36
xref:markForDelete .....	38
xref:markForDeleteNVP .....	39
xref:lookupXRef .....	40
xref:lookupXRefNVP .....	41
Domain Value Map Functions .....	43
dvm:lookupValue .....	43
dvm:lookupValueNVP .....	44
dvm:lookup-dvm .....	45
Generate-Guid Function .....	46
generate-guid .....	46
SetID Functions .....	47
SetID:lookupSetCtrlValues .....	47

## Chapter 6

<b>Accessing Maps Using PeopleCode .....</b>	<b>51</b>
Understanding Application Integration Framework Classes .....	51
How to Import Application Integration Framework Type Classes .....	51
DVM Utility Class Methods .....	52

LookupValue .....	52
LookupValue1M .....	53
LookupValueNVP .....	54
DVM Utility Class Properties .....	55
exceptionCaught .....	55
exceptionDetails .....	56
SetId Utility Class Methods .....	56
lookupSetCtrlValues .....	56
lookupSetID .....	57
Xref Utility Class Methods .....	59
LookupValue .....	59
LookupValue1M .....	60
LookupValueNVP .....	61
MarkForDelete .....	62
MarkForDeleteNVP .....	63
PopulateValue .....	64
PopulateValueNVP .....	65
Xref Class Properties .....	66
exceptionCaught .....	67
exceptionDetails .....	67

## Chapter 7

Accessing Maps Using Web Services .....	69
Understanding Application Integration Framework Web Services .....	69
Cross-Reference Lookup Web Service .....	70

## Appendix A

Application Integration Framework Example .....	73
Example Overview .....	73
Defining a Dynamic Value Map .....	74
Defining and Populating a Static Value Map .....	76
Using the XSLT Extension Function in the Transformation Program .....	79
Key Value Transformation .....	79
Domain Value Transformation .....	81
Updating the Service Operation Routing .....	82

Index .....	83
-------------	----



# Application Integration Framework

## Preface

This preface provides an overview of the Application Integration Framework PeopleBook.

---

## Application Integration Framework

Application Integration Architecture (AIA) is an effort across all Oracle divisions to facilitate integration among the various Oracle applications. In PeopleSoft applications, Integration Broker facilitates exposing PeopleSoft business logic as services and consuming external web services for PeopleSoft applications to invoke. Application Integration Framework builds upon Integration Broker functionality to provide a standard way to map data names and domain values used internally by PeopleSoft applications to the names and values used by external applications, particularly AIA.

This PeopleBook describes the processes for using the Application Integration Framework to define data transformation definitions and populate the transformation data. These processes include creating and populating data maps. They also include creating transformation application engine programs using XSLT functions, PeopleCode APIs, or web services to access the data maps and translate the data.

## Common Terms Used in This PeopleBook

<b>ABM</b>	Application Business Message: Defines the shape of the payload of a message on a point or edge system. ABM refers to the message structure used internally by the PeopleSoft application.
<b>AIA</b>	Application Integration Architecture: Refers to the common integration framework used to ease the challenges of integrating point systems.
<b>EBM</b>	Enterprise Business Message: Defines the shape of the payload of a message on the AIA. This is often referred to as the canonical message.
<b>EBO</b>	Enterprise Business Object: Refers to a data model consisting of standard business data object definitions and reusable data components representing a business object, such as Sales Order, Party, Item, and so forth.
<b>EBS</b>	Enterprise Business Service: Represents an application-independent web-service definition for performing a business task.
<b>IB</b>	PeopleSoft Integration Broker: Facilitates integrations among internal systems and third-party integration partners, while managing data structure, data format, and transport disparities.

---

## PeopleBooks and the Online PeopleSoft Library

A companion PeopleBook called PeopleBooks and the Online PeopleSoft 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.
- 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 PeopleSoft online library including full-text searching and configuring a reverse proxy server.
- Understanding documentation integration and how to integrate customized documentation into the library.
- Glossary of useful PeopleSoft terms that are used in PeopleBooks.

You can find this companion PeopleBook in your PeopleSoft online library.



## Chapter 1

# Getting Started with Application Integration Framework

This chapter provides an overview of the Application Integration Framework and discusses Application Integration Framework implementation.

---

## Understanding the Application Integration Framework

This PeopleBook describes using Application Integration Framework to create integrations between diverse systems using a common framework.

---

## Application Integration Framework Implementation

Application Integration Framework uses PeopleSoft Integration Broker framework. These implementation steps assume PeopleSoft Integration Broker is configured and service operations have been created for the integration.

To implement Application Integration Framework, you will:

- Define value maps.
- Populate domain value maps.
- Create application engine transform programs.
- Update service operation routing.

## Define Value Maps

To define value maps, perform the following steps:

<b>Step</b>	<b>Reference</b>
Define value maps	See <a href="#">Chapter 3, "Defining and Populating Value Maps,"</a> <a href="#">Defining Value Maps, page 24.</a>

## Create Application Engine Transform Programs

To create the application engine transform program, perform the following steps:

<b>Step</b>	<b>Reference</b>
Create application engine program	See <i>Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker</i> . Applying Filtering, Transformation and Translation, Defining Transform Programs
Code the XSLT step	See <i>Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker</i> . Creating Transform Programs and Updating Service Operations, Adding XSLT Steps to Transformation Programs

## Update Service Operation Routing

To update the service operation routing, perform the following steps:

<b>Step</b>	<b>Reference</b>
Update the routing	See <i>Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker</i> . Creating Transform Programs and Updating Service Operations, Updating Service Operation Routing

## Chapter 2

# Understanding Application Integration Framework

This chapter discusses:

- Application Integration Framework.
- Maps.
- Functions to populate and maintain the cross-reference and domain value maps (DVMs).
- Use Case: Integration Broker transformation without Application Integration Architecture (AIA) middleware.
- Use Case: Integration Broker point-to-point transformation.
- Use Case: Integration Broker transformation in which a third party uses AIA middleware.

---

## Application Integration Framework

Application Integration Framework extends PeopleSoft Integration Broker (IB) functionality to provide a standard way to represent, classify, store, query, publish, acquire, and invoke data that maps element names, structures, and values between PeopleSoft Application Business Messages (ABMs) and other applications. PeopleSoft Integration Broker provides the framework to send and receive messages with other PeopleSoft systems or third-party systems. If the message structure differs between systems, transformation programs are used to transform the incoming or outgoing message to a message format that the PeopleSoft system understands.

Each application that you are integrating with may use different data values or identifiers to represent the same information. For example, for a new customer in a PeopleSoft application, a new row is inserted in its customer database with a unique identifier such as *PS1001*. When the same information is propagated to an Oracle E-Business Suite application and a Siebel application, a new row should be inserted with different identifiers, such as *EBS1001* and *SBL1001*. The application integration framework enables you to transform this data.

## ***Application Integration Architecture***

Application Integration Architecture (AIA) is built on Oracle's Service Operation Architecture (SOA) as a unified approach for integrating business processes across applications, including third-party applications, based on a common architecture and common definition of business objects called Enterprise Business Objects (EBOs). These applications were designed using different technologies and use different names and structures to represent the same business object. AIA is the foundation for creating transformations on messages sent between diverse systems to integrate multiple applications without the need to create separate point-to-point integrations for each system involved.

AIA middleware can be used to transform business objects to a common object. Oracle's Fusion middleware includes AIA as well as prebuilt integrations for Oracle products.

## ***Oracle Application Integration Architecture Foundation Pack***

AIA Process Integration Packs (PIPs) are prebuilt integrations across Oracle applications, such as Siebel CRM, Oracle E-Business Suite, Agile PLM, and Oracle Communications Billing and Revenue Management. The integrations consist of EBOs and Enterprise Business Services (EBS). Enterprise Business Messages (EBMs) are designed to be operation-specific.

AIA provides a middle layer between the PeopleSoft system and other third-party systems. If a third-party Siebel customer uses AIA, when a message from that customer arrives at the PeopleSoft system, the common AIA names and values are seen by the PeopleSoft application, rather than the Siebel names and values. On the other hand, if a Siebel customer did not purchase the AIA PIP, then the PeopleSoft application would see the Siebel names and values.

---

**Note.** PIPs are available from the Oracle E-Delivery website.

---

## ***Application Integration Framework Integrations***

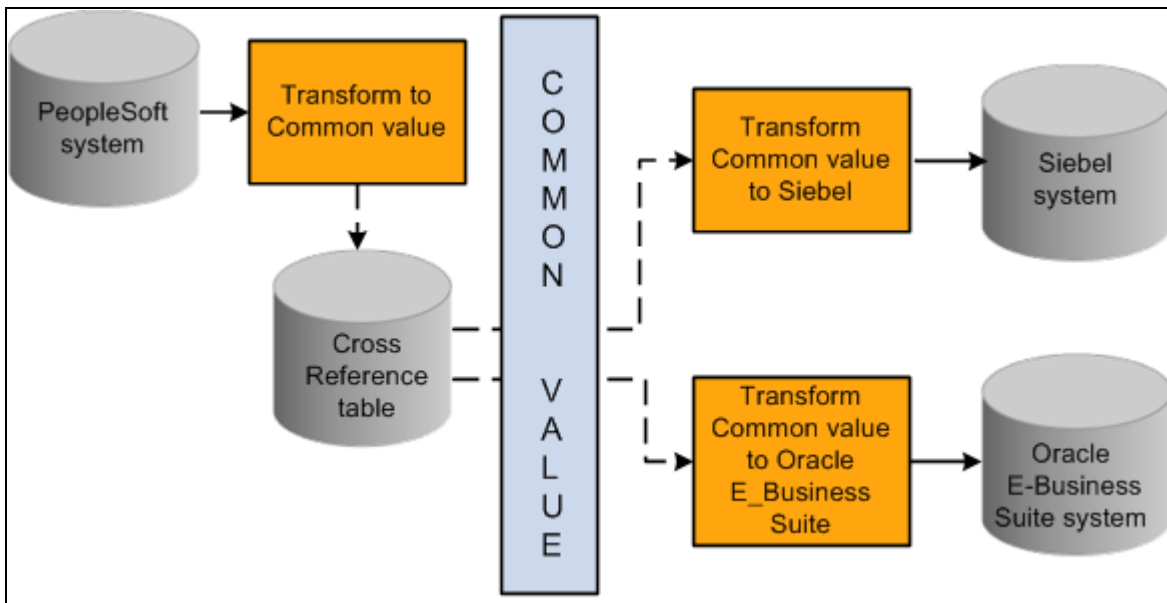
Application Integration Framework is designed to assist developers with the tasks required to produce integrations that are architected to be AIA-supportive. The integration produces ABMs with the same semantic content and approximately the same shape as the canonical (AIA) EBMs, thus minimizing transformation requirements. Each AIA-supportive integration that PeopleSoft applications delivers includes a sample transformation to enable the integrations to map as closely as possible to the EBOs. This strategy enables PeopleSoft customers to utilize IB to complete partner integrations without the need for middleware to perform transformations. Of course, customers who choose to purchase Oracle AIA middleware or who already own it can use the features of the EBS for transformations instead of or in addition to using the delivered IB samples.

---

**Note.** PeopleSoft-delivered integrations transform messages to the EBM format for direct integration with other Oracle products.

---

This diagram illustrates a PeopleSoft message transformed to a common value.



Common value mapping for outgoing message

## Use Cases for Application Integration Framework

Use cases fall into two categories:

- Key-mapping transforms using the dynamic cross-reference framework
- Static value transforms using the domain value map framework

These two frameworks are assumed to be separate; however, they in fact share common elements. The values of keys are open-ended and are usually extended; thus they are created programmatically during the transformation process, not in advance of the transform. Static value transforms generally have all values entered into the maps prior to the transformation process, and are less frequently extended.

In addition to these two categories, the integration use case depends on whether the third party is using AIA middleware.

This table shows where the transformations take place depending on whether AIA middleware is used:

<b>Use Case</b>	<b>PeopleSoft Integration Broker</b>	<b>AIA Middleware</b>	<b>Third Party</b>
Integration Broker Transformations without AIA middleware	Transform PeopleSoft ABM to EBM for outbound  Transform EBM to PeopleSoft ABM for inbound	not used	Transform EBM to third-party ABM for inbound  Transform third-party ABM to EBM for outbound

<b>Use Case</b>	<b>PeopleSoft Integration Broker</b>	<b>AIA Middleware</b>	<b>Third Party</b>
Integration Broker Point-to-Point transformation	Transform PeopleSoft ABM to third-party ABM for outbound  Transform third-party ABM to PeopleSoft ABM for inbound	not used	No transformations are necessary  Optionally, the third party can do the outbound transformation to PeopleSoft ABM.
Integration Broker Transformation in which third party uses AIA middleware	Transform PeopleSoft ABM to EBM for outbound  Transform EBM to PeopleSoft ABM for inbound	Transform EBM to third-party ABM for outbound from PeopleSoft application  Transform third-party ABM to EBM for inbound to PeopleSoft application	No transforms are performed

## Maps

Two different kinds of maps are available, domain values maps (DVM) and cross-reference maps (XREF). The maps are similar in that both maps group elements by domain name and are composed of elements that map specific values from one domain to another. The difference from a technical perspective is that XREF values are constantly being created, whereas DVM values are relatively static.

<b>XREF</b>	<b>DVM</b>
Usually keys	Usually attributes
Data maintained programmatically	Data maintained manually through the user interface

Map cardinality provides an independent classification of the mapping functions. Maps are based on single values (1:1) or multivalues (N;N).

## Domain Value Map

A Domain Value Map (DVM) is used for values that are relatively static and are relatively limited in total number, such as country codes and states. DVMs generally have all values entered into the maps prior to the transformation process, and are less frequently extended compared to cross-reference maps.

The DVM consists of domains, maps, and elements.

### Domain

A participating integrating system, for example, Siebel or Oracle E-Business Suite.

---

**Note.** Multiple domains can be maintained for a map.

---

<b>Maps</b>	A static object for which the mapping needs to be maintained, for example, currency code or country code.
<b>Element</b>	The unit of data in the local or remote message upon which an irreducible transformation operates. An irreducible transformation is one that cannot be broken into smaller transformations. Most elements contain a single data value, but that may not always be the case. For example, an address could be represented as a single string, or it could be composed of separate strings representing city, state, street, and house number. Examples of DVM elements are country code and common value.

## Cross-Reference Map

A cross-reference map is used for values that are dynamic in nature, such as key elements used to identify an instance. These are referred to as key maps or cross-references. The values are the names of larger data entities.

Cross-references consist of domains, maps, and elements.

<b>Domain</b>	A participating integrating system, for example, Siebel or E-Business Suite.
---------------	--

---

**Note.** Multiple domains can be maintained for a map.

---

<b>Maps</b>	A transaction object where the cross-reference for the keys is maintained, for example, voucher or vendor.
-------------	--

<b>Element</b>	An individual element captured as part of the key information, for example, SETID, VENDOR_ID.
----------------	---

---

## Functions to Populate and Maintain the Cross-Reference and DVMs

Three types of functions are available to query and maintain the DVM and cross-reference data during the transformation process.

<b>Application Class methods</b>	Application class methods are used in PeopleCode. These classes are typically used in cases in which the transformations are implemented as a PeopleCode step in an application engine program. These PeopleCode classes are used internally by the XPATH extension functions and by the web services.
<b>XPath extension functions</b>	XPath extension functions are used in the XSLT steps in application engine transform programs to invoke XSLT transforms using the TransformEx PeopleCode API.
<b>Web services</b>	Used by external systems to perform cross-reference lookups.

These functions enable you to query, manipulate, and delete cross-reference (XREF) and DVM data during transforms. The cardinality of the function is incorporated into the API naming convention. Each set of functions can be further broken down into three activities: lookup, populate, and delete.

The provided functions by class, activity, and form are:

<b><i>Class</i></b>	<b><i>Activity</i></b>	<b><i>Form</i></b>	<b><i>Description</i></b>
XREF	Populate	populateValue	Populate a transform item for a single valued element.
XREF	Populate	populateValue1M	Populate a transform element for a 1 to many mapping.
XREF	Populate	populateValueNVP	Populate a transform item for a multivalued element.
XREF	Delete	markForDelete	Mark for deletion a transform element for a single valued element. Items marked for delete can be reactivated later.
XREF	Delete	markForDeleteNVP	Mark for deletion a transform element for a multivalued element (name value pair).
XREF	Lookup	lookupValue	Look up a cross-reference value.
XREF	Lookup	lookupValue1M	Look up a cross-reference element for multiple values corresponding to a specific value in a reference element (1 to many).
XREF	Lookup	lookupValueNVP	Look up a cross-reference value for a multivalued element.
DVM	Lookup	lookupValue	Look up a domain value.
DVM	Lookup	lookupValue1M	Look up multiple domain values corresponding to a specific value in a reference element.
DVM	Lookup	lookupValueNVP	Look up a domain value for a multivalued element.
DVM	Lookup	lookup-dvm	Look up a domain value.



## Use Case: Integration Broker Transformation Without AIA Middleware

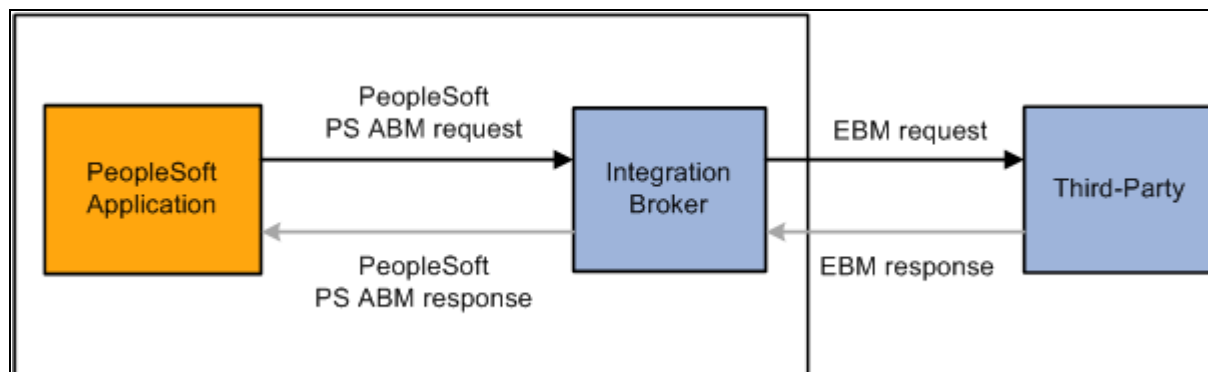
This section discusses the use case in which both the PeopleSoft and third-party maps and transforms take place within the PeopleSoft Integration Broker through Application Integration Framework. In this use case, PeopleSoft applications can take advantage of a canonical integration model without the need to purchase AIA middleware.

Use case includes:

- Outbound request or post to a third party.
- Inbound request or post from a third party.

### ***Outbound Request or Post to a Third Party***

This diagram illustrates an outbound request or post to a third party:



Outbound request to a third party without AIA middleware

The following steps are performed in this scenario:

1. A user in the PeopleSoft system triggers publishing of an AIA supportive integration to a third-party system.

2. Within PeopleSoft Integration Broker, a transform on the outbound routing generates the EBM. The method used to translate the data values depends on the transaction. This table lists the transaction types and the necessary transformation action.

<i><b>Transaction</b></i>	<i><b>Action</b></i>
Dynamic key add-request	If the transaction is an add-request, the transform creates a new common key (GUID) and uses the appropriate form of the <b>populate XRef</b> XPath extension function to put the new common key and the corresponding PeopleSoft keys into the PeopleSoft cross-reference framework.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the common key using the appropriate form of the <b>lookup XRef</b> XPath extension function with the PeopleSoft keys as input. If a common key does not exist, the developer determines whether to create a new common key, generate an error, or return an error message in the reply message if the integration supports it.
Dynamic key delete request	For asynchronous notification (request-only) integrations that delete a keyed entity, developers may choose to leave the cross-reference values in place for historical purposes or, if desired, they may choose to delete the cross-reference values. To mark the PeopleSoft keys for deletion, the transform uses the appropriate form of the <b>deleteXRef</b> XPath extension function with the PeopleSoft keys as input.
Static value lookup	If the transaction uses a domain value map, the transform looks up the PeopleSoft values using the appropriate form of the <b>lookupDVM</b> XPath extension function with the PeopleSoft values as input. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values by default, leaves them blank, throws an error, or returns an error message in the reply message if the integration supports it.

3. The EBM is routed to the third-party system for processing.

Integrations supporting response messages have these additional steps.

1. The third-party system processes the request, formats the reply message, and then returns it along with the common key or value.

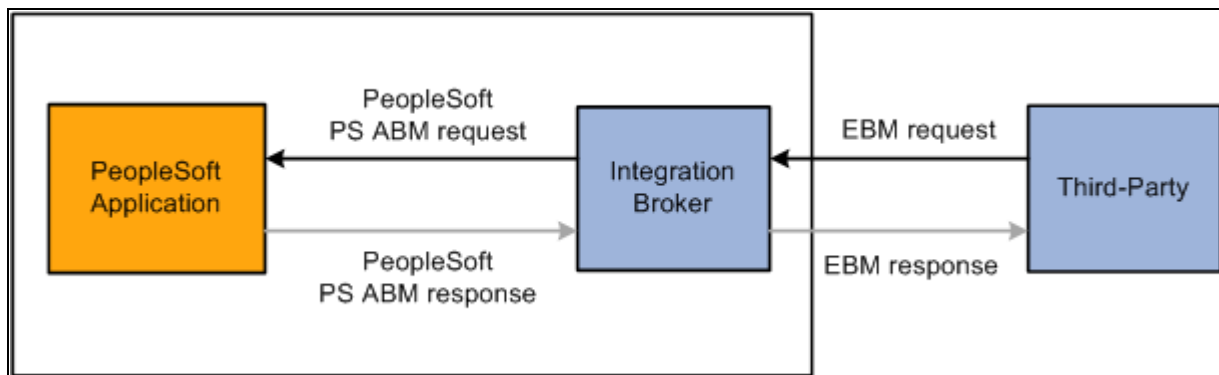
2. Within PeopleSoft Integration Broker, asynchronous request-reply operations have a transform to look up the PeopleSoft keys using the appropriate form of the **lookupXref** XPath extension function with the common key as input. The PeopleSoft keys are then put into the PeopleSoft ABM. Synchronous operations do not require this lookup because the PeopleSoft application already knows the PeopleSoft keys from the initial request.

<i><b>Transaction</b></i>	<i><b>Action</b></i>
Asynchronous request-reply operation using dynamic key value	Requires a transform program to look up the PeopleSoft keys using the appropriate form of the <b>lookupXref</b> XPath extension function with the common key as input. The PeopleSoft keys are then put into the PeopleSoft ABM.
Synchronous operations using dynamic key value	Lookup is not required because the PeopleSoft application already knows the PeopleSoft keys from the initial request.
Asynchronous request-reply operation using DVM value	Requires a transform program to look up the PeopleSoft values using the appropriate form of the <b>lookupDVM</b> XPath extension function with the common value as input. The PeopleSoft keys are then put into the PeopleSoft ABM.

3. The PeopleSoft ABM reply is then returned to the originating PeopleSoft application.

### ***Inbound Request or Post from a Third Party***

This diagram illustrates an inbound request from a third party:



Inbound request from third party without AIA middleware

1. A user in the third-party system triggers publishing of an integration to a PeopleSoft system.
2. The EBM is routed to the PeopleSoft system for processing.

3. Within PeopleSoft Integration Broker, a transform on the inbound routing performs the following actions based on transaction type:

<b>Transaction</b>	<b>Action</b>
Dynamic key add request	If the transaction is an add-request, the transform leaves the PeopleSoft keys blank in the ABM and passes the common key along for use in the handler.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the PeopleSoft keys using the appropriate form of the <b>lookup XRef</b> XPath extension function with the common key in the EBM as input. Optionally, this can be done in the PeopleSoft inbound handler through the appropriate PeopleSoft <b>lookup XRef</b> API with the common key as input.
DVM lookup	The transform on the inbound routing looks up the PeopleSoft values using the appropriate form of the <b>lookupDVM</b> XPath extension function with the common value from the EBM. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values, omits them, throws an error, or returns an error message in the reply message if the integration supports it.

4. The PeopleSoft inbound handler processes the message.

<b>Transaction</b>	<b>Action</b>
Dynamic key add request	If the transaction is an add-request, it creates the PeopleSoft keys and then uses the appropriate form of the <b>populate XRef</b> API to add the PeopleSoft keys and the corresponding common key to the PeopleSoft cross-reference framework. If the keys cannot be created or added to the framework, an error occurs or an error message is returned in the reply message if the integration supports it.
Dynamic key lookup	If the transaction is not an add-request, the message is processed with the PeopleSoft keys from the ABM. If the PeopleSoft key values are not found, an error occurs or a message is returned in the reply message if the integration supports it.

Integrations supporting response messages have these additional steps.

1. The PeopleSoft application processes the request and returns the ABM reply to the PeopleSoft inbound handler.
2. The PeopleSoft inbound handler formats the EBM reply and returns it to the third-party system with the common key.
3. Optionally, if the third-party system uses the PeopleSoft cross-reference framework to persist their key mappings, asynchronous request-reply operations need to look up the third-party keys using the appropriate form of the **lookup XRef** web service with the common key from the EBM reply as input.

## Use Case: Integration Broker Point-to-Point Transformation

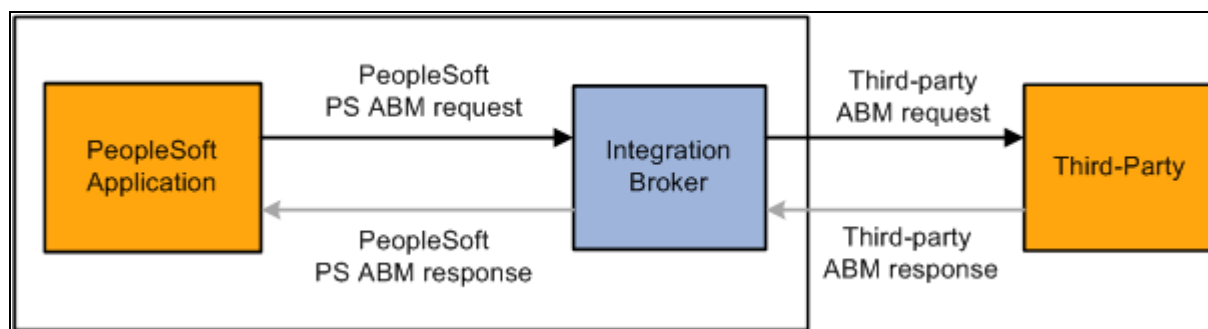
This section discusses the use case in which PeopleSoft Integration Broker performs all of the transformations with a third party. In the previous use case, the messages were transformed to the EBM format; in this use case, the message is transformed into the third-party ABM. If the third party is another PeopleSoft system, no transform is necessary.

Use case includes:

- Outbound request or post to a third party.
- Inbound request or post from a third party.

### ***Outbound Request or Post to a Third Party***

This diagram illustrates an outbound point-to-point request or post to a third party:



Outbound point-to-point request to a third party

The following steps are performed in this scenario:

1. A user in the PeopleSoft system triggers publishing of a point-to-point integration to a third-party system.

2. Within PeopleSoft Integration Broker, a transform on the outbound routing generates the third-party ABM. The method used to translate the data values depends on the transaction. This table lists the transaction types and the necessary transformation action.

<i><b>Transaction</b></i>	<i><b>Action</b></i>
Dynamic key add-request	If the transaction is an add-request, the transform creates a new common key (GUID) and uses the appropriate form of the <b>populate XRef</b> XPath extension function to put the new common key, corresponding PeopleSoft key, and third-party key in the PeopleSoft cross-reference framework.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the key using the appropriate form of the <b>lookup XRef</b> XPath extension function with the PeopleSoft keys as input. If a common key does not exist, the developer determines whether to create a new common key, generate an error, or return an error message in the reply message if the integration supports it.
Dynamic key delete request	For asynchronous notification (request-only) integrations that delete a keyed entity, developers may choose to leave the cross-reference values in place for historical purposes or, if desired, they may choose to delete the cross-reference values. To mark the PeopleSoft keys for deletion, the transform uses the appropriate form of the <b>deleteXRef</b> XPath extension function with the PeopleSoft keys as input.
Static value lookup	If the transaction uses a domain value map, a transform on the outbound routing looks up the common value using the appropriate form of the <b>lookupDVM</b> XPath extension function with the PeopleSoft value as input. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values by default, omits them, throws an error, or returns an error message in the reply message if the integration supports it.

3. The third-party ABM is routed to the third-party system for processing.

Integrations supporting response messages have these additional steps.

1. The third-party system processes the request, formats the reply message, and then returns it along with the common key or value.

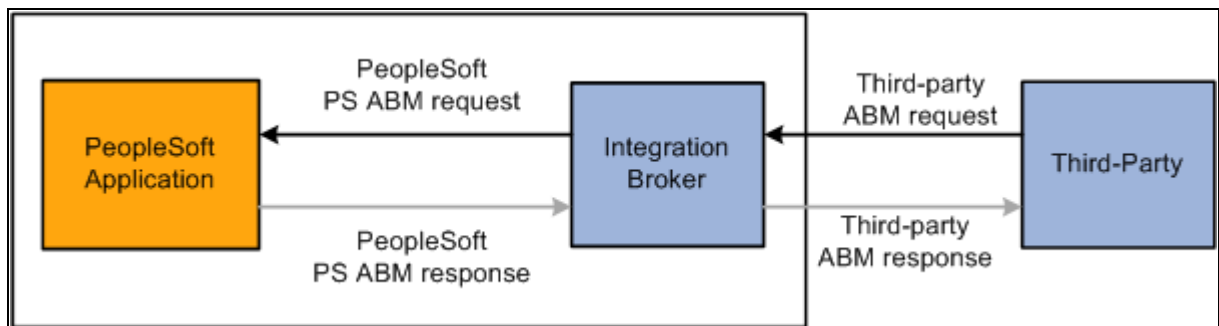
2. Within PeopleSoft Integration Broker, asynchronous request-reply operations have a transform to look up the PeopleSoft keys using the appropriate form of the lookup **XRef** XPath extension function with the third-party key as input. The PeopleSoft keys are then put into the PeopleSoft ABM. Synchronous operations do not require this lookup because the PeopleSoft application already knows the PeopleSoft keys from the initial request.

<b>Transaction</b>	<b>Action</b>
Asynchronous request-reply operation using dynamic key value	Requires a transform program to look up the PeopleSoft keys using the appropriate form of the <b>lookupXRef</b> XPath extension function with the third-party key as input. The PeopleSoft keys are then put into the PeopleSoft ABM.
Synchronous operations using dynamic key value	Lookup is not required because the PeopleSoft application already knows the PeopleSoft keys from the initial request.
Asynchronous request-reply operation using DVM value	Requires a transform program to look up the PeopleSoft value using the appropriate form of the <b>lookupDVM</b> XPath extension function with the third-party value as input. The PeopleSoft keys are then put into the PeopleSoft ABM.

3. The PeopleSoft ABM reply is returned to the originating PeopleSoft application.

### ***Inbound Request or Post from a Third Party***

This diagram illustrates an outbound point-to-point request or post to a third-party:



Inbound point-to-point request from a third party

1. A user in the third-party system triggers publishing an integration to a PeopleSoft system.
2. In this example, the request is sent in the third-party ABM format.

---

**Note.** In many instances, the third-party may be aware of the PeopleSoft ABM format and perform a transformation before sending the message; in this case, no transform is required.

---

3. Within PeopleSoft Integration Broker, a transform on the inbound message performs the following actions based on transaction type:

<b>Transaction</b>	<b>Action</b>
Dynamic key add request	If the transaction is an add-request, the transform leaves the PeopleSoft keys blank in the ABM and passes the third-party key along for use in the handler.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the PeopleSoft keys using the appropriate form of the <b>lookup XRef</b> XPath extension function with the reference key in the third-party ABM as input.
DVM lookup	The transform on the inbound routing looks up the PeopleSoft values using the appropriate form of the <b>lookupDVM</b> XPath extension function with the reference key from the third-party ABM. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values, omits them, throws an error, or return an error message in the reply message if the integration supports it.

4. Within PeopleSoft Integration Broker, a transform on the inbound message performs the following actions based on transaction type.
5. The PeopleSoft Inbound handler processes the message.

<b>Transaction</b>	<b>Action</b>
Dynamic key add request	If the transaction is an add-request, it creates the PeopleSoft keys and then uses the appropriate form of the <b>populate XRef</b> API to add the PeopleSoft keys and the corresponding UniqueGUID to the PeopleSoft cross-reference framework. If the keys cannot be created or added to the framework, an error occurs or a message is returned in the reply message if the integration supports it.
Dynamic key lookup	If the transaction is not an add-request, the message is processed with the PeopleSoft key values from the ABM. If the PeopleSoft key values not found, an error occurs or a message is returned in the reply message if the integration supports it.

Integrations supporting response messages have these additional steps.

1. The PeopleSoft application processes the request and the PeopleSoft inbound handler formats the ABM reply and returns it along with the common key or value.
2. No key translation is required in the transform, so the EBM reply is then returned to the third-party system along with the common key from the ABM reply.



## Use Case: Integration Broker Transformation in Which a Third Party Uses AIA Middleware

From a PeopleSoft perspective, this use case has an identical flow to the transformation without the AIA middleware. From a third-party perspective, however, it is similar in flow, but not in implementation. In this case, the third-party system integrates with the AIA using their ABM instead of with PeopleSoft software using an EBM. All of the third-party value maps and transforms between the EBM and the third-party ABM are done on the AIA layer using the EBS graphical-mapper and XPath extension functions instead of being done on the third-party system.

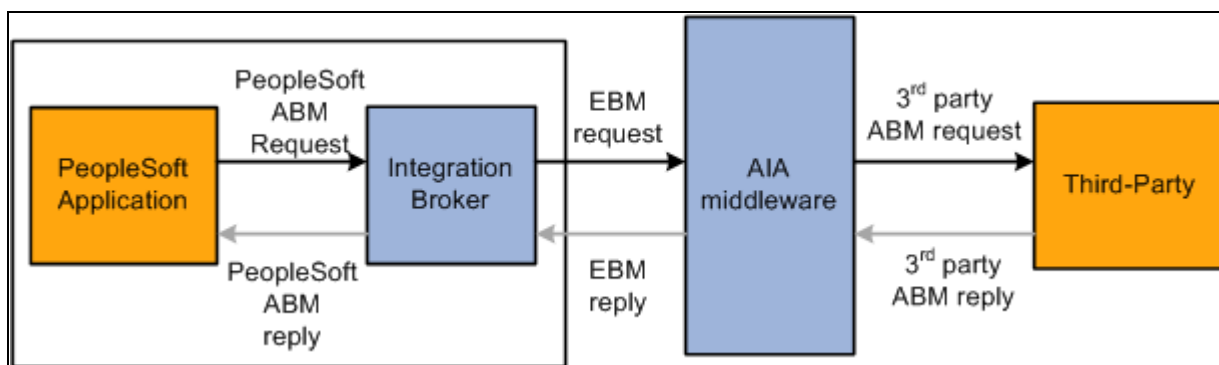
**Note.** This is the model used with AIA PIPs.

Use cases include:

- Outbound request or post to a third party.
- Inbound request or post from a third party.

### ***Outbound Request or Post to a Third Party***

In this use case, a request is sent from the PeopleSoft application to a third party that uses their own AIA middleware to perform message transformation.



Outbound request to a third party that uses AIA middleware

1. A user in the PeopleSoft system triggers publishing of an AIA supportive integration to a third-party system through the AIA.

2. Within PeopleSoft Integration Broker, a transform on the outbound routing occurs, generating the EBM. The method used to translate the data values depends on the transaction. This table lists the transaction types and the necessary transformation action.

<b>Transaction</b>	<b>Action</b>
Dynamic key add request	If the transaction is an add-request, the transform creates a new common key (GUID) and uses the appropriate form of the <b>populateXRef</b> XPath extension function to put the new common key and the corresponding PeopleSoft keys in the PeopleSoft cross-reference framework.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the common key using the appropriate form of the <b>lookupXRef</b> XPath extension function with the PeopleSoft keys as input. If a common key does not exist, the developer determines whether to create a new common key, generate an error, or return an error message in the reply message if the integration supports it.
Dynamic key delete request	For asynchronous notification (request-only) integrations that delete a keyed entity, developers may choose to leave the cross-reference values in place for historical purposes or, if desired, they may choose to delete the cross-reference values. To mark the PeopleSoft keys for deletion, the transform uses the appropriate form of the <b>deleteXRef</b> XPath extension function with the PeopleSoft keys as input.
DVM lookup	If the transaction uses a domain value map, the transform looks up the PeopleSoft values using the appropriate form of the <b>lookupDVM</b> XPath extension function with the common value from the EBM. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values by default, omits them, throws an error, or returns an error in the reply message if the integration supports it.

3. The EBM is routed to the AIA.
4. Upon receiving the EBM request, the AIA transforms the common key to the third-party key and sends the transformed request to the third party using their ABM.

Integrations supporting response messages use these additional steps.

1. The third party processes the request, formats the reply message, and returns it to the AIA middleware.
2. The AIA transforms the third-party key in the reply to the common key and returns the EBM reply to PeopleSoft software.

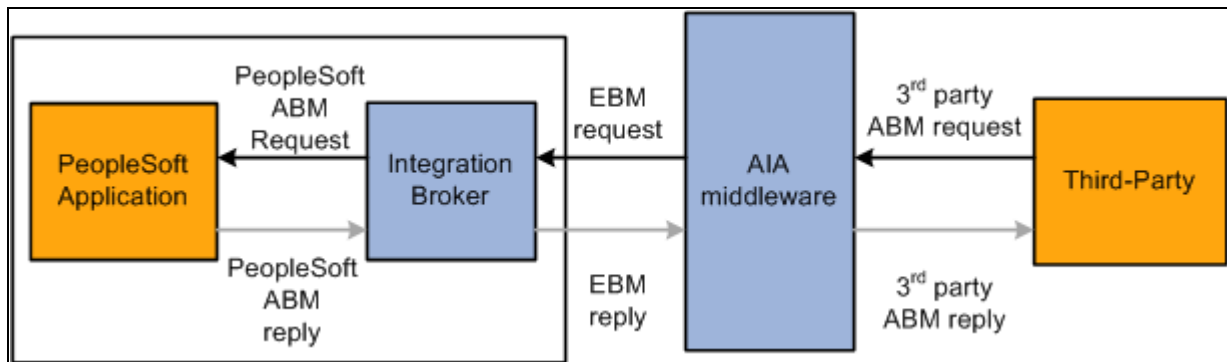
3. Within PeopleSoft Integration Broker, asynchronous request-reply operations have a transform program to transform the common key or value and place it into the PeopleSoft ABM. Synchronous operations do not require this lookup because the PeopleSoft application already knows the PeopleSoft keys or values from the initial request.

<b>Transaction</b>	<b>Action</b>
Dynamic key lookup	Transform looks up the PeopleSoft keys using the appropriate form of the <b>lookup XRef</b> XPath extension function with the common key as input.
Dynamic key delete	(Optional) For integrations that delete a keyed entity, developers may choose to leave the cross-reference values in place for historical purposes or, if desired, they may choose to delete the cross-reference values. To mark the PeopleSoft keys for deletion, the transform uses the appropriate form of the <b>deleteXRef</b> XPath extension function with the PeopleSoft keys as input.
DVM lookup	The transform looks up the common value using the appropriate form of the lookup DVM XPath extension function with the PeopleSoft values as input.

4. The PeopleSoft ABM reply is returned to the originating PeopleSoft application.

### ***Inbound Request or Post from a Third Party***

In this use case, a third-party application creates a request that is transformed using the third-party AIA middleware and sent to the PeopleSoft application.



Inbound request from a third party using AIA middleware

1. A user in the third-party system triggers publishing of an integration to a PeopleSoft system through the AIA.
2. The AIA transforms the third-party ABM key and values to the common key and values and routes the EBM request to the PeopleSoft system for processing.

3. Within PeopleSoft Integration Broker, a transform on the inbound routing transforms the EBM to PeopleSoft ABM using one or more of the following methods depending on the transaction.

<i><b>Transaction</b></i>	<i><b>Action</b></i>
Dynamic key add	If the transaction is an add-request, the transform leaves the PeopleSoft keys blank in the ABM and passes the common key along for use in the handler.
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the PeopleSoft key using the appropriate form of the <b>lookupXRef</b> XPath extension function with the common key in the EBM as input. Optionally, this can be done in the PeopleSoft inbound handler using the appropriate PeopleSoft lookup XRef API with the common key as input.
DVM lookup	A transform on the inbound routing looks up the PeopleSoft values using the appropriate form of the lookup DVM XPath extension function with the common value from the EBM. If a value is not found, the developer determines whether the transform supplies the PeopleSoft values by default, leaves them blank, throws an error, or returns an error message in the reply message if the integration supports it.

For synchronous and asynchronous request-reply messages, the following additional steps are performed.

1. The PeopleSoft application processes the request and the PeopleSoft inbound handler formats the ABM reply and returns it along with the common key or value.
2. No key translation is required in the IB transform, so the EBM reply is returned to the AIA for routing back to the third-party system.
3. The AIA transforms the common key and values to the third-party ABM key and values and routes the third-party ABM reply back to the third-party system.

## Chapter 3

# Defining and Populating Value Maps

This chapter provides an overview of value maps and discusses how to:

- Define value map.
- Populate a domain value map.

---

## Understanding Value Maps

The Define Value Maps component enables users to define and classify value maps and elements. Each value map must be defined as either dynamic (cross-reference) or static (DVM).

Value maps support the following types of maps:

- One-to-one
- One-to-many
- Multiple elements per domain

### ***One-to-One Cross-Reference***

Cross-reference maps support a one-to-one relationship between two systems. For example, the PeopleSoft system uses one ID for customer A, the enterprise business object (EBO) or common value uses a GUID, and the external system uses another ID for the same customer.

The cross-reference map would be defined like this:

<b><i>PeopleSoft</i></b>	<b><i>UniqueGUID</i></b>	<b><i>External System</i></b>
PS	UniqueGUID	EXT
PS100	<guid1>	EXT-100
PS102	<guid2>	EXT-102

In this scenario, when a PeopleSoft application creates a new customer and sends the create customer message to the external system, the routing includes a transformation program that uses the cross-reference map to translate the data.

### One-to-Many Cross-Reference

Cross-reference maps support a one-to-many relationship between two systems. Two or more values in a system may correspond to a single value in another system. For example, three different job codes might exist in the PeopleSoft system that correspond to one job code in the external system.

The cross-reference map would be defined like this:

<b>PeopleSoft</b>	<b>UniqueGUID</b>	<b>External System</b>
PS	UniqueGUID	EXT
AS01 AS02 AS03	<guid1> <guid2> <guid3>	ASST
MG01	<guid4>	MNGR

In this scenario, when the PeopleSoft application assigns or changes a job code and sends the message to the external system, the routing includes a transformation program that uses the cross-reference map to translate the data. If the integration requires translation from the external system value to a PeopleSoft value, the developer is responsible for determining how to handle the multiple PeopleSoft values returned from the lookup. Options include replicating the source data for each value, implementing a method of choosing a default value, and generating an error for the transaction.

### Cross-Reference with Multiple Domains

Many times a system implements functionality using compound keys. This is supported in the cross-reference framework by means of multiple elements to identify the value set. For example, item integrations from a PeopleSoft application to an external system would use a cross-reference map to translate SetID/ItemID on the PeopleSoft system to Product on the external system.

The cross-reference map would be defined like this:

<b>PeopleSoft</b>	<b>PeopleSoft</b>	<b>UniqueGUID</b>	<b>External System</b>
SETID	ITEMID	UniqueGUID	Product
SHARE	1001	<guid1>	RP001
SHARE	1002	<guid2>	RP002
SHARE	1003	<guid3>	RP003
SHR03	1000	<guid4>	RP006

In this scenario, when the PeopleSoft application creates an item and sends the message to the external system, the routing includes a transformation program that maps the setID and itemID to a common element (UniqueGUID) and the external system receives the translated data.

### ***One-to-One DVM***

In a domain value map, the one-to-one relationship contains the actual data values. For example, the PeopleSoft application uses the short name for the state code, the EBO defines state code using the full name as the common value, and the external application uses an abbreviated name for state code.

The DVM would be defined like this:

<b><i>PeopleSoft</i></b>	<b><i>Common</i></b>	<b><i>External System</i></b>
Short	Full	Abbrev
MA	Massachusetts	Mass
CA	California	Calif

In this scenario, when a PeopleSoft application creates an outbound message to the external system that includes the state code, the routing includes a transformation program mapping the short name to the full name and the external system requires a transformation from the full name to the abbreviated name. The transformation for the external system can be done by means of the AIA middleware or a proprietary transformation on the external system.

### ***One-to-Many DVM***

In the domain value map, a one-to-many relationship is created by entering multiple values for one domain mapping to a single value in another domain. For example, in the PeopleSoft application, multiple person types can map to a single person type in the external application.

The DVM would be defined like this:

<b><i>PeopleSoft</i></b>	<b><i>UniqueGUID</i></b>	<b><i>External System 2</i></b>
PS	UniqueGUID	EXT2
PS001 PS002	<guid1>	SB001
PS003	<guid2>	SBL002

In this scenario, when a PeopleSoft application creates an outbound message to an external system that includes the person type, the routing includes a transformation program to translate the data values.

### ***DVM with Multiple Elements Per Domain***

In the DVM, you can map multiple elements to define a value set. For example, in the PeopleSoft application, the Business Unit/Chartfield combination could map to a Ledger/Segment combination in an external application.

In this scenario, when a PeopleSoft application creates an outbound message to the external system that includes the person type, the routing includes a transformation program providing the data translation.

The DVM would be defined like this:

<b>PeopleSoft</b>	<b>PeopleSoft</b>	<b>UniqueGUID</b>	<b>External System 1</b>	
Business Unit	Chartfield	UniqueGUID	Segment	Ledger
US001	ACCOUNT	<guid1>	SEGMENT1	Ledger1
US001	DEPARTMENT	<guid2>	SEGMENT2	Ledger1
US001	PRODUCT	<guid3>	SEGMENT3	Ledger1
US002	ACCOUNT	<guid4>	SEGMENT1	Ledger2

In this scenario, when a PeopleSoft application creates an outbound message to an external system that includes the Business Unit/Chartfield, the routing includes a transformation program to translate the data values for the multiple elements to a single UniqueGUID. The external system would then need to transform the UniqueGUID to the equivalent Segment/Ledger.

### **DVM with Qualifiers**

Qualifiers provide additional context to uniquely identify mapping values. A mapping may not be valid unless qualified with additional contextual information. For example, a domain value map containing city code to city name mapping may have multiple mappings for a city based on the country. For example, Kensington is a city in Canada as well as the United States.

The DVM would be defined like this:

<b>EXT (Qualifier 1) PSFT (Qualifier 1)</b>	<b>EXT (Qualifier 2) PSFT (Qualifier 2)</b>	<b>Common</b>	<b>EXT</b>	<b>PSFT</b>
Country	State	Common	CityCode	CityName
USA	Minnasota	BELG_MN	BELG	Belgrade
USA	North Carolina	BELG_NC	BELG	Belgrade
USA	Kansas	KN_KS	KN	Kensington
Canada	Prince Edward Island	KN_PEI	KN	Kensington

In this scenario, when the PeopleSoft application creates an outbound message to an external system that includes the Business Unit/Chartfield, the routing includes a transformation program mapping city name, country, and state to a common value.

---

## **Defining Value Maps**

This section discusses how to:

- Add a value map.
- Add elements to a value map.



- Assign domains to a value map.

## Pages Used to Define Value Maps

<i>Page Name</i>	<i>Definition Name</i>	<i>Navigation</i>	<i>Usage</i>
Define Value Maps	EOTF_DEFINE_MAPS	Enterprise Components, Integration Definitions, Transformation Framework, Define Value Maps	Assign elements to the map.
Define Value Maps - Domains	EOTF_DEFINE_MAPS2	From the Define Value Maps-Elements page, select the Domains tab	Assign elements to domains.

## Adding a Value Map

Access the Define Value Maps search page (Select Enterprise Components, Integration Definitions, Transformation Framework, Define Value Maps.)

Define Value Maps search page: Add a New Value tab

To add a value map:

1. Select the Add a New Value tab.
2. In the Map Name field, enter a name for the map.
3. In the Map Type field, select the map type.
4. Click the Add button.

The Elements page appears, where you can define the elements for the map.

---

**Note.** After you save the map, you can not change the map type.

---

### Map Types

Maps are either static or dynamic.

<b>Domain Value Map (static)</b>	Static map to which values are provided by means of the Populate Domain value component.
<b>Cross Reference (dynamic)</b>	Dynamic map to which values are provided based on key information.

Adding Elements to the Map

Access the Define Value Map - Elements page (Select Enterprise Components, Integration Definitions, Transformation Framework, Define Value Maps.)

ElementsDomains

Map Name:StateCodeDVMType:Domain Value Map (static)

\*Description:State & Country Table DVM

Comments:

Assign Elements to Maps

CustomizeFind

First1-4 of 4Last

Order	*Element Name	*Data Type	Length	Required		
1	UniqueGUID	String	36	<input checked="" type="checkbox"/>	+	-
2	COUNTRY	String	3	<input checked="" type="checkbox"/>	+	-
3	STATE	String	6	<input checked="" type="checkbox"/>	+	-
4	COMMON	String	32	<input checked="" type="checkbox"/>	+	-

Define Map Values - Elements page

<b>Description</b>	Enter a description for the value map.
<b>Comments</b>	Enter comments for the value map.
<b>Order</b>	The common element is always assigned order 1. All other elements must be assigned an order of 2 or higher.
<b>Element Name</b>	<div>Enter the element name. Select one and only one element as the common element. The common element must always contain a unique value for each row of data entered into both dynamic and static value maps.</div> <div>For dynamic maps, the common element must be a unique guid.</div> <div>For static maps, the common element can be assigned as required.</div>

**Data Type**

Values are:

- *String*
- *Numeric*

The data type is used for validation when you are entering data values.

**Length**

Enter the length of the element.

The length is used for validation when you are entering data values.

**Required**

Select to indicate that this is a required element.

## Assigning Domains to the Value Map

Access the Define Value Map - Domains page (Select the Domains tab on the Define Value Map - Elements page.)

The screenshot displays the 'Define Maps Values - Domains' page for the 'StateCodeDVM' value map. The page has two tabs: 'Elements' and 'Domains', with 'Domains' currently selected. The 'Map Name' is 'StateCodeDVM' and the 'Type' is 'Domain Value Map (static)'. The page is divided into three sections, each for a different domain.

**Domain 1: AIA**

- \*Domain Name: AIA
- ☒ Is Unique
- Assign Elements to the Domain: 1 of 1
- Element 1: COMMON

**Domain 2: PSFT**

- \*Domain Name: PSFT
- ☒ Is Unique
- Assign Elements to the Domain: 1-2 of 2
- Element 1: COUNTRY
- Element 2: STATE

**Domain 3: UniqueGUID**

- Domain Name: UniqueGUID
- ☒ Is Unique
- Assign Elements to the Domain: 1 of 1
- Element 1: UniqueGUID

Define Maps Values - Domains page

Each element in a value map must be assigned to at least one domain.

<b>Domain Name</b>	Enter the domain name for the participating system.
<b>Is Unique</b>	Select to indicate that elements within the domain make up a unique instance.
<b>Element Name</b>	Select the element to assign.

## Populating a Domain Value Map

This section discusses how to populate a static value map.

### Pages Used to Populate a Domain Value Map




<i>Page Name</i>	<i>Definition Name</i>	<i>Navigation</i>	<i>Usage</i>
Populate Domain Value Maps	EOTF_POPULATE_DVMS	Enterprise Components, Integration Definitions, Transformation Framework, Populate Domain Value Maps	Populate a domain value map.











### Populating a Domain Value Map

Access the Populate Domain Value Maps page (Select Enterprise Components, Integration Definitions, Transformation Framework, Populate Domain Value Maps).

### Populate Domain Value Maps

**Map Name:** StateCodeDVM **Description:** State & Country Table DVM

**Assign Data to Value Maps** Customize | Find |  First  1-5 of 5  Last

	COUNTRY	STATE	COMMON		
1	USA	CA	900		
2	USA	NY	901		
3	USA	MD	902		
4	USA	MI	903		
5	CAN	NF	904		

Populate Domain Value Maps page

You use domain value maps to enter and maintain data in static value maps. The elements that you define for the value map make up the columns displayed on the page. Elements are ordered and validated as specified in the map definition.

A unique value must be entered for each row of data in every DVM. When you save the page, the component looks for a cached rowset and destroys it. The cache is reloaded the next time it is called.

## Chapter 4

# Creating Transform Programs and Updating Service Operations

This chapter provides an overview of transform programs and discusses how to:

- Create a transform program
- Update service operation routing

---

## Understanding 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 and 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 that you want to achieve.

You specify which transform program to apply within a routing definition for a service operation.

### ***Transformation Programming Languages***

You can use PeopleCode or Extensible Stylesheet Language Transformation (XSLT) as a programming language for creating transformation logic. XSLT is a recognized standard language that is well-suited to manipulating XML structures, so it is highly recommended for transformations.

PeopleSoft applications provide XSLT extension functions and PeopleCode APIs to perform value map lookups, deletes, and population.

---

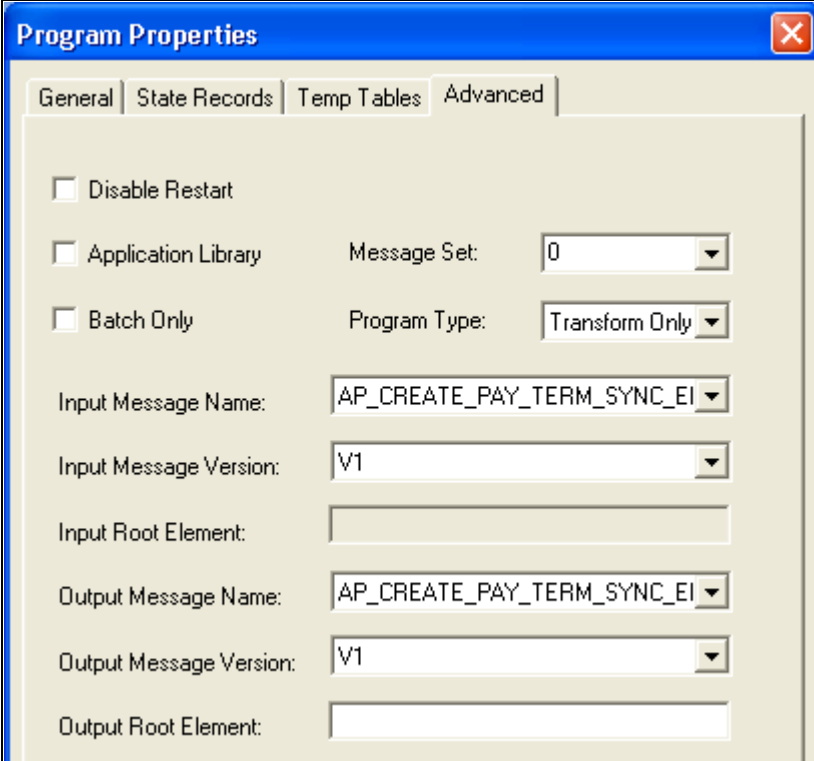
**Note.** When programming using XSLT, you can manually code the XSLT or use the Oracle XSL Mapper to graphically associate records and fields. The Oracle XSL Mapper then automatically generates the XSLT code.

---

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, Applying Filtering, Transformation and Translation, Developing Transform Programs*

## Creating a Transform Program

Create a new application engine program in Application Designer. On the toolbar, click the Properties button and select the Advanced tab.



**Program Properties**

General | State Records | Temp Tables | **Advanced**

☐ Disable Restart

☐ Application Library      Message Set: 0

☐ Batch Only      Program Type: Transform Only

Input Message Name: AP\_CREATE\_PAY\_TERM\_SYNC\_EI

Input Message Version: V1

Input Root Element:

Output Message Name: AP\_CREATE\_PAY\_TERM\_SYNC\_EI

Output Message Version: V1

Output Root Element:

Application Engine program properties for a transform program

The application engine program must be defined as program type *Transform Only*. Optionally, you can indicate the input and output message name and version. The input and output message name and version are required if you want to use the Oracle Graphical Mapper to create the XSLT for the transform program.

---

**Note.** Refer to *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, Applying Filtering, Transformation and Translation, Developing Transform Programs* for details on creating a transform program using Oracle Graphical Mapper.

---

Refer to the chapter "Accessing Maps Using XSLT" for the syntax to use for your cross-references and domain value maps.

---

**Note.** Snippets of code are provided in the appendix, "Application Integration Framework Example."

---

## Updating Service Operation Routing

A routing definition contains routing parameters for each inbound request, inbound response, outbound request, and outbound response associated with a service operation. For each request or response, you define the routing alias, message names before and after transformation, and transformation program names.

Your transform program is invoked by PeopleSoft Integration Broker if you specify its name in the a routing definition for a service operation.

### Adding Routing Parameters

To access the Routing Definition page to add routing parameters:

1. Select Integration Broker, Integration Setup, Service Operations.
2. Select the Routing tab.
3. Either click the link for an existing routing or add a new routing.
4. If it is a new routing, add the routing information.
5. Select the Parameters tab.

Routing Definitions		Parameters	Connector Properties
<b>Routing Name:</b>	AP_CREATE_PAY_TERM_SYNC_EFF		
<b>Service Operation:</b>	AP_CREATE_PAY_TERM_SYNC_EFF		
<b>Service Operation Version:</b>	v1		
<b>Sender Node:</b>	E900B20		
<b>Receiver Node:</b>	PSFT_XOUTBND		
<b>Parameters</b>			
<b>Type:</b>	Outbound Request		
<b>External Alias:</b>	CreatePaymentTerm		
	<a href="#">Alias References</a>		
<b>Message.Ver into Transform 1:</b>	AP_CREATE_PAY_TERM_SYNC_EFF.V1		
<b>Transform Program 1:</b>	AP_PT_ADD_XF		
<b>Transform Program 2:</b>			
<b>Message.Ver out of Transforms:</b>	AP_CREATE_PAY_TERM_SYNC_EBM.V1		

Routing Parameters page for asynchronous one-way service operation

If the application engine transform program includes the messages into and out of the transformation, the messages are automatically populated with these values. If the application engine program does not include the message names, enter the appropriate message names and save the page.

---

**Note.** For synchronous service operations, you can define transformation on both the outbound and inbound messages.

---

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker*, Managing Routing Definitions, Creating Routing Definitions.



## Chapter 5

# Accessing Maps Using XSLT Extension

This chapter provides an overview of Extensible Stylesheet Language Transformation (XSLT) extension functions and discusses:

- Cross-reference functions
- Domain value map (DVM) functions
- Generate-guid function
- SetID functions

---

## Understanding XSLT Extension Functions

When you have created your value maps, you need to write a transformation program that will map the elements in the message and use the value maps for data translation.

PeopleSoft applications provide XSLT extension functions to perform value map lookups, deletes, and population. Two nearly identical sets of functions are available, one for cross-references and one for DVMs.

---

## Cross-Reference Functions

This section describes the cross-reference XSLT extension functions.

### **xref:populateXRefRow**

#### **Syntax**

```
xref:populateXRefRow(mapName,referenceElementName,referenceValue,elementName ,  
elementValue,mode)
```

#### **Description**

Use the populateXRefRow function to populate a cross-reference element with a value.

## Parameters

<i>Parameter</i>	<i>Description</i>
mapName	The name of the cross-reference map, as string.
referenceElementName	The name of the reference element, as string.
referenceValue	The value corresponding to the reference element name, as string.
elementName	The name of the element to be populated, as string.
elementValue	The value with which to populate the element, as string.
mode	The mode in which the function populates the element. You can specify any of the following values: ADD, LINK, or UPDATE. The mode parameter values are case-sensitive and must be specified in the uppercase only.

## Returns

This function returns the cross-reference value being populated as a string.

This table lists the results for the Xref:populateXRefRow function.

<i>Mode</i>	<i>Reference Value</i>	<i>Value to Be Added</i>	<i>Result</i>
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

## Example

This table lists examples of the modes with their descriptions and exception reasons:

<b>Mode</b>	<b>Description</b>	<b>Exception Reasons</b>
ADD	<p>Adds the reference value and the value. For example:</p> <pre>xref:populateXRefRow( "customers"⇒ , "PS", "PS101", "Common", "CM001", "⇒ ADD" )</pre> <p>adds the reference value <i>PS101</i> in the <i>PS</i> element and the value <i>CM001</i> in the <i>Common</i> element of the <i>customers</i> cross-reference map.</p>	<p>Exceptions can occur for the following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The value being added is not unique across that element for that map.</li> <li>• The element for that row already contains a value.</li> <li>• The reference value exists.</li> </ul>
LINK	<p>Adds the cross-reference value corresponding to the existing reference value. For example:</p> <pre>xref:populateXRefRow( "customers"⇒ , "Common", "CM001", "SBL", "SB⇒ 101", "LINK" )</pre> <p>adds the value <i>SB-101</i> in the <i>SBL</i> element of the <i>customers</i> cross-reference map and links it to the value <i>CM001</i> in the <i>Common</i> element.</p>	<p>Exceptions can occur for the following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The reference value is not found.</li> <li>• The value being linked exists in that element for that map.</li> </ul>
UPDATE	<p>Updates the cross-reference value corresponding to an existing reference element-value pair. For example:</p> <pre>xref:populateXRefRow( "customers"⇒ , "PS", "PS100", "PS", "PS1001", "UPD⇒ ATE" )</pre> <p>updates the value <i>PS100</i> in the <i>PS</i> element of the <i>customers</i> cross-reference map to value <i>PS1001</i>.</p>	<p>Exceptions can occur for the following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The value being updated is not unique across that element for that map.</li> <li>• Multiple values are found for the element being updated.</li> <li>• The reference value is not found.</li> <li>• The element for that row does not have a value.</li> </ul>

## xref:populateXrefRowNVP

### Syntax

**xref:populateXrefRowNVP**(*mapName*,*referenceDomain*,*referenceNVP*,*targetDomain* ,  
*targetNVP*,*mode*)

### Description

Use the xref:populateXrefRowNVP function to populate multiple elements in the cross-reference map with values.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	The name of the cross-reference map, as string.
referenceDomain	The name of the reference domain, as string.
referencesNVP	NVP list of reference elements and values, as string.
targetDomain	The name of the domain to be populated, as string.
targetNVP	NVP list of elements and values to be populated in the elements, as string.
mode	The mode in which the function populates the element. You can specify any of the following values: ADD, LINK, or UPDATE. The mode parameter values are case-sensitive and must be specified in the uppercase only.

### Returns

This table lists the results for the populateXrefRowNVP function.

<i>Mode</i>	<i>Reference Value</i>	<i>Value to Be Added</i>	<i>Result</i>
ADD	Absent	Absent	Success
	Present	Absent	Exception
	Present	Present	Exception
LINK	Absent	Absent	Exception
	Present	Absent	Success
	Present	Present	Exception

<b>Mode</b>	<b>Reference Value</b>	<b>Value to Be Added</b>	<b>Result</b>
UPDATE	Absent	Absent	Exception
	Present	Absent	Exception
	Present	Present	Success

## Example

This table lists the modes with their descriptions and exception reasons:

<b>Mode</b>	<b>Description</b>	<b>Exception Reasons</b>
ADD	<p>Adds the reference value and the value to be added. For example:</p> <pre>xref:populateXRefRowNVP("Items", =&gt; "PeopleSoft", "&lt;Setid&gt;SHARE&lt;=&gt; /Setid&gt;&lt;ItemID&gt;1005&lt;/ItemID&gt;", "C=&gt; ommon", "&lt;Common&gt;"   generate-&gt; guid()   "&lt;/Common&gt;", "ADD")</pre> <p>adds the reference values <i>SHARE/1005</i> in the <i>PeopleSoft</i> domain and the value <i>&lt;guid1&gt;</i> in the <i>Common</i> domain.</p>	<p>Exceptions can occur for the following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified domains are not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The values being added are not unique across that domain for that map.</li> <li>• The element for that row already contains a value.</li> <li>• The reference value exists.</li> </ul>
LINK	<p>Adds the cross-reference value corresponding to the existing reference value. For example:</p> <pre>xref:populateXRefRowNVP("Items=&gt; ", "PeopleSoft", "&lt;Setid&gt;SHA=&gt; RE&lt;/Setid&gt;&lt;ItemID&gt;1005&lt;/ItemID&gt;"=&gt; , "Retail", "&lt;Product&gt;RP0005&lt;=&gt; /Product&gt;", "LINK")</pre> <p>adds value <i>RP005</i> to the <i>Retail</i> domain and links it to reference values <i>SHARE/1005</i> in the <i>PeopleSoft</i> domain.</p>	<p>Exceptions can occur due for following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified domains are not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The reference value is not found.</li> <li>• The value being linked exists in that domain for that map.</li> </ul>

<b>Mode</b>	<b>Description</b>	<b>Exception Reasons</b>
UPDATE	<p>Updates the cross-reference value corresponding to an existing reference element-value pair. For example:</p> <pre>xref:populateXRefRowNVP( "Items⇒ ", "PeopleSoft", "&lt;Setid&gt;SHA⇒ RE&lt;/Setid&gt;&lt;ItemID&gt;1000&lt;/ItemID&gt;"⇒ , "PeopleSoft", "&lt;Setid&gt;SHARE&lt;⇒ /Setid&gt;&lt;ItemID&gt;10000&lt;/ItemID&gt;" , "⇒ UPDATE" )</pre> <p>updates the value <i>1000</i> in the <i>ItemID</i> element of the <i>PeopleSoft</i> domain to value <i>10000</i>.</p>	<p>Exceptions can occur for the following reasons:</p> <ul style="list-style-type: none"> <li>• The specified cross-reference map is not found.</li> <li>• The specified domains are not found.</li> <li>• The specified elements are not found.</li> <li>• The values provided are empty.</li> <li>• The values being updated are not unique across that domain for that map.</li> <li>• Multiple values are found for the domain being updated.</li> <li>• The reference value is not found.</li> <li>• The element for that row does not have a value.</li> </ul>

## xref:markForDelete

### Syntax

**xref:markForDelete**(*mapName*, *elementName*, *elementValue*)

### Description

Use the xref:markForDelete function to delete a value in a cross-reference map when the element specified is the only element for a single domain. The value in the element is marked as deleted. If multiple domains reference the element or the domain the element is referenced by has multiple primary elements, use the xref:markForDeleteNVP function instead.

A cross-reference map row should have at least two mappings. Therefore, if you have only two mappings in a row and you mark one value for delete, then the value in another element is also deleted.

Any element value marked for delete is treated as if the value does not exist. Therefore, you can populate the same element with the xref:populateXRefRow function in ADD mode. However, if the element value is marked for delete as a reference, it cannot be used in the LINK mode of xref:populateXRefRow function.

### Parameters

<b>Parameter</b>	<b>Description</b>
mapName	The cross-reference map name, as string.
elementName	The name of the element from which you want to delete a value, as string.

<i>Parameter</i>	<i>Description</i>
elementValue	The value to be deleted, as string.

## Returns

This function returns true if deletion was successful; otherwise, it returns false.

An exception can occur for the following reasons:

- The cross-reference map with the given name is not found.
- The specified element name is not found.
- The specified element name is not unique to a domain.
- The specified value is empty.
- The specified value is not found in the element.
- Multiple values are found.

## Example

The following code deletes the *PS001* value in the *PS* element of the *customers* cross-reference map:

```
xref:markForDelete("customers","PS","PS001")
```

## xref:markForDeleteNVP

### Syntax

```
xref:markForDeleteNVP(mapName,referenceDomain,referenceNVP)
```

### Description

Use the xref:markForDeleteNVP function to delete a set of values in a cross-reference map for a specified domain. The values in the elements are marked as deleted.

A cross-reference map row should have at least two mappings. Therefore, if you have only two mappings in a row and you mark one value for delete, then the value in the other domain is also deleted.

Any values marked for delete are treated as if they do not exist. Therefore, you can populate the same elements with the xref:populateXRefRowNVP function in ADD mode. However, if the element value is marked for delete as a reference, it cannot be used in the LINK mode of xref:populateXRefRowNVP function.

## Parameters

<i>Parameter</i>	<i>Description</i>
mapName	The cross-reference map name, as string.
referenceDomain	The name of the reference domain, as string.
referenceNVP	NVP list of reference elements and values that you want to delete, as string.

## Returns

This function returns true if deletion was successful; otherwise, it returns false.

An exception can occur for the following reasons:

- The cross-reference map with the given name is not found.
- The specified element name is not found.
- All primary elements in this domain have not been specified.
- The specified value is empty.
- The specified value is not found in the element.
- Multiple values are found.

## Example

The following code deletes the specified values in the *Setid* and *ItemID* elements of the *PeopleSoft* domain from the *Items* cross-reference map:

```
xref:markForDeleteNVP("Items","PeopleSoft","<Setid>SHARE</Setid><ItemID>1000</ItemID>")
```

## xref:lookupXRef

### Syntax

```
xref:lookupXRef(mapName,referenceElementName,xrefReferenceValue,elementName,needAnException)
```

### Description

Use the lookupXRef function to look up a cross-reference element for a value that corresponds to a specific value in a reference element.



## Parameters

<i>Parameter</i>	<i>Description</i>
mapName	The name of the cross-reference map, as string.
referenceElementName	The name of the reference element, as string.
referenceValue	The value corresponding to the reference element name, as string.
elementName	The name of the element to be looked up for the value, as string.
needAnException	Specify true or false.  If the needAnException parameter is set to true, an exception occurs if the value being looked up in the map is not found. If the needAnException parameter is set to false, an empty value is returned if the value being looked up in the map is not found.

## Returns

The value of the requested element.

An exception can occur for the following reasons:

- The cross-reference map with the given name is not found.
- The specified element names are not found.
- The specified reference value is empty.
- Multiple target values are found.

## Example

The following code looks up the *Common* element of the *customers* cross-reference map for a value corresponding to the *PS001* value in the *PS* element:

```
xref:lookupXRef("customers","PS","PS001","Common",true())
```

## xref:lookupXRefNVP

### Syntax

```
xref:lookupXRefNVP(mapName,referenceDomain,referenceNVP,targetDomain,needAnException)
```

## Description

Use the `lookupXRefNVP` function to look up cross-reference values that correspond to a specified set of values in a reference domain. All primary elements in the reference domain must be included in the reference NVP list, but any qualifier elements are optional.

## Parameters

<i>Parameter</i>	<i>Description</i>
<code>mapName</code>	The name of the cross-reference map, as string.
<code>referenceDomain</code>	The name of the reference domain, as string.
<code>referenceNVP</code>	NVP list of reference elements and values, as string.
<code>targetDomain</code>	The name of the domain to be looked up for the values, as string.
<code>needAnException</code>	Specify true or false.  If the <code>needAnException</code> parameter is set to true, an exception occurs if the value being looked up in the map is not found. If the <code>needAnException</code> parameter is set to false, an empty value is returned if the value being looked up in the map is not found.

## Returns

The return string includes values for all primary and qualifier elements in the target domain as an NVP list.

An exception can occur for the following reasons:

- The cross-reference map with the given name is not found.
- The specified domain names are not found.
- The specified element names are not found.
- The specified reference value is empty.
- Multiple target values are found.

## Example

The following code looks up the values of all elements in the *Common* domain of the *Items* cross-reference map that correspond to values *SHARE/1000* in the *PeopleSoft* domain:

```
xref:lookupXRefNVP("Items","PeopleSoft", "<Setid>SHARE</Setid><ItemID>1000</ItemID>", "Common", true())
```

---

## Domain Value Map Functions

This section describes the domain value map functions.

### dvm:lookupValue

#### Syntax

```
dvm:lookupValue(mapName,referenceElementName,referenceValue,elementName,  
defaultValue,needAnException)
```

#### Description

The `dvm:lookupValue` function finds the reference element value in a domain value map and returns the equivalent value of the specified element name as a string. This form of DVM lookup is used to find a single reference element and return a single element value. Lookups involving multiple elements in a reference or return domain need to be done using the `dvm:lookupValueNVP` function.

#### Parameters

<i>Parameter</i>	<i>Description</i>
<code>mapName</code>	The domain value map name, as string.
<code>referenceElementName</code>	The source element name, as string.
<code>referenceValue</code>	The source value (an XPath expression bound to the source document of the XSLT transformation), as string.
<code>elementName</code>	The target element name, as string.
<code>defaultValue</code>	If the value is not found, then the default value is returned, as string.
<code>needAnException</code>	Specify true or false.  If the <code>needAnException</code> parameter is set to true, an exception occurs if the value being looked up in the map is not found. If the <code>needAnException</code> parameter is set to false, an empty value is returned if the value being looked up in the map is not found.

#### Returns

The `dvm:lookupValue` returns a string containing the value of the element.

An exception can occur for the following reasons:

- The DVM map with the given name is not found.
- The specified elements are not found.
- The specified source value is empty.

## Example

The following code looks up the value of the *Short* element in the *StateCodes* DVM map corresponding to the *California* value in the *Long* element:

```
dvm:lookupValue("StateCodes","Long","California","Short","CouldNotBeFound",True)
```

## dvm:lookupValueNVP

### Syntax

```
dvm:lookupValueNVP(mapName,referenceDomain,referenceNVP,targetDomain,defaultNVP,needAnException)
```

### Description

The `dvm:lookupValueNVP` function finds the reference domain element values in a DVM and returns the equivalent values of all elements in the specified domain as an NVP list. This form of DVM lookup should be used when multiple elements exist in either the reference or return domain. All required elements in the reference domain must be included in the reference NVP list, but optional elements (qualifiers perhaps) do not have to be included. The return string will include values for all elements in the target domain as an NVP list regardless of whether they are required.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	The domain value map name, as string.
referenceDomain	The source domain name, as string.
referenceNVP	NVP list of source elements and values, as string.
targetDomain	The target domain name, as string.
defaultNVP	If the value is not found, then the default values specified are returned, as string.
needAnException	Specify true or false.

## Returns

The return string will include values for all elements in the target domain as an NVP list regardless of whether they are required.

An exception can occur for one of the following reasons:

- The DVM map with the given name is not found.
- The specified domains are not found.
- The specified elements are not found.
- The specified source values are empty.

## Example

The following code looks up the specified values of the *BusinessUnit* and *Chartfield* elements in the *PeopleSoft* domain of the *ChartElements* DVM and returns the value of the *UniqueGUID* element:

```
dvm:lookupValueNVP("ChartElements","PeopleSoft","<BusinessUnit>US100</BusinessUnit><Chartfield>ACCOUNT</Chartfield>","UniqueGUID","<UniqueGUID>CouldNotBeFound</UniqueGUID>",True)
```

## dvm:lookup-dvm

### Syntax

```
dvm:lookup-dvm(mapName,referenceElementName,referenceValue,elementName,defaultValue,needAnException)
```

### Description

The `dvm:lookup-dvm` function finds the reference element value in a domain value map and returns the equivalent value of the specified element name as a string. This form of DVM lookup is used to find a single reference element and return a single element value. Lookups involving multiple elements in a reference or return domain need to be done using the `dvm:lookupValueNVP` function. This function is identical in purpose and function to the `dvm:lookupValue` function. It exists to mimic the function names defined in the ESB implementation of DVM.

### Parameters

Parameter	Description
mapName	The domain value map name, as string.
referenceElementName	The name of the source element in the DVM, as string.

<b><i>Parameter</i></b>	<b><i>Description</i></b>
referenceValue	The source value (an XPath expression bound to the source document of the XSLT transformation), as string.
elementName	The name of the target element in the DVM, as string.
defaultValue	A default value to assign to the target element if no value is found, as string.
needAnException	Specify true or false.  If the needAnException parameter is set to true, an exception occurs if the value being looked up in the map is not found. If the needAnException parameter is set to false, an empty value is returned if the value being looked up in the map is not found.

## Returns

This function returns a string by looking up the value for the target element in the DVM, where the value for the source element is equal to the source value. The source value is an XPath expression bound to the source document of the XSLT transformation. The expression is evaluated during the transformation and the result value is passed as the source value for lookup.

An exception can occur for the following reasons:

- The DVM map with a given name is not found.
- The specified elements are not found.
- The specified source value is empty.

## Example

The following code looks up the value of the *Short* element in the *StateCodes* DVM map corresponding to the *Calif* value in the *Abbrev* element:

```
dvm:lookup-dvm("StateCodes", "Abbrev", "Calif", "Short", "CouldNotBeFound", True)
```

---

## Generate-Guid Function

This section discuss the generate-guid function.

### generate-guid

#### Syntax

```
generate-guid()
```

## Description

Use this function to generate a guid.

## Parameters

none

## Returns

This function returns a string containing a randomly generated globally unique identifier (GUID).

## Example

The following code generates a random GUID that could be used as a new common key value:

```
generate-guid()
```

---

## SetID Functions

This section describes the SetID XSLT extension function.

### SetID:lookupSetCtrlValues

#### Syntax

**SetID:lookupSetCtrlValues** (*SetId,LookupType,dvmTranslate,dvmMapName,SourceElementName , TargetElementName,needAnException*)

#### Description

Use the lookupSetCtrlValues function to look up the list of set control values associated with the setID in the context of a record group or record. Optionally, each set control value can be translated through a DVM map if a map name, source element name, and target element name are provided.

#### Parameters

<i>Parameter</i>	<i>Description</i>
SetId	The SetId value interested in lookup.

<b>Parameter</b>	<b>Description</b>
LookupType	The lookup type is used to indicate the type of lookup. You can specify either <i>1</i> for the record group name or <i>2</i> for the record name.
LookupValue	Value should be either a record name or record group ID as determined by LookupType.
dvmTranslate	Specify <i>True</i> if translation to Common ID using DVM Name supplied is desired. Specify <i>False</i> if no translation is needed.
dvmMapName	DVM map to be used in translation if requested.
SourceElementName	The source element name to be used in DVM translation if requested.
TargetElementName	The target element name to be used in DVM translation if requested.
needAnException	Specify true or false to indicate whether an exception should occur if set control values are not found.

## Returns

This function returns a list of set control values or a list of translated set control values as a concatenated string that could be parsed in XSLT.

## Example

This example looks up the set control values (representing PeopleSoft business units in this example) associated with the setID *SHARE* for the record group *VENDOR* and translates them to the common IDs for Business Unit by means of the DVM mapping *BusinessUnit*. An exception is requested if set control values are not found:

```
xref:lookupSetCtrlValues("SHARE", "1", "VENDOR", true(), "BusinessUnit", "PSFT_⇒
BU", "COMMON", true())
```

Exceptions can occur for the following reasons:

- The DVM map name specified is not valid.
- The source element name specified is not associated with the DVM map.
- The target element name specified is not associated with the DVM map.
- No translated value is found for the set control value in the DVM map.
- If needAnException is set to true, an exception will occur if the set control values cannot be found for the given SETID/Record or record group ID.

This example looks up the set control values (representing PeopleSoft Business Units in this example) associated with the SETID *SHARE* for the record *VENDOR\_LOC* and translates them to the common IDs for Business Unit by means of the DVM mapping *BusinessUnit*. An exception is not requested if set control values are not found:



```
xref:lookupSetCtrlValues("SHARE","2", "VENDOR_LOC",true(),"BusinessUnit","PSFT_⇒  
BU","COMMON",false())
```

This example looks up the set control values (representing PeopleSoft Business Units in this example) associated with the setID *SHARE* for the record group *VENDOR*. An exception is requested if set control values are not found:

```
xref:lookupSetCtrlValues("SHARE","1", "VENDOR",false(),"","","",true())
```

An exception can occur if the set control values cannot be found for the given SETID/Record or record group ID.



## Chapter 6

# Accessing Maps Using PeopleCode

This chapter provides an overview of Application Integration Framework classes and discusses:

- How to import Application Integration Framework type classes
- DVM utility class methods
- DVM utility class properties
- SetId utility class methods
- Xref utility class methods
- Xref class properties

---

## Understanding Application Integration Framework Classes

Application Integration Framework classes provide functions to perform value map lookups, deletes, and population. The following functions mirror the functionality of the XPath extension functions provided for XSLT transformation.

<b>DVM Utility Class</b>	Functions for PeopleCode developers to access the data for a domain value map (DVM) during transformations. These functions mirror the functionality of the XPath extension functions provided for XSLT transformations.
<b>SetId Utility Class</b>	Functions for PeopleCode developers to access SetId data stored in a DVM during transformations.
<b>Xref Utility Class</b>	Functions for PeopleCode developers to access the data for a dynamic (cross-reference) value map during transformations.

---

## How to Import Application Integration Framework Type Classes

The Application Integration Framework type classes are not built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing Application Integration Framework classes, Oracle recommends that you import the functions class in the application package that is specific to your needs.

The function classes are stored in the following application packages:

- EOTF\_CORE:DVM
- EOTF\_CORE:SetId
- EOTF\_CORE:Xref

You should use one of the following import statements:

```
import EOTF_CORE:DVM:Functions;
import EOTF_CORE:SetId:Functions;
import EOTF_CORE:Xref;
```

---

## DVM Utility Class Methods

This section describes the DVM utility class methods. The methods are discussed in alphabetical order.

### LookupValue

#### Syntax

**LookupValue**(*mapName*,*referenceElementName*,*referenceValue*,*elementName*,*defaultValue*,  
*needAnException*)

#### Description

Locate the reference element value in a domain value map, and return the equivalent value for the specified element name. This form of DVM lookup is used to find a single reference element and return a single element value.

#### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a static value map definition (DVM), as string.
referenceElementName	Name of an element in the DVM in which to look for a value, as string.
referenceValue	Value of an element in the DVM to look for, as string.
elementName	Name of the element in the DVM to return an equivalent value for, as string.

<i>Parameter</i>	<i>Description</i>
defaultValue	Default value to be returned if needAnException is false and an error occurs, as string.
needAnException	<i>True</i> to return error messages, <i>false</i> to return the default value.

## Returns

The equivalent value of elementName in the DVM, or the default value.

## Example

This example is used to look up the value *&guid1* in the *UniqueGUID* element of the *&TestName* DVM and return the equivalent value of element *&EBS1*.

```
Local string &returnValue = &dvm.LookupValue(&TestName, &UniqueGUID, &guid1,⇒
    &EBS1, "Value not found.", True);
```

This example will look up a value that does not exist in element *&EBS1* of the *&TestName* DVM to verify that the default value is returned when the NeedAnException parameter is false.

```
&returnValue = &dvm.LookupValue(&TestName, &EBS1, "NotFound", &UniqueGUID, &Value⇒
    NotFound, False);
```

## LookupValue1M

### Syntax

**LookupValue1M**(mapName,referenceElementName,referenceValue,elementName,needAnException)

### Description

Locate the reference element value in a domain value map, and return the equivalent values of the specified element name as an NVP list. This form of DVM lookup is used to find a single reference element and return one to many equivalent values for the specified element.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a static value map definition (DVM), as string.
referenceElementName	Name of an element in the DVM in which to look for a value, as string.

<i><b>Parameter</b></i>	<i><b>Description</b></i>
referenceValue	Value of an element in the DVM to look for, as string.
elementName	Name of the element in the DVM to return equivalent values for, as string.
needAnException	<i>True</i> to return error messages, <i>false</i> to return the default values.

## Returns

An array of string containing the equivalent values of elementName in the DVM.

## Example

This example will look up value *&guid1* in the *UniqueGUID* element of the *&TestName* DVM and return the equivalent value of element *&EBS1*

```
&returnValue = &dvm.LookupValue1M(&TestName, &UniqueGUID, &guid1, &EBS1, True);
```

This example will look up a value that does not exist in the *UniqueGUID* element of the *&TestName* DVM to verify that no value is returned when the NeedAnException parameter is false:

```
&returnValue = &dvm.LookupValue1M(&TestName, &UniqueGUID, "NotFound", &EBS1, =>
False);
```

## LookupValueNVP

### Syntax

**LookupValueNVP** ( *mapName*, *referenceDomain*, *referenceNVP*, *targetDomain*, *defaultNVP*, *needAnException* )

### Description

Locate the reference domain element values in a DVM, and return the equivalent values of all elements in the specified domain as an NVP list. This form of DVM lookup should be used when multiple elements exist in either the reference or return domain. All required elements in the reference domain must be included in the reference NVP list, but optional elements (qualifiers perhaps) do not have to be included. The return string will include values for all elements in the target domain as an NVP list regardless of whether they are required.

### Parameters

<i><b>Parameter</b></i>	<i><b>Description</b></i>
mapName	Name of a static value map definition (DVM), as string.

<b>Parameter</b>	<b>Description</b>
referenceDomain	Name of a domain in the DVM in which to look for a value, as string.
referenceNVP	Name value pairs of elements and values in the DVM domain to look for, as an array of DataElement.
targetDomain	Name of the domain to return equivalent values for, as string.
defaultNVP	Default values (NVPs) to be returned if needAnException is false and an error occurs, as an array of DataElement.
needAnException	<i>True</i> to return error messages, <i>false</i> to return an NVP with the default values.

## Returns

An array of DataElement. Name value pairs containing the equivalent values for the elements in the target domain, or the default values.

## Example

This example is used to look up value *&guid1* in the *UniqueGUID* domain of the *&TestName* DVM and return the equivalent values for the *&RTK* domain.

```
Local array of EOTF_CORE:Common:DataElement &UniqueGUIDrequestValues = CreateArray⇒
(create EOTF_CORE:Common:DataElement(&UniqueGUID));

&UniqueGUIDrequestValues [1].value = &guid1;
Local array of EOTF_CORE:Common:DataElement &returnValue = &dvm.LookupValueNVP⇒
(&TestName, &UniqueGUID, &UniqueGUIDrequestValues, &RTK, &DefaultRTKValues,⇒
True);
```

---

## DVM Utility Class Properties

This section describes the DVM utility class properties.

### exceptionCaught

#### Description

Value true when a lookup method suppresses an exception because the needAnException parameter was false.

## exceptionDetails

### Description

Exception object containing the detail of the exception that was caught.

---

## SetId Utility Class Methods

This section describes the SetId Utility class methods. The methods are discussed in alphabetical order.

## lookupSetCtrlValues

### Syntax

**lookupSetCtrlValues**(*SetId*,*LookupType*,*LookupName*,*dvmTranslate*,*dvmMapName*,*PsftElementName*,*CommonElementName*,*needAnException*)

### Description

You can use the lookupSetCtrlValues function to look up the set control values associated with a setId in the context of a record or record group. Furthermore, you can request that the set control value be translated through a specified DVM map from the element in the PeopleSoft domain to an element in the common domain. In a typical PeopleSoft implementation the set control values represent PeopleSoft Business Unit. In such case, you would have a DVM defined for Business Unit mapping for translation.

### Parameters

<i>Parameter</i>	<i>Description</i>
SetId	The SetId value interested in lookup, as string.
LookupType	The type of lookup to perform. Valid values are 1 for record group ID and 2 for record name.
LookupName	Record group ID or record name, as string. The value should correspond to the LookupType specified.
dvmTranslate	Specify <i>True</i> if translation of set control values using the static value map (DVM) supplied is desired.
dvmMapName	The name of the static value map (DVM) to be used in translation, as string.



<i>Parameter</i>	<i>Description</i>
PsftElementName	The name of the element belonging to the PeopleSoft domain in the DVM to use in lookup, as string.
CommonElementName	The name of the element belonging to the Common domain in the DVM for which to return an equivalent value, as string.
needAnException	<i>True</i> to cause exception to occur in case set control values are not found. <i>False</i> to request an empty string in such case instead.

## Returns

The equivalent values in array of string.

## Example

This example looks up the set control values (representing PeopleSoft Business Units in this example) associated with the setID *SHARE* for the record group *VENDOR* and translates them to the common IDs for Business Unit by means of the DVM mapping *BusinessUnit*. An exception is requested if set control values are not found:

```
&oSetIdUtil = create EOTF_CORE:SetId:Functions();

    Local array of string &arrReturnValue = CreateArrayRept("", 0);

    try

        &arrReturnValue = &oSetIdUtil.lookupSetCtrlValues("SHARE", "1", "VENDOR", =>
true, "BusinessUnit", "PSFT_BU", "COMMON", true);

        catch Exception &exReturn
. . .
        {Your Exception Handling Logic Here}
. . .
    end-try;
```

## lookupSetID

### Syntax

**lookupSetID**(*LookupValue*,*dvmTranslate*,*dvmMapName*,*PsftElementName*,*CommonElementName*,  
*LookupType*,*LookupName*,*needAnException*)

## Description

You can use the `lookupSetID` function to look up the `setID` corresponding to a set control value in the context of a record or record group. Furthermore, you can request that the set control value to use for lookup be translated through a specified DVM map from an element in a common domain to the element in the PeopleSoft domain. In a typical PeopleSoft implementation the set control values represent PeopleSoft Business Units. In this case, you would have a DVM defined for Business Unit mapping translation.

## Parameters

<i>Parameter</i>	<i>Description</i>
LookupValue	Value to use in <code>setID</code> lookup, as string. Value should either be a set control value if <code>dvmTranslate</code> is false or a common element value if <code>dvmTranslate</code> is true.
dvmTranslate	Specify <i>True</i> if translation of set control values using static value map (DVM) supplied is desired.
dvmMapName.	The name of the static value map (DVM) to be used in translation, as string.
PsftElementName	Name of the element belonging to the PeopleSoft domain in the DVM for which to retrieve the equivalent set control value, as string.
CommonElementName	Name of the element belonging to the Common domain in the DVM to use in lookup, as string .
LookupType	The type of lookup to perform. Valid values are <i>1</i> for record group ID and <i>2</i> for record name.
LookupName	Record group ID or record name, as string. The value should correspond to the <code>LookupType</code> specified.
needAnException	<i>True</i> to cause exception to occur in case <code>setID</code> is not found. <i>False</i> to request an empty string in such case instead.

## Returns

The equivalent value as string.

## Example

The following code looks up the `setID` associated with the common ID for Business Unit *BUID001* in the DVM mapping *BusinessUnit* for the record group *VENDOR*. No exception is requested if `setID` is not found (empty string will be returned.)

```

&oSetIdUtil = create EOTF_CORE:SetId:Functions();

Local string &ReturnValue;

&ReturnValue = &oSetIdUtil.lookupSetId("BUID001", "BusinessUnit", "PSFT_BU", =>
    "COMMON", "1" ,"VENDOR", false);

```

---

## Xref Utility Class Methods

This section describes the Xref class methods. The methods are discussed in alphabetical order.

### LookupValue

#### Syntax

**LookupValue**(*mapName*,*referenceElementName*,*referenceValue*,*elementName*,*needAnException*)

#### Description

Locate the reference element value in a cross-reference value map, and return the equivalent value for the specified element name. This form of lookup is used to find a single reference element and return a single element value.

#### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceElementName	Name of an element in the Xref in which to look for a value, as string.
referenceValue	Value of an element in the Xref to look for, as string.
elementName	Name of the element in the Xref for which to return equivalent values, as string.
needAnException	<i>True</i> to return error messages, <i>false</i> to return blank.

#### Returns

An array of string containing the equivalent values of elementName in the Xref .

## Example

This example will look up value *&guid1* in the *UniqueGUID* element of the *&TestName* cross-reference map and return the equivalent value of element *&EBS1*.

```
Local string &returnValue = &xref.LookupValue(&TestName, &UniqueGUID, &guid1,⇒
&EBS1, True);
```

This example will look up a value that does not exist in element *&EBS1* of the *&TestName* cross-reference map to verify that the default value (blank) is returned when the *NeedAnException* parameter is false.

```
&returnValue = &xref.LookupValue(&TestName, &EBS1, "NotFound", &UniqueGUID, False);
```

## LookupValue1M

### Syntax

**LookupValue1M**( *mapName*, *referenceElementName*, *referenceValue*, *elementName*, *needAnException* )

### Description

Locate the reference element value in a cross-reference value map, and return the equivalent values of the specified element name as a named value pair (NVP) list. This form of lookup is used to find a single reference element and return one to many equivalent values for the specified element

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceElementName	Name of an element in the Xref in which to look for a value, as string.
referenceValue	Value of an element in the Xref to look for, as string.
elementName	Name of the element in the Xref for which to return equivalent values, as string.
needAnException	<i>True</i> to return error messages, <i>false</i> to return blank.

### Returns

An array of string containing the equivalent values of *elementName* in the cross-reference.

## Example

This example will look up value *&guid1* in the *UniqueGUID* element of the *&TestName* cross-reference map and return the equivalent value of element *&EBS1*.

```
Local array of string &returnValue;
```

```
&returnValue = &xref.LookupValue1M(&TestName, &UniqueGUID, &guid1, &EBS1, True);
```

This example will look up a value that does not exist in the *UniqueGUID* element of the *&TestName* cross-reference map to verify that no value is returned when the *NeedAnException* parameter is false.

```
&returnValue = &xref.LookupValue1M(&TestName, &UniqueGUID, "NotFound", &EBS1, →  
False);
```

## LookupValueNVP

### Syntax

**LookupValueNVP**( *mapName*, *referenceDomain*, *referenceNVP*, *targetDomain*, *needAnException* )

### Description

Locate the reference domain element values in a cross-reference map, and return the equivalent values of all elements in the specified domain as an NVP list. This form of lookup should be used when multiple elements exist in either the reference or return domain. All required elements in the reference domain must be included in the reference NVP list, but optional elements (qualifiers perhaps) do not have to be included. The return string will include values for all elements in the target domain as an NVP list regardless of whether they are required.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceDomain	Name of a domain in the XREF in which to look for a value, as string.
referenceNVP	Name value pairs of elements and values in the reference domain to look for, as an array of DataElement.
targetDomain	Name of the domain to return equivalent values for, as string.
needAnException	<i>True</i> to return error messages, <i>false</i> to return an NVP with the default values.

## Returns

An array of DataElement for name value pairs containing the equivalent values for the elements in the target domain, or the default values.

## Example

This example will look up value *&guid1* in the *UniqueGUID* domain of the *&TestName* cross-reference map and return the equivalent values for the *&RTK* domain.

```
Local array of EOTF_CORE:Common:DataElement &returnValue = &xref.LookupValueNVP⇒
(&TestName, &UniqueGUID, &UniqueGUIDrequestValues, &RTK, True);
```

## MarkForDelete

### Syntax

**MarkForDelete**(*mapName*,*elementName*,*elementValue*)

### Description

Delete a value in a cross-reference map when the element specified is the only element for a single domain. If multiple domains reference the element, or the element is used in a domain containing multiple primary elements, the `xref:markForDeleteNVP` function should be used instead. The values in the elements are marked as deleted. If only two mappings are in a row and one of them is marked for deletion, then the value in the other domain will also be deleted. Any element value marked for delete is treated as if the value does not exist. Therefore, you can populate the same element with the `xref:populateXRefRow` function in ADD mode. However, using the element value marked for delete as a reference value in the LINK mode of the `xref:populateXRefRow` function would cause an error.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
elementName	Name of the element in the Xref from which to delete a value, as string.
elementValue	Value of the element in the Xref to be deleted, as string.

### Returns

True if the delete succeeds .

## Example

This example will delete the *PS001* value in the *PS* element of the *Customers* cross-reference map.

```
&return=&xref.MarkForDelete("Customers", "PS", "PS001")
```

## MarkForDeleteNVP

### Syntax

**MarkForDeleteNVP**(*mapName*,*referenceDomain*,*referenceNVP*)

### Description

Delete a set of values in a cross-reference map for a specified domain. The values in the elements are marked as deleted. If only two mappings are in a row and one of them is marked for deletion, then the value in the other domain will also be deleted. Any values marked for delete are treated as if they do not exist. Therefore, you can populate the same elements with `xref:populateXRefRowNVP` function in ADD mode. However, using the values marked for delete as a reference value in the LINK mode of the `xref:populateXRefRowNVP` function would cause an error.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceDomain	Name of a domain in the Xref from which to delete values, as string.
referenceNVP	Name value pairs of elements and values in the reference domain to be deleted, as an array of DataElement.

### Returns

True if the delete succeeds.

## Example

This example deletes the specified values in the EBS domain from the Items cross-reference map.

```
Local array of EOTF_CORE:Common:DataElement &ebsNVP1 = CreateArrayRept(create EOTF_⇒
CORE:Common:DataElement(&EBS1), 1);
&ebsNVP1 [1].value = &value1;
Local boolean &return = &xref.MarkForDeleteNVP("Items", &EBS1, &ebsNVP1);
```

## PopulateValue

### Syntax

**PopulateValue**(*mapName*,*referenceElementName*,*referenceValue*,*elementName*,*elementValue*,*mode*)

### Description

Locate a reference element value in a cross-reference map and populate another element in the same data row with an equivalent value. This form of populate is used to find a single reference element and populate a single element value.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceElementName	Name of an element in the Xref in which to look for a value, as string.
referenceValue	Value of an element in the Xref to look for, as string.
elementName	Name of the element in the Xref in which to supply an equivalent value, as string.
elementValue	The equivalent value to be supplied to the element, as string.
mode	The mode in which the function populates the element. You can specify any of the following values: ADD, LINK, or UPDATE. The mode parameter values are case-sensitive and must be specified in uppercase only.

### Returns

The UniqueGUID value of the Xref row where the data was populated.

### Example

This example will locate value *&guid1* in the *UniqueGUID* element of the *&TestName* cross-reference map, and update the equivalent value of element *EBS1* to *&value1*.

```
&value1 = &value1 | "0";
&returnValue = &xref.PopulateValue(&TestName, &UniqueGUID, &guid1, &EBS1, =>
&value1, &UPDATE);
```

This example will add values *&guid3* in the *UniqueGUID* element and *&value3* in the *EBS1* element to the *&TestName* cross-reference map data.



```
Local string &guid3 = UuidGen();
Local string &value3 = "00003";
&returnValue = &xref.PopulateValue(&TestName, &EBS1, &value3, &UniqueGUID, &guid3,⇒
&ADD);
```

This example will add value *&value4* in the *EBS1* element to the *&TestName* cross-reference map data, and let the code generate a random *UniqueGUID*.

```
Local string &value4 = "00004";
&returnValue = &xref.PopulateValue(&TestName, &EBS1, &value4, &UniqueGUID, "",⇒
&ADD);
```

## PopulateValueNVP

### Syntax

**PopulateValueNVP**(*mapName*,*referenceDomain*, *DataElement referenceNVP*, *targetDomain*,*targetNVP*,  
*mode*)

### Description

Locate the reference domain element values in a cross-reference map, and populate another domain's elements in the same data row with an equivalent value. This form of populate should be used when multiple elements exist in either the reference or target domain. All required elements in the reference domain must be included in the reference NVP list, but optional elements such as qualifiers do not have to be included.

### Parameters

<i>Parameter</i>	<i>Description</i>
mapName	Name of a dynamic (cross-reference) value map definition, as string.
referenceDomain	Name of a domain in the Xref in which to look for a value, as string.
referenceNVP	Name value pairs of elements and values in the reference domain to look for, as an array of DataElement.
targetDomain	Name of the domain in which to populate equivalent values, as string.
targetNVP	Element names and their equivalent values (NVPs) to be populated in the Xref map, as an array of DataElement.
mode	The mode in which the function populates the element. You can specify any of the following values: ADD, LINK, or UPDATE. The mode parameter values are case-sensitive and must be specified in uppercase only.

## Returns

The UniqueGUID value of the Xref row where the data was populated.

## Example

This example will locate value *&guid1* in the *UniqueGUID* element of the *&TestName* cross-reference map, and update the equivalent value of element *EBS1* to *&value1*.

```
&value1 = &value1 | "0";
&ebsNVP1 [1].value = &value1;
&returnValue = &xref.PopulateValueNVP(&TestName, &UniqueGUID, &guidNVP1, &EBS, =>
    &ebsNVP1, &UPDATE);
```

This example will add values *&guid3* in the *UniqueGUID* element and *&value3* in the *EBS1* element to the *&TestName* cross-reference map data.

```
Local string &guid3 = UuidGen();
Local array of EOTF_CORE:Common:DataElement &guidNVP3 = CreateArrayRept(create =>
    EOTF_CORE:Common:DataElement(&UniqueGUID), 1);
&guidNVP3 [1].value = &guid3;
Local string &value3 = "00003";
Local array of EOTF_CORE:Common:DataElement &ebsNVP3 = CreateArrayRept(create EOTF_ =>
    CORE:Common:DataElement(&EBS1), 1);
&ebsNVP3 [1].value = &value3;
&returnValue = &xref.PopulateValueNVP(&TestName, &EBS, &ebsNVP3, &UniqueGUID, =>
    &guidNVP3, &ADD);
```

This example will add value *&value4* in the *EBS1* element to the *&TestName* cross-reference map data, and let the code generate a random UniqueGUID.

```
Local array of EOTF_CORE:Common:DataElement &guidNVP4 = CreateArrayRept(create =>
    EOTF_CORE:Common:DataElement(&UniqueGUID), 1);
Local string &value4 = "00004";
Local array of EOTF_CORE:Common:DataElement &ebsNVP4 = CreateArrayRept(create EOTF_ =>
    CORE:Common:DataElement(&EBS1), 1);
&ebsNVP4 [1].value = &value4;
&returnValue = &xref.PopulateValueNVP(&TestName, &EBS, &ebsNVP4, &UniqueGUID, =>
    &guidNVP4, &ADD);
&requestValues = CreateArrayRept(create EOTF_CORE:Common:DataElement(&EBS1), 1);
&requestValues [1].value = &value4;
```

---

## Xref Class Properties

This section discusses the Xref class properties.

## **exceptionCaught**

### **Description**

Value true when a lookup method suppresses an exception because the `needAnException` parameter was false.

## **exceptionDetails**

### **Description**

Exception object containing the detail of the exception that was caught.



## Chapter 7

# Accessing Maps Using Web Services

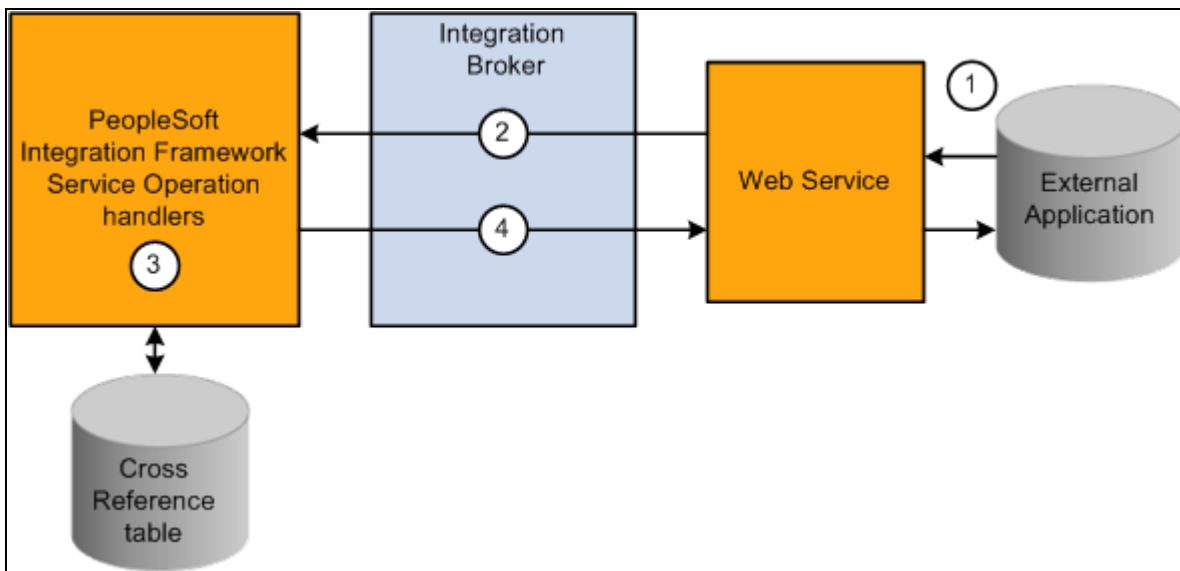
This chapter provides an overview of Application Integration Framework web services and discusses the cross-reference lookup service.

---

## Understanding Application Integration Framework Web Services

Application Integration Framework web services provide external applications a web-service-based means of accessing Application Integration Framework map information. Web services are implemented by means of the PeopleTools Integration Broker (IB) framework. The Integration Gateway web application receives all the web service requests and forwards them to the Integration Engine (application server) for processing.

This diagram illustrates an external application using the cross-reference lookup web service to look up values in an Application Integration Framework cross-reference map.



Using Application Integration Framework web service with an external application

1. The external application invokes one of the web service operations.
2. PeopleSoft Integration Broker receives the service operation and validates the WS security credentials.
3. The request is passed to the application server for processing. The application server authenticates the service operation and routes it to the respective handler. The handler runs the PeopleCode and sends the response to the Integration Gateway.

- Integration Broker sends the response to the external application.

## Cross-Reference Lookup Web Service

The service operation EOTF\_XREF\_LOOKUP.v1 is used by external systems to look up a cross-reference value. For the external system to have access to the web service, it must be published on the PeopleSoft application.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, Providing Services*.

### Example: Cross-Reference Map Definition

In this example, the third-party system requests a cross-reference lookup for payment terms. This page shows elements in the value map PaymentTermsGUID:

Elements Domains

Map Name: PaymentTermGUID


Type: Cross-reference (dynamic)

\*Description: Payment Terms Cross-Reference

Delete

Comments:

Assign Elements to Maps

Customize | Find |  First 1-3 of 3 Last

Order	*Element Name	*Data Type	Length	Required		
1	UniqueGUID	String	36	<input checked="" type="checkbox"/>	+	-
2	SETID	String	5	<input checked="" type="checkbox"/>	+	-
3	PYMNT_TERMS_CD	String	5	<input checked="" type="checkbox"/>	+	-

Domain value map used in example

Three elements are defined: UniqueGUID, SETID, and PYMNT\_TERMS\_CD.

This page shows the domains that are defined for the domain value map:

**Elements** **Domains**

**Map Name:** PaymentTermGUID **Type:** Cross-reference (dynamic)

**Assign Domains to Value Maps** Find | View 1 First 1-2 of 2 Last

\*Domain Name: PSFT ☒ Is Unique

**Assign Elements to the Domain** Customize | Find | View All | First 1-2 of 2 Last

\*Element Name

1 SETID

2 PYMNT\_TERMS\_CD

---

Domain Name: UniqueGUID ☒ Is Unique

**Assign Elements to the Domain** Customize | Find | View All | First 1 of 1 Last

Element Name

1 UniqueGUID

Value Map - Domains page used in this example

Two domains are defined for this map:

1. *PSFT* represents the PeopleSoft application which contains the elements SETID and PYMNT\_TERMS\_CD.
2. UniqueGUID is the domain used by the third party; it contains the UniqueGUID element.

### Example: Web Service Request and Response

This is a sample request to obtain the PeopleSoft values for a common GUID value:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xref="http://xmlns.oracle.com/Enterprise/tools/schema/xrefLookupValue.v1">
  <soapenv:Header/>
  <soapenv:Body>
    <xref:xrefLookupValue needFault="true">
      <xref:mapname>PaymentTermGUID</xref:mapname>
      <xref:Values>
        <xref:domain>UniqueGUID</xref:domain>
        <!--1 or more repetitions:-->
        <xref:element name="UniqueGUID">9d266732-90e2-11dd-a062-96c8921a7858<=>
      </xref:element>
      </xref:Values>
      <xref:targetDomain>PSFT</xref:targetDomain>
    </xref:xrefLookupValue>
  </soapenv:Body>
</soapenv:Envelope>
```

This is the response:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:⇒
soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org⇒
/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <xrefResponse xmlns="http://xmlns.oracle.com/Enterprise/tools/schema/xref⇒
Response.v1">
      <status>Success</status>
      <responseValues>
        <domain>PSFT</domain>
        <element name="SETID">SHARE</element>
        <element name="PYMNT_TERMS_CD">CD01</element>
      </responseValues>
    </xrefResponse>
  </soapenv:Body>
</soapenv:Envelope>

```



## Appendix A

# Application Integration Framework Example

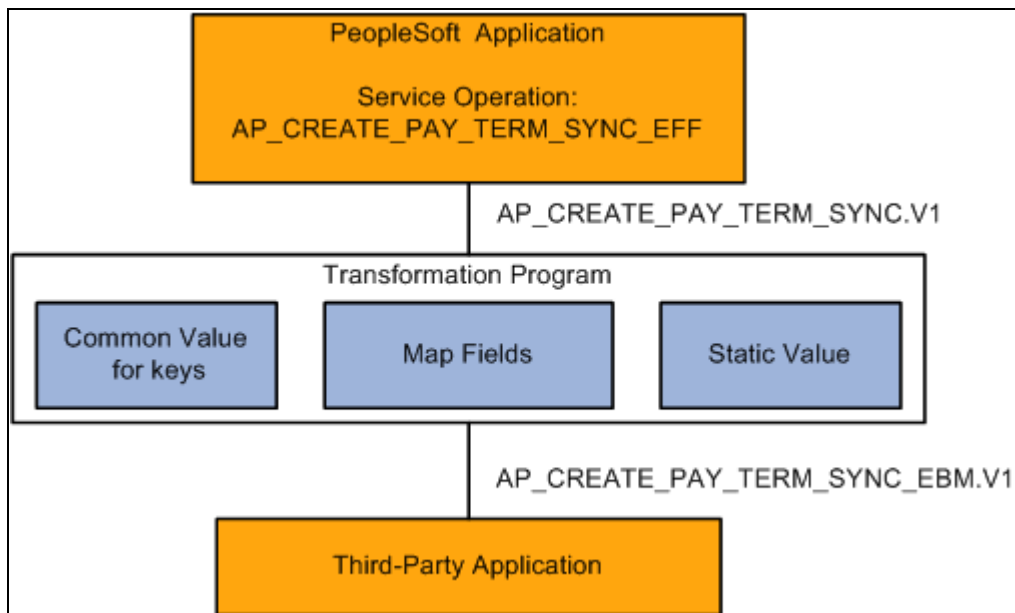
This appendix provides an overview of an example of an Application Integration Framework transformation and discusses how to:

- Define a dynamic value map
- Define and populate a static value map
- Use the XSLT extension function in the transformation program
- Update the service operation routing

---

## Example Overview

This diagram represents the scenario for this application:



Payment terms integration flow

In this example, the PeopleSoft application updates the third-party application every time a new payment term is added. This transformation requires mapping the PeopleSoft ABM elements to the EBM elements, as well as mapping keys and static values. This example covers the following data translations within Application Integration Framework:

- The combination of the fields SETID and PYMNT\_TERMS\_CD is used as the key in the PeopleSoft application. This value needs to be assigned a common GUID for the EBM.
- The PeopleSoft application uses a 3-character code for language. The third-party system uses a numeric code.

This integration requires a transformation that:

- Maps the message structure.
- Creates a common value for the key fields.
- Translates the data values for static fields that differ.

---

## Defining a Dynamic Value Map

To create a new dynamic value map:

1. Select Enterprise Components, Integration Definitions, Transformation Framework, Define Value Maps.
2. Select the Add a New Value tab.
3. Enter *PaymentTermsGUID* for the map name.
4. Select *Cross-reference (Dynamic)* for the map type.
5. Click Add.

The map will contain the element UniqueGUID.

6. Add two additional elements SETID and PYMNT\_TERMS\_CD.

Elements Domains

Map Name: PaymentTermGUID

Type: Cross-reference (dynamic)

\*Description: Payment Terms Cross-Reference

Delete

Comments:

Assign Elements to Maps

Customize | Find | First 1-3 of 3 Last

Order	*Element Name	*Data Type	Length	Required		
1	UniqueGUID	String	36	<input checked="" type="checkbox"/>	+	-
2	SETID	String	5	<input checked="" type="checkbox"/>	+	-
3	PYMNT_TERMS_CD	String	5	<input checked="" type="checkbox"/>	+	-

Domain value map used in example

Three elements are defined: UniqueGUID, SETID, and PYMNT\_TERMS\_CD.

To add the domains:

1. Select the Domains tab.  
The UniqueGUID domain appears.
2. Click the Add a new row icon to create another domain.
3. Enter *PSFT* for the domain name.
4. Select *SETID* for the first element.
5. Click the Add a new row icon to add another row.
6. Select *PYMNT\_TERMS\_CD* for the second element.
7. Save the page.

The screenshot displays the 'Value Map-Domains' configuration page. At the top, the 'Map Name' is 'PaymentTermGUID' and the 'Type' is 'Cross-reference (dynamic)'. The page is divided into two sections for domain configuration.

**Domain 1: PSFT**

- Domain Name:** PSFT
- Is Unique:** ☒
- Assign Elements to the Domain:**
  - 1 SETID
  - 2 PYMNT\_TERMS\_CD

**Domain 2: UniqueGUID**

- Domain Name:** UniqueGUID
- Is Unique:** ☒
- Assign Elements to the Domain:**
  - 1 UniqueGUID

Value Map-Domains page used in this example

Two domains are defined for this map:

- PSFT represents the PeopleSoft application and contains the elements SETID and PYMNT\_TERMS\_CD.
- UniqueGUID is the domain used by the third party; it contains the UniqueGUID element.

## Defining and Populating a Static Value Map

To create a new domain value map:

1. Select Enterprise Components, Integration Definitions, Transformation Framework, Define Value Maps.
2. Select the Add a New Value tab.
3. Enter *LanguageCodeDVM* for the map name.
4. Select *Domain Value Map (DVM)* for the map type.
5. Click Add.

The map will contain the element UniqueGUID.

6. Add two additional elements, LANGUAGE\_CD and COMMON.

Elements

Domains

Map Name:

LanguageCodeDVM

Type:

Domain Value Map (static)

\*Description:

Language Code DVM

Comments:

Assign Elements to Maps

Customize | Find |

First

1-3 of 3

Last

Order	*Element Name	*Data Type	Length	Required		
1	UniqueGUID	String	36	<input checked="" type="checkbox"/>		
2	LANGUAGE_CD	String	3	<input checked="" type="checkbox"/>		
3	COMMON	String	32	<input checked="" type="checkbox"/>		

Domain value map for language code

To add the domains:

1. Select the Domains tab.  
The UniqueGUID domain appears.
2. Click the Add a new row icon to create another domain.
3. Enter *PSFT* for the domain name.
4. Select *LANGUAGE\_CD* for the first element.
5. Click the Add a new row icon in the header to add another domain.
6. Enter *AIA* for the domain name.
7. Select *COMMON* for the first element.
8. Save the page.

**Elements** **Domains**

**Map Name:** LanguageCodeDVM **Type:** Domain Value Map (static)

**Assign Domains to Value Maps** Find | View 1 First 1-3 of 3 Last

**\*Domain Name:** AIA ☒ **Is Unique** + -

**Assign Elements to the Domain** Customize | Find | View All | First 1 of 1 Last

	*Element Name		
1	COMMON	+	-

**\*Domain Name:** PSFT ☒ **Is Unique** + -

**Assign Elements to the Domain** Customize | Find | View All | First 1 of 1 Last

	*Element Name		
1	LANGUAGE_CD	+	-

**Domain Name:** UniqueGUID ☒ **Is Unique** + -

**Assign Elements to the Domain** Customize | Find | View All | First 1 of 1 Last

	Element Name		
1	UniqueGUID	+	-

Domain value map domains for language code

Three domains are defined: AIA, PSFT and UniqueGUID.

To populate the DVM:

1. Select Enterprise Components, Integration Definitions, Transformation Framework, Populate Domain Value Maps.
2. Select *LanguageCodeDVM*.
3. In the LANGUAGE\_CD column, enter a data value as defined in the PeopleSoft (PSFT) domain.
4. In the corresponding COMMON column, enter the value to be used for the AIA domain.
5. Add as many rows as necessary to map all of the static values.
6. Save the page.

Populate Domain Value Maps				
Map Name: LanguageCodeDVM		Description: Language Code DVM		
Assign Data to Value Maps				Customize   Find
	LANGUAGE_CD	COMMON	First	1 of 1 Last
1	ENG	500	+	-

Populate Domain Value Maps page

## Using the XSLT Extension Function in the Transformation Program

Create a transformation program that maps the fields in the PeopleSoft ABM message to the corresponding fields in the EBM message. You will then use the XSLT functions to map the data values.

This section provides sample coding for sections of the transform application engine program for:

- Key value transformation
- Domain value transformation

### Key Value Transformation

This example shows the elements in XML in the ABM that need to be translated, the code for the translation, and the resulting elements in the EBM.

#### *Key Elements for Translation in PeopleSoft ABM*

Elements in XML message:

```
<MsgData>
  <Transaction>
    <PYMT_TRMS_HDR class="R">
      <SETID IsChanged="Y">SHARE</SETID>
      <PYMNT_TERMS_CD IsChanged="Y">DIT91</PYMNT_TERMS_CD>
```

#### *XSLT Code*

This snippet of the XSLT code shows the transform XSLT in the application engine program:

1. Create a new variable by concatenating SETID and PYMT\_TERMS\_CD:

```
<corecom:ApplicationObjectKey>
  <corecom:ID>
    <xsl:attribute name="schemeID">
      <xsl:text disable-output-escaping="no">PSFT</xsl:text>
    </xsl:attribute>
    <xsl:attribute name="schemeAgencyID">
      <xsl:text disable-output-escaping="no">PSFT_COMMON</xsl:text>
    </xsl:attribute>
    <xsl:variable name="NamedValuePair" select='concat("&lt;SETID&gt;",PYMT_TRMS_>
HDR/SETID,"&lt;/SETID&gt;&lt;PYMNT_TERMS_CD&gt;",PYMT_TRMS_HDR/PYMNT_TERMS_>
CD,"&lt;/PYMNT_TERMS_CD&gt;")' />
```

2. Use the generate-guid utility to generate a unique GUID:

```
<xsl:variable name="CommonGuid" select='concat("&lt;UniqueGUID&gt;",utility:=>
generate-guid(),"&lt;/UniqueGUID&gt;")' />
```

---

**Note.** This step is not necessary. If the GUID does not exist, the appropriate xref:populate function automatically generates the unique GUID.

---

3. Use the xref:populateValueNVP function to add the new GUID to the cross-reference:

```
<xsl:variable name="GUIDAdd" select='xref:populateValueNVP("PaymentTermGUID","PSFT">
,$NamedValuePair,"UniqueGUID",$CommonGuid,"ADD")' />
  <xsl:call-template name="Process-GUID">
    <xsl:with-param name="returnValue">
      <xsl:value-of select="$GUIDAdd" />
    </xsl:with-param>
    <xsl:with-param name="statusDelimiter"></xsl:with-param>
  </xsl:call-template>
</corecom:ID>
```

4. Insert the values in the EBM message:

```
<corecom:ContextID>
  <xsl:attribute name="schemeID">
    <xsl:text disable-output-escaping="no">SETID</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="schemeAgencyID">
    <xsl:text disable-output-escaping="no">PSFT</xsl:text>
  </xsl:attribute>
  <xsl:value-of select="PYMT_TRMS_HDR/SETID" />
</corecom:ContextID>
<corecom:ContextID>
  <xsl:attribute name="schemeID">
    <xsl:text disable-output-escaping="no">Payment Terms Code</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="schemeAgencyID">
    <xsl:text disable-output-escaping="no">PSFT</xsl:text>
  </xsl:attribute>
  <xsl:value-of select="PYMT_TRMS_HDR/PYMNT_TERMS_CD" />
</corecom:ContextID>
```

### ***Transformed Elements in EBM Message***

The transformed EBM message contains the common GUID value:



```
<corecom:ApplicationObjectKey>
  <corecom:ID schemeAgencyID="PSFT_COMMON" schemeID==>
    "PSFT">b15f3c34-72bc-11dd-b7dd-aaf7c4308a71</corecom:ID>
  <corecom:ContextID schemeAgencyID="PSFT" schemeID==>
    "SETID">SHARE</corecom:ContextID>
  <corecom:ContextID schemeAgencyID="PSFT" schemeID==>
    "Payment Terms Code">DIT91</corecom:ContextID>
```

## Domain Value Transformation

This example shows the domain value elements in XML in the ABM that need to be translated, the code for the translation, and the resulting elements in the EBM.

### *Domain Value Elements for Translation in PeopleSoft ABM*

Here is the element for language in PeopleSoft ABM message:

```
<<PSCAMA class="R">
  <LANGUAGE_CD>ENG</LANGUAGE_CD>
```

### *XSLT Code*

This snippet of the XSLT code shows the dvm lookup in the transform application engine program:

```
<xsl:variable name="MsgLang" select='dvm:lookup-dvm ("LanguageCodeDVM"=>
, "LANGUAGE_CD", //MsgData/Transaction/PSCAMA/LANGUAGE_CD, "COMMON"=>
, //MsgData/Transaction/PSCAMA/LANGUAGE_CD, false())' />
<xsl:variable name="BaseLang" select='dvm:lookup-dvm ("LanguageCodeDVM"=>
, "LANGUAGE_CD", //MsgData/Transaction/PSCAMA/BASE_LANGUAGE_CD, "COMMON", =>
//MsgData/Transaction/PSCAMA/BASE_LANGUAGE_CD, false())' />
<xsl:attribute name="languageCode">
  <xsl:value-of select="substring-after($MsgLang, ';' )" />
</xsl:attribute>
```

### *Transformed Elements in EBM Message*

Here is the translated elements in the resulting EBM message:

```
<?xml version="1.0"?>
<paytermcreate:CreatePaymentTermEBM languageCode="500" xmlns:corepaymenttermcust=>
"http://xmlns.oracle.com/EnterpriseObjects/Core/Custom/EBO/PaymentTerm/V1">
  xmlns:paytermcreate="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/Create=>
PaymentTermEBM/V1">
  <corecom:EBMHeader languageCode="500" xmlns:corecom="http://xmlns.oracle.com=>
/EnterpriseObjects/Core/Common/V2">
    <corecom:Sender>
      <corecom:ID>E900B20</corecom:ID>
    </corecom:Sender>
  </corecom:EBMHeader>
```

## Updating the Service Operation Routing

To update the service operations routing:

- 1. Select Integration Broker, Integration Setup, Service Operations.
- 2. Select the service operation that you need to update.
- 3. Select the Routing tab.
- 4. Either enter a new routing or click the link for an existing routing.
- 5. If it is a new routing, enter the routing information.
- 6. Access the Parameters page.

<b>Message.Ver into Transform 1</b>	This is the PeopleSoft ABM message.
<b>Transform Program 1</b>	This is the transformation program created for this integration.
<b>Message.Ver out of Transforms</b>	This is the EBM message that will be sent to the third party.

- 7. Save the routing.

Routing DefinitionsParametersConnector Properties

Routing Name:AP\_CREATE\_PAY\_TERM\_SYNC\_EFF

Service Operation:AP\_CREATE\_PAY\_TERM\_SYNC\_EFF

Service Operation Version:v1

Sender Node:E900B20

Receiver Node:PSFT\_XOUTBND

Parameters

Type:Outbound Request

External Alias:CreatePaymentTerm

[Alias References](#)

Message.Ver into Transform 1:AP\_CREATE\_PAY\_TERM\_SYNC\_EFF.V1

Transform Program 1:AP\_PT\_ADD\_XF

Transform Program 2:

Message.Ver out of Transforms:AP\_CREATE\_PAY\_TERM\_SYNC\_EBM.V1

Service Operations Routing - Parameters page

# Index

## A

ABM vii  
AIA 4  
Application Integration Architecture 4

## C

cross-reference functions 33  
cross-reference map 7

## D

DVM map 6

## E

EBM vii  
EBO vii  
EBS vii

## F

framework 3

## G

generate-guid 46

## L

lookup-dvm 45  
lookupSetCtrlValues 47, 56  
lookupSetID 57  
lookupValue 43  
LookupValue 52, 59  
LookupValueIM 53, 60  
lookupValueNVP 44  
LookupValueNVP 61  
LookupVaueNVP 54  
lookupXRef 40  
lookupXRefNVP 41

## M

maps

value maps 6  
markForDelete 38  
MarkForDelete 62  
markForDeleteNVP 39  
MarkForDeleteNVP 63

## P

PIP 4  
PopulateValue 64  
Populate ValueNVP 65  
populateXRefRow 33  
populateXrefRowNVP 36

## S

SOA 4

## W

web service 69

## X

xref map 7

