ORACLE
PeopleSoft

PeopleSoft 9.2: Application Integration Framework

March 2018

ORACLE

Contents

Preface

PeopleSoft Enterprise Components Related Links

PeopleSoft Information Portal

My Oracle Support

PeopleSoft Training from Oracle University

Chapter 1

Getting Started with Application Integration Framework

Understanding the Application Integration Framework

This product documentation 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.

See *PeopleTools: PeopleSoft Integration Broker*

Define Value Maps

To define value maps, perform the following steps:

Step	***Reference***
Define value maps	See Define Value Maps.

Create Application Engine Transform Programs

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

Step	Reference
Create application engine program	See *PeopleTools: PeopleSoft Integration Broker,* Applying Filtering, Transformation and Translation, Defining Transform Programs.
Code the XSLT step	See *PeopleTools: PeopleSoft Integration Broker,* 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:

Step	Reference
Update the routing	See *PeopleTools: PeopleSoft Integration Broker,* Creating Transform Programs and Updating Service Operations, Updating Service Operation Routing.

Chapter 2

Understanding Application Integration Framework

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.

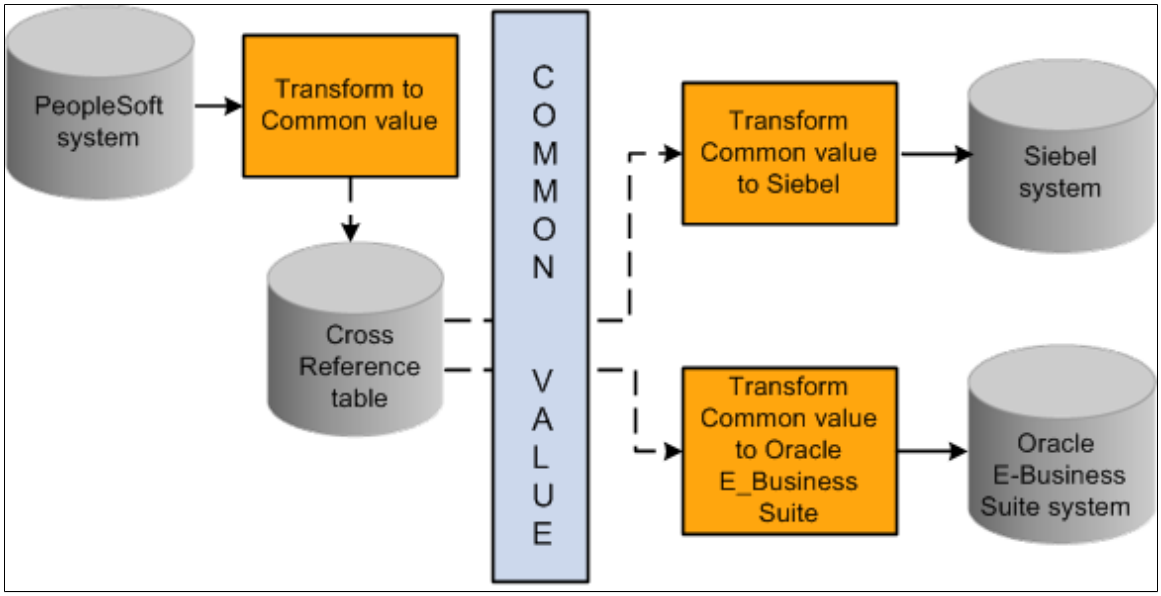
Application Integration Framework Integrations

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.

Image: Common value mapping for outgoing message

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



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:

Use Case	*PeopleSoft Integration Broker*	*AIA Middleware*	*Third Party*
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
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.

XREF	*DVM*
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.

Maps A static object for which the mapping needs to be maintained, for example, currency code or country code.

Element The unit of data in the local or remote message upon which an irreducible transanimation operates. An irreducible transanimation 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.

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

Note: Multiple domains can be maintained for a map.

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

Element 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.

Application Class methods 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.

XPath extension functions	XPath extension functions are used in the XSLT steps in application engine transform programs to invoke XSLT transforms using the TransformEx PeopleCode API.
Web services	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:

Class	***Activity***	***Form***	***Description***
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 map transformations 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.

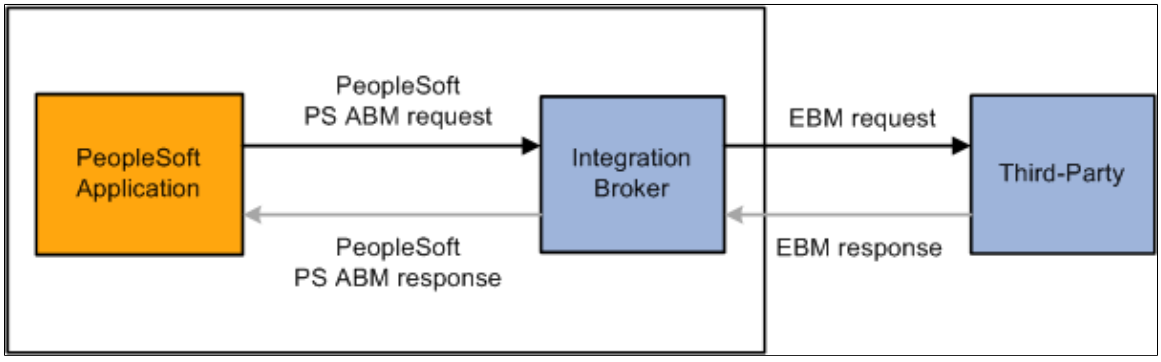
Use case includes:

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

Outbound Request or Post to a Third Party

Image: Outbound request to a third party without AIA middleware

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



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.

Transaction	***Action***
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 **populate XRef** XPath extension function to put the new common key and the corresponding PeopleSoft keys into the PeopleSoft cross-reference framework.

Transaction	*Action*
Dynamic key lookup	If the transaction is not an add-request, the transform looks up the common key using the appropriate form of the **lookup XRef** 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 **deleteXRef** 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 **lookupDVM** 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.

Transaction	*Action*
Asynchronous request-reply operation using dynamic key value	Requires a transform program 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 using dynamic key value	Lookup is not required because the PeopleSoft application already knows the PeopleSoft keys from the initial request.

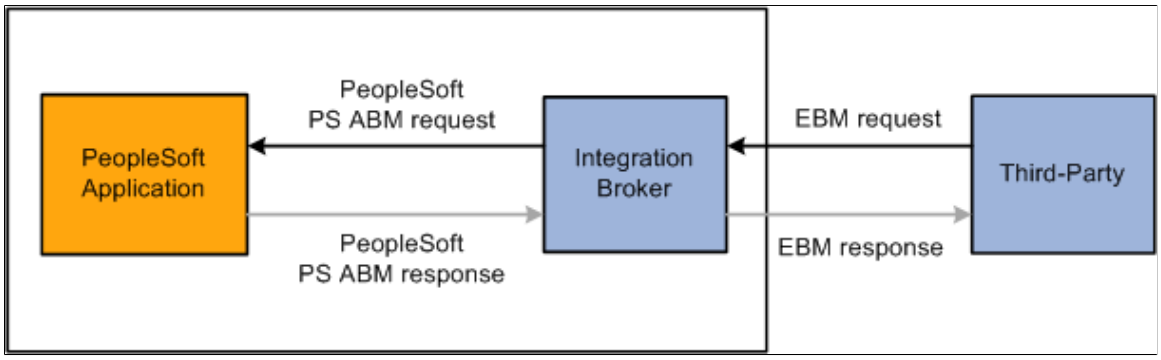
Transaction	*Action*
Asynchronous request-reply operation using DVM value	Requires a transform program to look up the PeopleSoft values using the appropriate form of the **lookupDVM** XPath extension function with the common value as input. The PeopleSoft keys are then put into the PeopleSoft ABM.

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

Inbound Request or Post from a Third Party

Image: Inbound request from third party without AIA middleware

This diagram illustrates an inbound request or post from a third party.



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:

Transaction	*Action*
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 **lookup XRef** 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 **lookup XRef** API with the common key as input.

Transaction	*Action*
DVM lookup	The transform on the inbound routing looks up the PeopleSoft values using the appropriate form of the **lookupDVM** 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.

Transaction	*Action*
Dynamic key add request	If the transaction is an add-request, it creates the PeopleSoft keys and then uses the appropriate form of the **populate XRef** 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:

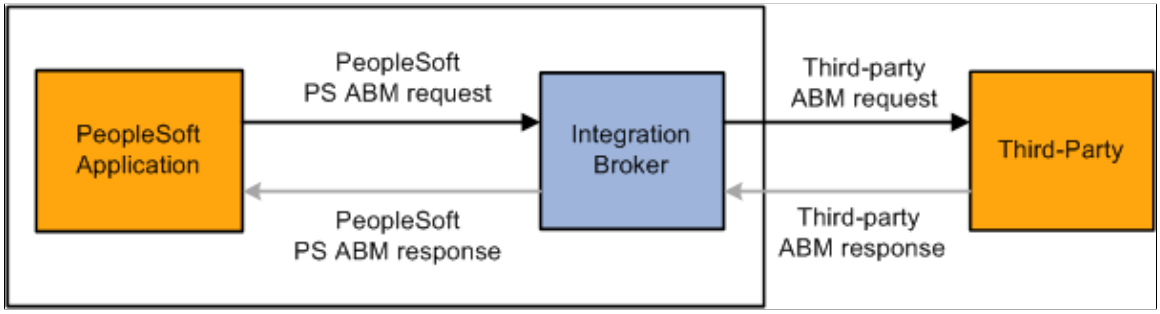
- Outbound request or post to a third party.

- Inbound request or post from a third party.

Outbound Request or Post to a Third Party

Image: Outbound point-to-point request to a third party

This diagram illustrates an outbound point-to-point request or post 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.

Transaction	***Action***
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 **populate XRef** 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 **lookup XRef** 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 **deleteXRef** XPath extension function with the PeopleSoft keys as input.

Transaction	*Action*
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 **lookupDVM** 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.

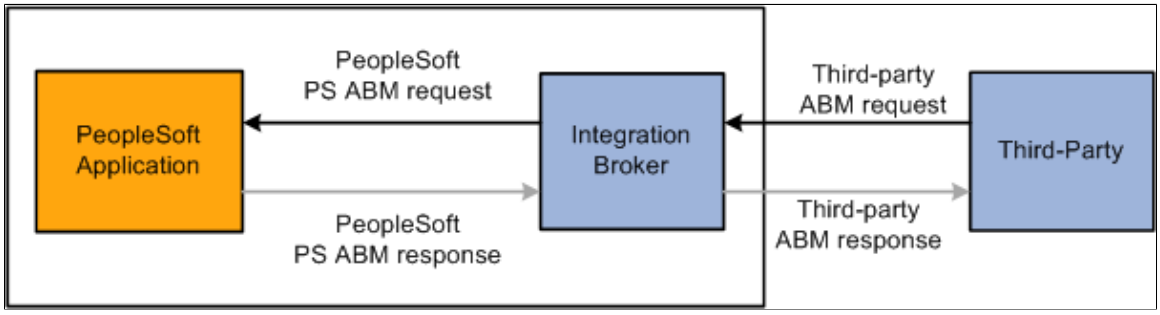
Transaction	*Action*
Asynchronous request-reply operation using dynamic key value	Requires a transform program to look up the PeopleSoft keys using the appropriate form of the **lookupXRef** XPath extension function with the third-party key as input. The PeopleSoft keys are then put into the PeopleSoft 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 **lookupDVM** 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

Image: Inbound point-to-point request from a third party

This diagram illustrates an inbound point-to-point request or post 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:

Transaction	***Action***
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 **lookup XRef** 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 **lookupDVM** 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.

Transaction	*Action*
Dynamic key add request	If the transaction is an add-request, it creates the PeopleSoft keys and then uses the appropriate form of the **populate XRef** 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:

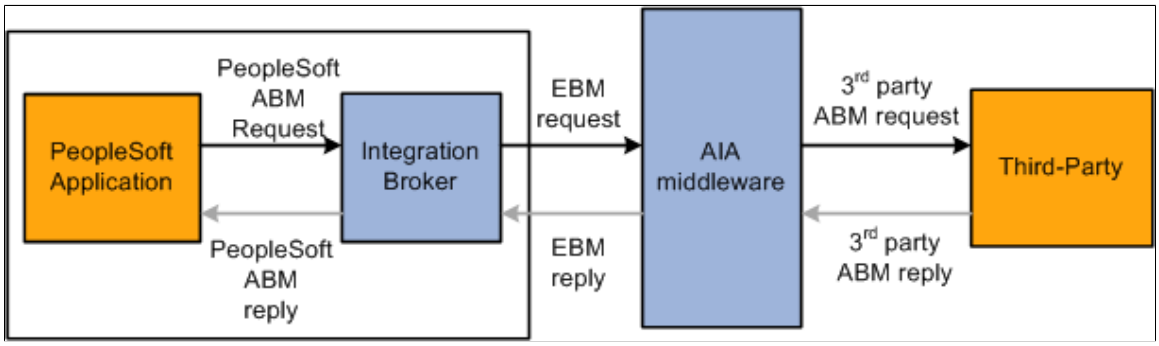
- Outbound request or post to a third party.
- Inbound request or post from a third party.

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.

Image: Outbound request to a third party that uses AIA middleware

This diagram illustrates an outbound request or post 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.

Transaction	***Action***
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 **populateXRef** 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 **lookupXRef** 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 **deleteXRef** XPath extension function with the PeopleSoft keys as input.

Transaction	*Action*
DVM lookup	If the transaction uses a domain value map, the transform looks up the PeopleSoft values using the appropriate form of the **lookupDVM** 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.

Transaction	*Action*
Dynamic key lookup	Transform looks up the PeopleSoft keys using the appropriate form of the **lookup XRef** 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 **deleteXRef** 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.

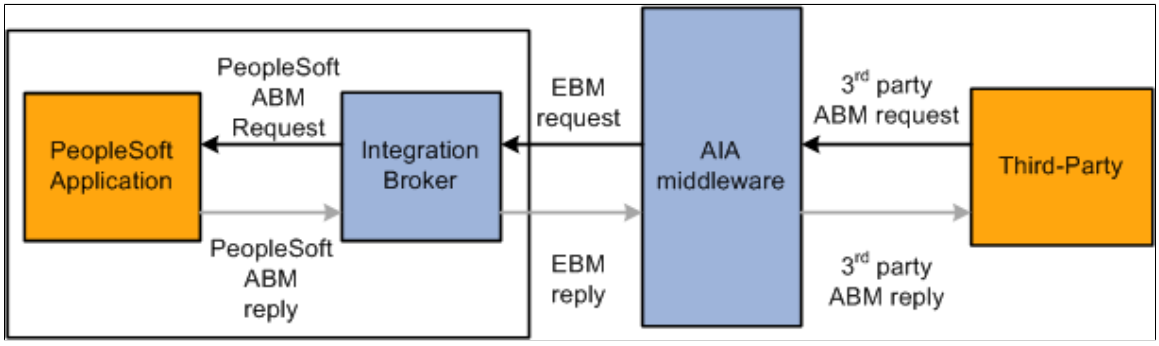
4. The PeopleSoft ABM reply is returned to the originating PeopleSoft application.

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.

Image: Inbound request from a third party using AIA middleware

This diagram illustrates an inbound request or post 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.

Transaction	*Action*
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 **lookupXRef** 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

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:

PeopleSoft	***UniqueGUID***	***External System***
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:

PeopleSoft	***UniqueGUID***	***External System***
PS	UniqueGUID	EXT

PeopleSoft	*UniqueGUID*	*External System*
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:

PeopleSoft	*PeopleSoft*	*UniqueGUID*	*External System*
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:

PeopleSoft	*Common*	*External System*
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:

PeopleSoft	*UniqueGUID*	*External System 2*
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:

PeopleSoft	*PeopleSoft*	*UniqueGUID*	*External System 1*	
Business Unit	Chartfield	UniqueGUID	Segment	Ledger
US001	ACCOUNT	<guid1>	SEGMENT1	Ledger1

PeopleSoft	*PeopleSoft*	*UniqueGUID*	*External System 1*	
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:

EXT (Qualifier 1) *PSFT (Qualifier 1)*	*EXT (Qualifier 2)* *PSFT (Qualifier 2)*	*Common*	*EXT*	*PSFT*
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 Map Options

This section discusses how to define value map options.

Page Used to Define Value Map Options

Page Name	*Definition Name*	*Usage*
Value Map Options Page	EOTF_MAP_OPTIONS	Use this page to select default values for the options available on value map definitions.

Value Map Options Page

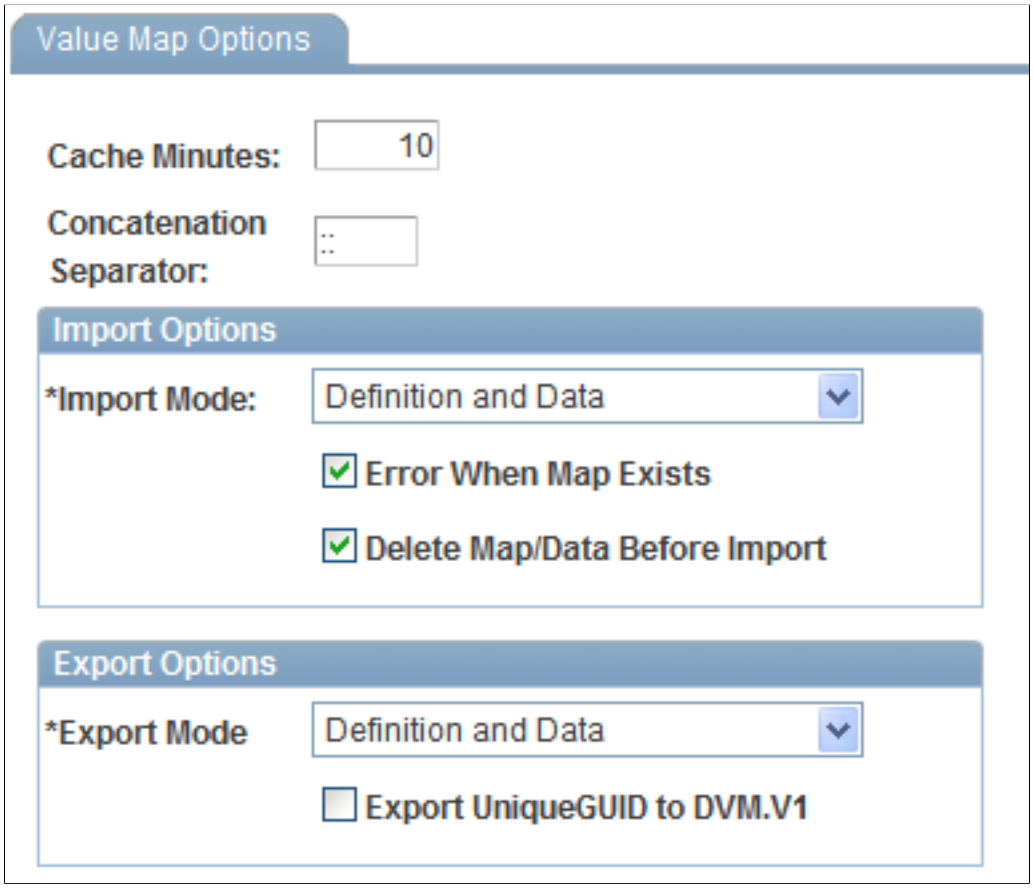
Use the Value Map Options page (EOTF_MAP_OPTIONS) to use this page to select default values for the options available on value map definitions.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Define Map Options

Image: Value Map Options page

This example illustrates the fields and controls on the Value Map Options page. You can find definitions for the fields and controls later on this page.



Cache Minutes The cache for map definitions and DVM data uses this value to determine the stale datetime. Once this time is exceeded, the cache is refreshed before use.

Concatenation Separator

Indicates the separator for concatenation when importing or exporting data. For V1 (Fusion Middleware FMW) schemas, if values are concatenated from multiple element domains into a single column during export or unconcatenated during an import of data then this separator value is used.

Import Options

These options are defaulted into the import component and any maps that are created. The import process uses the map specific options.

Import Mode

Allows a user to import a definition, alter the definition via the define value maps component, and subsequently import data only. This gives users the ability to rename elements, or perhaps specify multiple elements for a domain prior to importing data.

Valid values are:

- Definition Only

  Allows users to move definitions from one environment to another without the data.

- Definition and Data

  Imports both the definition and associated data.

- Values Only

  Allows users to import data using the map definition in the database and not the definition in the file.

Error When Map Exists

Select this check box to prevent maps from being wiped out accidentally if a user tries to import a map with an existing name. The default value is selected.

Delete Map/Data Before Import

Select this check box to delete existing definition and data prior to import (default). If this check box is cleared, append/merge is used. Append/merge allows a user to add elements, domains or data to an existing map without deleting existing data.

Note: This option is for advanced users only as they must understand the implications.

Export Options

These options are defaulted into maps that are created. The export process uses map specific options.

Export Mode

Valid options are:

- Definition Only

  Allows users to move definitions from one environment to another without the data.

This is the default for XREF.

- Definition and Data

  Exports both the definition and associated data.

  This is the default for DVM.

Export UniqueGUID to DVM.V1 Select to remove PeopleSoft required UniqueGUID domain from the DVM when moving maps to Fusion Middleware (FMW).

Defining Value Maps

This section discusses how to define value maps.

Pages Used to Define Value Maps

Page Name	*Definition Name*	*Usage*
Define Value Map - Elements Page	EOTF_DEFINE_MAPS	Assign elements to the map.
Define options for a value map Page	EOTF_MAP_OPT_SEC	Set the map level options. For new maps, the option values will default to the defined system level option values.
Define Value Maps - Domains Page	EOTF_DEFINE_MAPS2	Assign elements to domains.

Define Value Maps search Page

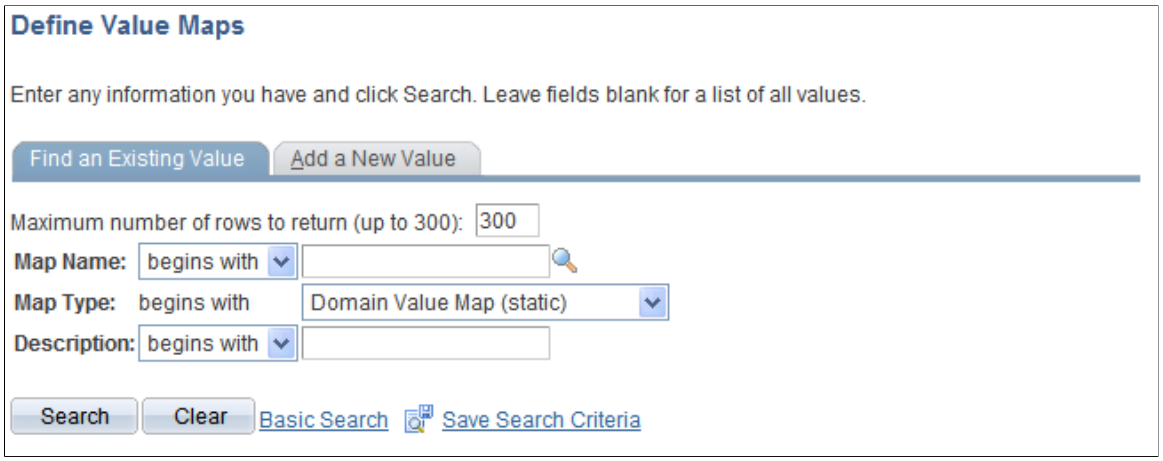
Use the Define Value Maps search page to add a new value.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps

Image: Define Value Maps search page: Add a New Value tab

This example illustrates the fields and controls on the Define Value Maps search page: Add a New Value tab. You can find definitions for the fields and controls later on this page.



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.

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

Define Value Map - Elements Page

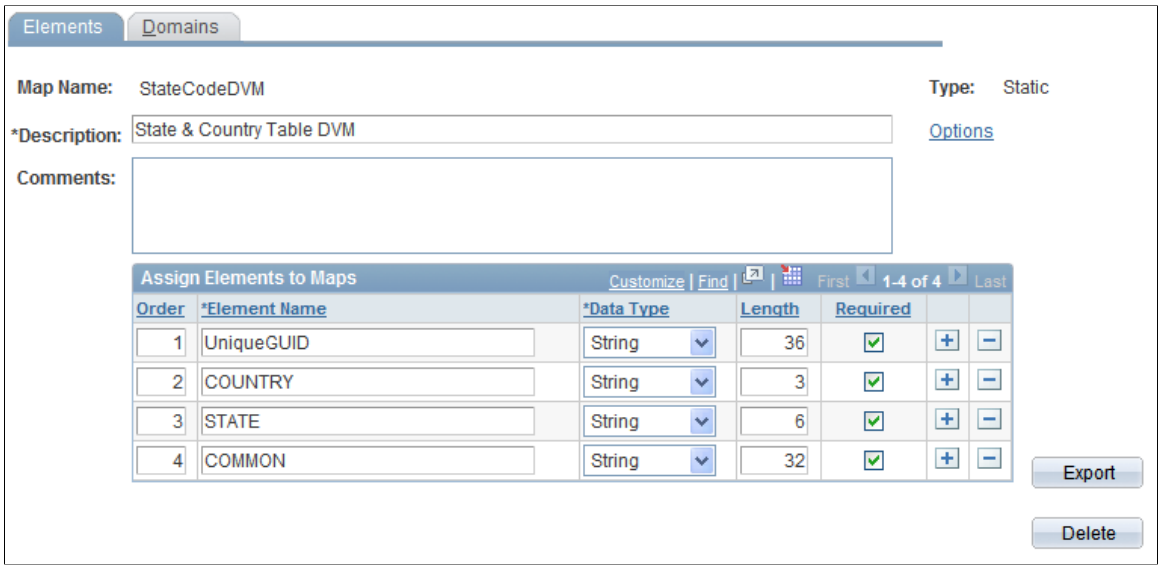
Use the Define Value Maps - Elements page (EOTF_DEFINE_MAPS) to assign elements to the map.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps > Elements

Image: Define Value Maps - Elements page

This example illustrates the fields and controls on the Define Value Maps - Elements page. You can find definitions for the fields and controls later on this page.



Description Enter a description for the value map.

Options Use this link to modify the options for this map. The link will take you to the Define options for a value map page, where you can set up the options, as well as delete the existing cache for the map. When you click OK, the options will be applied to the map.

For new maps, the option values will default to the defined system level option values.

See Value Map Options Page.

Comments Enter comments for the value map.

Order The common element is always assigned order 1. All other elements must be assigned an order of 2 or higher.

Element Name 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.

For dynamic maps, the common element must be a unique guid.

For static maps, the common element can be assigned as required.

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.
Export	This button allows the user to publish the map definition and data in an XML format using the options defined for the value map. See Exporting Value Maps.
Delete	Use this button to delete the value map.

Define options for a value map Page

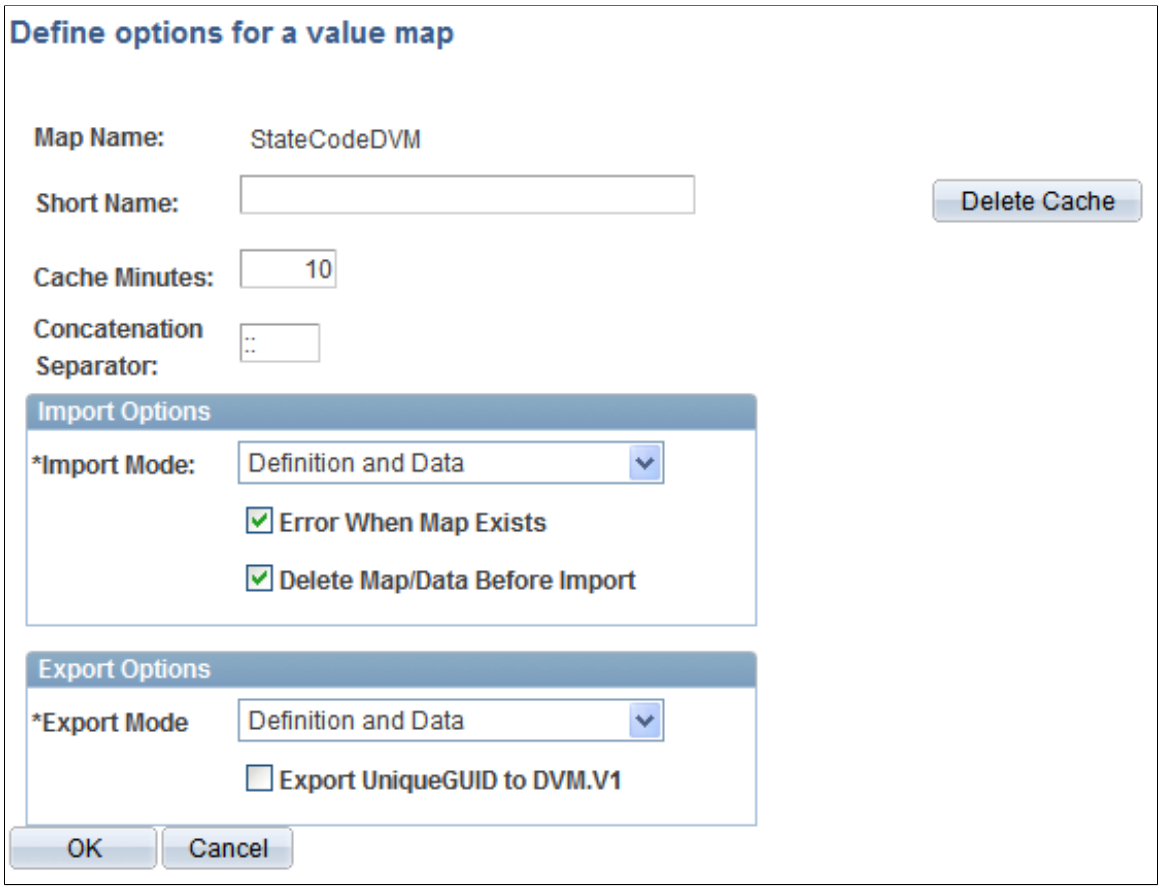
Use the Define options for a value map page (EOTF_MAP_OPT_SEC) to set the map level options. For new maps, the option values will default to the defined system level option values.

Navigation

Click the Options link on the Define Value Maps - Elements page.

Image: Define options for a value map page for a specific map

This example illustrates the fields and controls on the Define options for a value map page for a specific map. You can find definitions for the fields and controls later on this page.



This page is similar to Define options for a value map used to define the default options, the difference is:

- The options apply to this specific map.
- A Delete Cache button is available to delete the existing cache for the map.

When you click the OK button, the options will be applied to the map and you will be returned to the Define Value Maps page.

See Value Map Options Page.

Define Value Maps - Domains Page

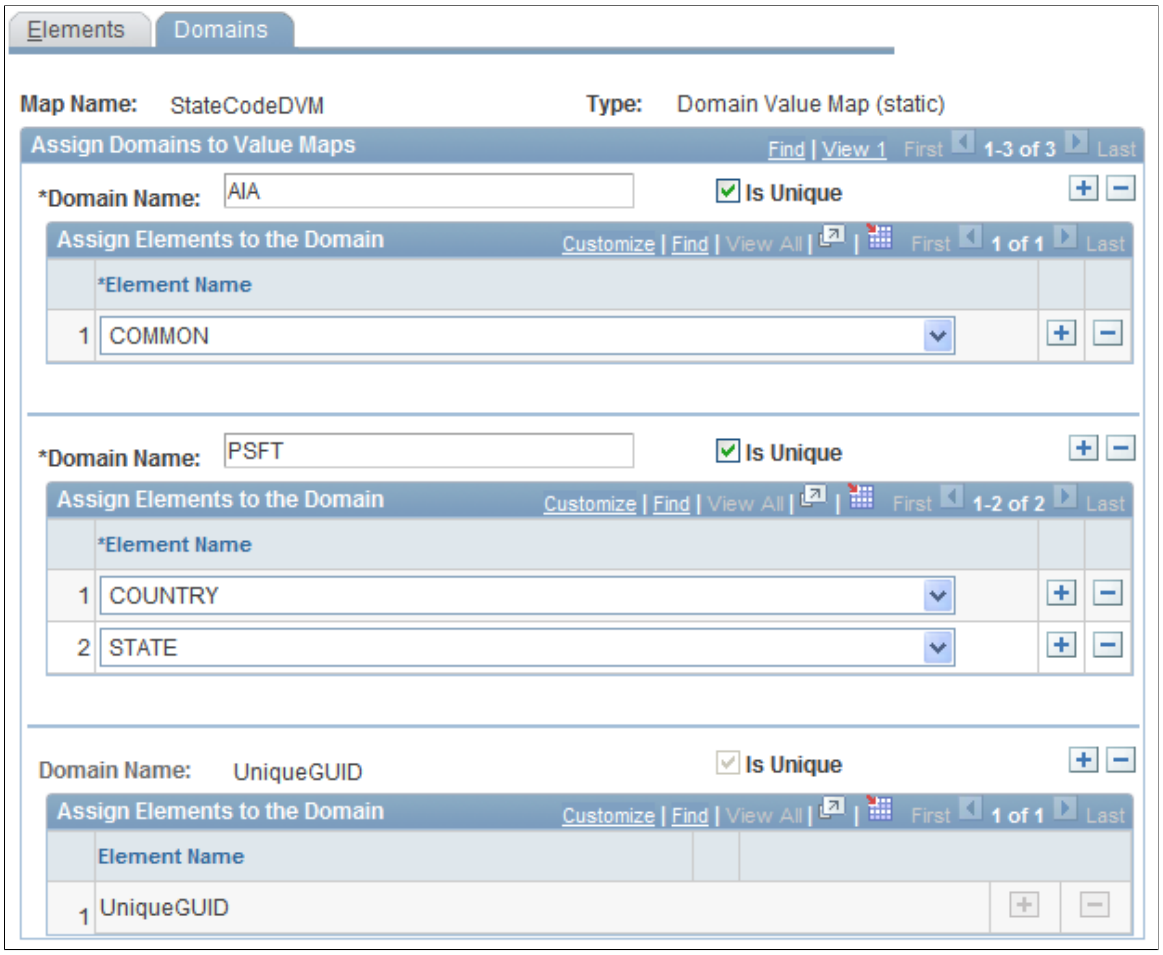
Use the Define Value Maps - Domains page (EOTF_DEFINE_MAPS2) to assign elements to domains.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps > Domains

Image: Define Value Maps - Domains page

This example illustrates the fields and controls on the Define Value Maps - Domains page. You can find definitions for the fields and controls later on this page.



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

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

Note: Domain name “UniqueGUID” will be automatically generated as it is required to maintain a unique value for each row of data in a map. Elements may be shared across multiple domains, as long as users understand the data value in the element is unique to the map, not the domain. In general, it is expected most DVM domains will contain a single element, and most PeopleSoft XREF domains will contain multiple elements, though this is not a rule.

Populating a Domain Value Map

This section discusses how to populate a domain value map.

Page Used to Populate a Domain Value Map

Page Name	*Definition Name*	*Usage*
Domain Value Map Page	EOTF_POPULATE_DVMS	Populate a domain value map.

Domain Value Map Page

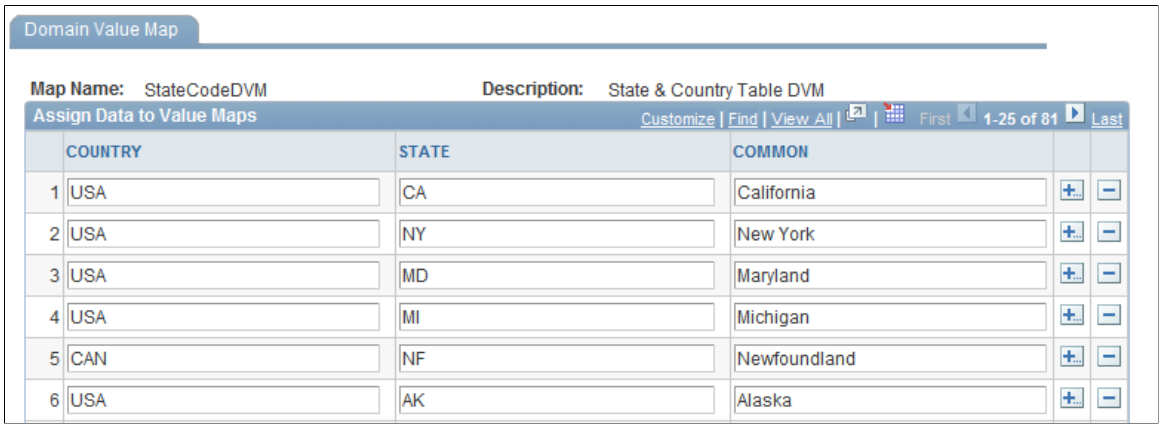
Use the Domain Value Map page (EOTF_POPULATE_DVMS) to populate a domain value map.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Populate Domain Value Maps

Image: Domain Value Map page

This example illustrates the fields and controls on the Domain Value Map page. You can find definitions for the fields and controls later on this 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.

Importing Value Maps

This section provides an overview of the import file types used with value maps and describes how to import value maps.

Page Used to Import Value Map

Page Name	*Definition Name*	*Usage*
Import Value Maps Page	EOTF_MAP_IMPORT	Import a DVM or XREF from a specified file.

Understanding Import File Types Used with Value Maps

There are 2 import file types supported in the Transformation Framework:

- XML Files
- CSV Files

Importing XML Files

There are 2 types of schemas supported in the Transformation Framework:

- V1 Fusion Middleware (FMW)

  This is the schema used with FMW using 11g format. The associated service operation is EOTF_DVM_IMPORT.v1. This is not the default version of the service operation, therefore a version transformation program is executed when a value map is imported in this format.

  See EOTF_DVM_IMPORT.v1, EOTF_XREF_IMPORT.v1.

- PeopleSoft Format

  This is the schema used for PeopleSoft. The associated service operation is EOTF_DVM_IMPORT .v2, which is the default service operation.

  See EOTF_DVM_IMPORT.v2, EOTF_XREF_IMPORT.v2.

When the specified file contains XML, the XML must conform to one of the schemas supported by the import web service. When the file contains XML, users may choose to import a map definition, map data, or both. When importing using the FMW (V1) schema, users may desire to have the import process break composite values into element values. To do so, users need to import the definition, then alter the definition to specify multiple elements for one or more of the domains using the Value Map Definition component, then go back and import data values only. In this scenario, the process will unconcatenate the values using the specified separator. An error will be thrown if a required element is not valued.

Importing CSV Files

When the file contains CSV data, the following rules apply:

- The first row in the file must contain the type of the value map (*DVM* or *XREF*) in the first column, and the name of the value map to be imported in the second column.
- Each column in the second row of the file must identify either domain names or element names to import into (no mixing allowed), or reference data. When importing into an existing map, domain and element names will be validated against the map definition. An error will be thrown if not found, or if all required elements for a domain are not included in the file. To define a reference data column that will be ignored by the import process, prefix the name with an asterisk, or leave the name blank.

- Subsequent rows in the file identify the data values to be imported, and must contain the same number of columns as the second row.
- When the data values in a column are composite (concatenated) values identifying the value of each element in the domain, and the second row in the file identifies domain names, the import process will unconcatenate the values using the specified separator for each domain defined with multiple elements. An error will be thrown if a required element is not valued.
- If the value map already contains data for other domains, and the user wishes to add this domain's values to the existing rows of data, the domain referenced in the first column of data will be used to locate an existing row of data. If a row of data is not found, a new row of data will be created. If multiple rows are found and the additional domain(s) in the file do not allow duplicates, an error will occur. Otherwise, the existing rows of data will be updated with the data values for the other domain(s) in the file.
- If UniqueGUID is blank or not specified in the file, a value will be generated for each new row of data.
- When importing a map definition, each column will be treated as a domain with a single element by the same name. Users may import definition only, then use the Define Value Maps component to alter the definition, then import data only to allow importing composite (concatenated) values or domains with multiple elements.

This is an example of the cross-reference value map csv file in MicroSoft Excel:

Image: Sample csv file to import a cross reference value map

This example illustrates a sample csv file to import a cross reference value map. You can find definitions for the fields and controls later on this page.

XrefTest2.csv

	A	B	C	D
1	XREF	XrefTest		
2	UniqueGUID	PSFT	*FusionValue	FusionGUID
3	asdf1	US001::VEND000001	V0101	1
4	asdf2	US002::VEND000002	V0202	2
5	asdf3	US003::VEND000003	V0303	3
6	asdf4	US004::VEND000004	V0404	4
7	asdf5	US005::VEND000005	V0405	5
8	asdf6	US006::VEND000006	V0406	6
9	asdf7	US007::VEND000007	V0407	7
10				

This is an example of the DVM value map csv file in MicroSoft Excel:

Image: Sample csv file to import DVM value map

This example illustrates a sample csv file to import a DVM value map. You can find definitions for the fields and controls later on this page.

DVMTest.csv

	A	B	C
1	DVM	StateCodes	
2	PeopleSoft	*Description	ExternalSystem
3	MA	Massachusetts	Mass
4	CA	California	Calif
5	NV	Nevada	Nev
6	OR	Oregon	Oreg
7			

Import Value Maps Page

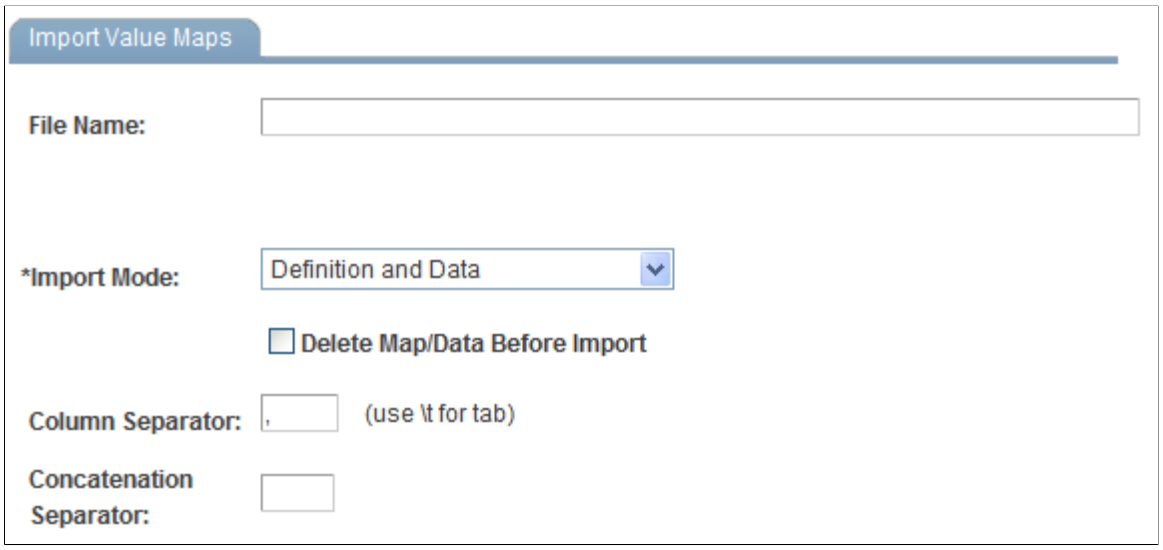
Use the Import Value Maps page (EOTF_MAP_IMPORT) to import a DVM or XREF from a specified file.

Navigation

Enterprise Components > Integration Definitions > Transformation Framework > Import Value Maps

Image: Import Value Maps page

This example illustrates the fields and controls on the Import Value Maps page. You can find definitions for the fields and controls later on this page.



The import value maps component provides users the ability to import a DVM or XREF from a specified file. The import file must be formatted as XML or comma separated values (CSV). If the import may result in data loss, a confirmation dialog will be displayed. Option defaults will be pulled from the system defaults set up by the user in the Value Map Options component.

File Name	The name of the file including the path. Files must be XML or CSV format and must contain a map name.
Import Mode	Allows a user to import a definition, alter the definition via the define value maps component, and subsequently import data only. This gives users the ability to import a definition and then rename elements, or perhaps specify multiple elements for a domain prior to importing data. Valid values are: • Definition Only • Definition and Data • Values Only
Delete Map/Data Before Import	Select to delete existing definition and data prior to import (default). If this check box is cleared, append/merge is used. Append/merge allows a user to add elements, domains or data to an existing map without deleting existing data. **Note:** This option is for advanced users only as they must understand the implications.
Column Separator	CSV files allow characters other than a comma to be utilized for delimiting column values. The value entered here is the delimiter value that will be used when reading the specified file. To specify a tab character, enter *\t* as the separator value.
Concatenation Separator	For V1 (FMW) schemas, if values are concatenated from multiple element domains into a single column during export or unconcatenated during an import of data only, this is the separator value to be used.
Import	Verifies the options and initiates the import process.

Exporting Value Maps

This section provides an overview of export schemas and discusses how to export a value map.

Understanding Export Schemas

Value maps can be exported from the Define Value Maps page using the Export button. The map definition and data is published in an XML format using the options defined for the value map. The XML will conform to the schemas specified in the active version of the EOTF_DVM_EXPORT service operation.

- FMW format (V1)

  To export using the FMW schema, you must activate and supply a valid routing for the service operation EOTF_DVM_EXPORT.v1.

Domain names are used as column names when exporting to a V1 schema. If multiple element domains are exported to the V1 schema, the element values are concatenated to form a compound value for each domain.

- PeopleSoft

  To export using the PeopleSoft schema, you must activate and supply a valid routing for the service operation EOTF_DVM_EXPORT.v2.

- Both

  Activate and provide a routing for both EOTF_DVM_EXPORT.v1 and EOTF_DVM_EXPORT.v2.

See *PeopleTools: PeopleSoft Integration Broker,* Managing Service Operations, Configuring Service Operation Definitions

Exporting a Value Map

To export a value map:

1. Select Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps.
2. Select the map you want to export.
3. Click the Export button.

Chapter 4

Creating Transform Programs and Updating Service Operations

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.

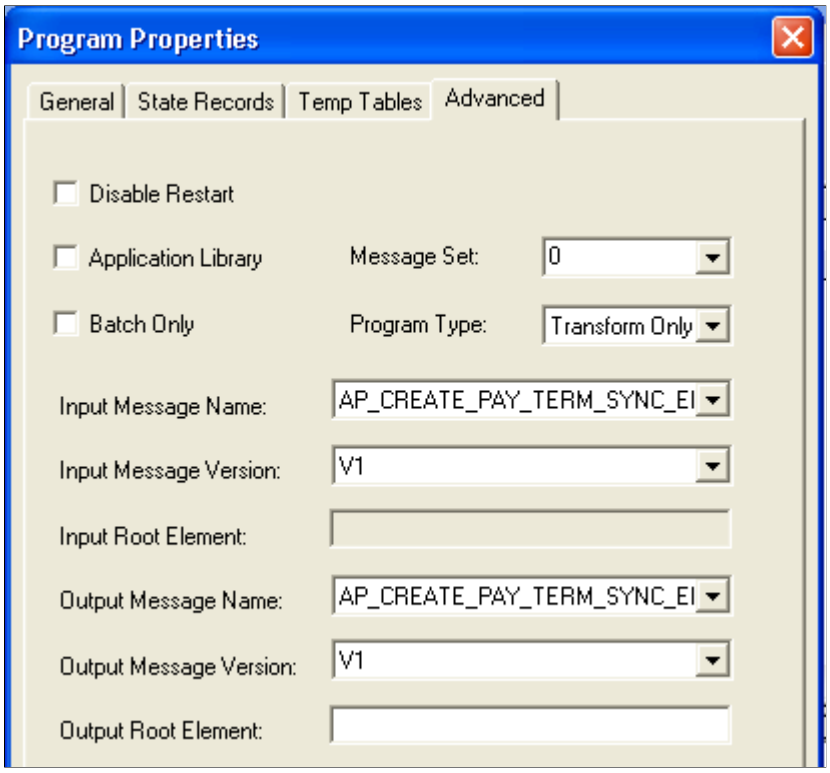
See *PeopleTools: PeopleSoft Integration Broker,* Applying Filtering, Transformation and Translation, Developing Transform Programs.

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.

Image: Application Engine program properties for a transform program

This example illustrates the fields and controls on the Application Engine program properties for a transform program. You can find definitions for the fields and controls later on this page.



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 *PeopleTools: 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 Understanding XSLT Extension Functions topic 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.

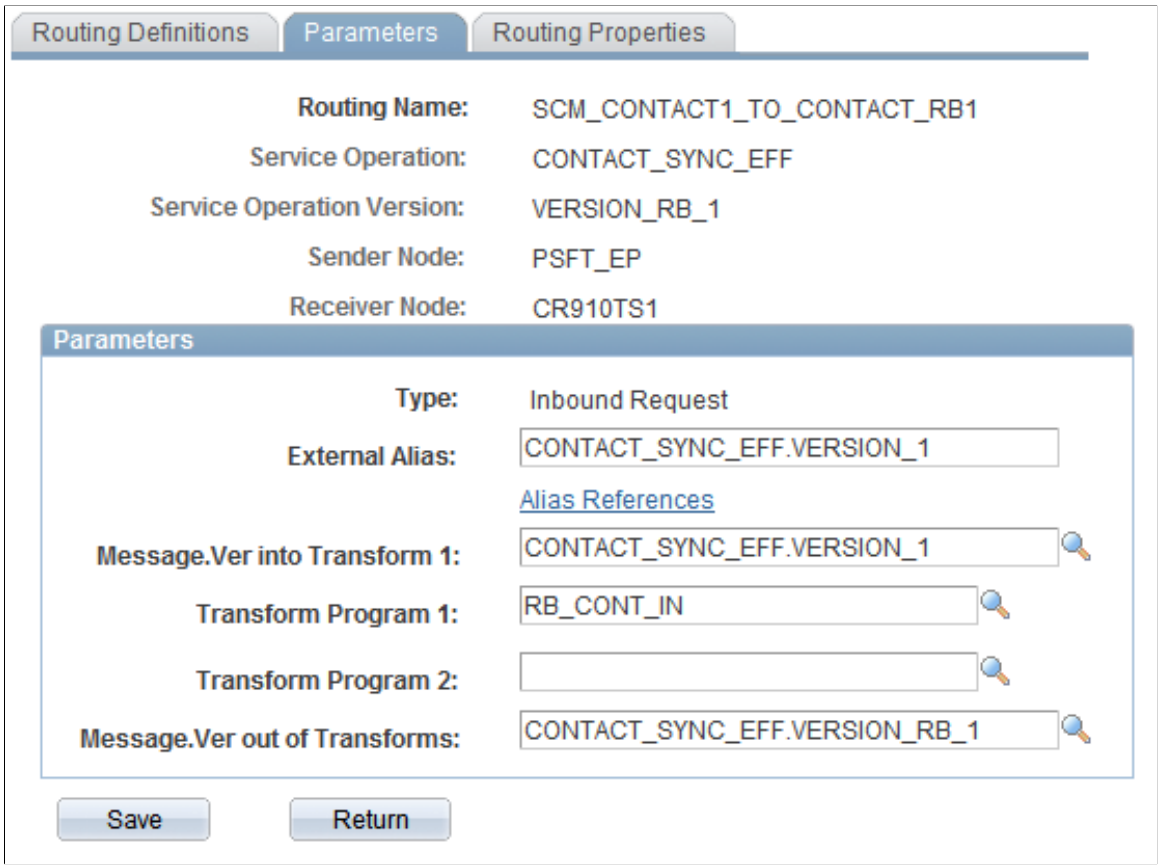
Adding Routing Parameters

To access the Routing Definition page to add routing parameters:

1. Select PeopleTools > Integration Broker > Integration Setup > Service Operations.
2. Select the Routings page.
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 page.

Image: Routing Parameters page for asynchronous one-way service operation

This example illustrates the fields and controls on the Routing Parameters page for asynchronous one-way service operation. You can find definitions for the fields and controls later on this page.



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 *PeopleTools: PeopleSoft Integration Broker,* Managing Routing Definitions, Creating Routing Definitions.

Chapter 5

Accessing Maps Using XSLT Extension

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 in alphabetical order.

xref:BulkPopulateDomainData

Syntax

xref:BulkPopulateDomainData(*mapName*, *domainList*, *dataRowsPath*, *dataValuesPath*, *mode*)

Description

The XREF bulk populate domain data function populates cross-reference data from XML. It sequences through the rows of data provided in the XML nodeset and uses the supplied domain list and data values path to access the values for each domain in each row. Each row of data is then inserted into the XREF framework for the specified map utilizing bulk insert to maximize performance. This function is expected to be used for processing large volumes of data. It provides an abstraction layer from the physical data persistence layer of the XREF framework. because inserts are done using bulk mode, duplicates are not looked for until the end of the insert sequence. If a duplicate is encountered, none of the rows of data will be inserted. When a domain contains multiple elements, each value for the domain must contain a compound value consisting of a value for each element in the domain separated by the concatenation string specified in the value map options. In this scenario, this function will unconcatenate the domain element values before inserting the data into the transformation framework.

Parameters

Parameter	*Description*
mapName	The name of a dynamic (cross reference) value map, as string.
domainList	A comma separated list of domain names identifying the order of the data values to be imported.

Parameter	*Description*
dataRowsPath	An XPath expression resulting in a series of XML nodes where each node represents one row of data.
dataValuesPath	An XPath expression (relative to each row of data) used to identify each node containing a data value within a row of data. The nodeset returned when this path is evaluated must contain the same number of nodes as there are domains in the specified domainList.
mode	Only *ADD* mode is supported.

Returns

Boolean indicating success or failure of the process.

xref:BulkPopulateElementData

Syntax

`xref:BulkPopulateElementData`(*`mapName`*, *`elementList`*, *`dataRowsPath`*, *`dataValuesPath`*, *`mode`*)

Description

The XREF bulk populate element data function populates cross-reference data from XML. It sequences through the rows of data provided in the XML nodeset and uses the supplied pairs of elements and XPath expressions to access the values for the elements in each row. Each row of data is then inserted into the XREF framework for the specified map utilizing bulk insert to maximize performance. This function is expected to be used for processing large volumes of data. It provides an abstraction layer from the physical data persistence layer of the XREF framework. because inserts are done using bulk mode, duplicates are not looked for until the end of the insert sequence. If a duplicate is encountered, none of the rows of data will be inserted.

Parameters

Parameter	*Description*
mapName	The name of a dynamic (cross reference) value map, as string.
elementList	A comma separated list of element names identifying the order of the data values to be imported.
dataRowsPath	An XPath expression resulting in a series of XML nodes where each node represents one row of data.

Parameter	*Description*
dataValuesPath	An XPath expression (relative to each row of data) used to identify each node containing a data value within a row of data. The nodeset returned when this path is evaluated must contain the same number of nodes as there are domains in the specified domainList.
mode	Only *ADD* mode is supported.

Returns

Boolean indicating success or failure of the process.

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

Parameter	*Description*
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.

Mode	*Reference Value*	*Value to Be Added*	*Result*
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:

Mode	*Description*	*Exception Reasons*
ADD	Adds the reference value and the value. For example: `xref:populateXRefRow("customers", "PS","PS101","Common","CM001","ADD⇒ ")` adds the reference value *PS101* in the *PS* element and the value *CM001* in the *Common* element of the *customers* cross-reference map.	Exceptions can occur for the following reasons: • The specified cross-reference map is not found. • The specified elements are not found. • The values provided are empty. • The value being added is not unique across that element for that map. • The element for that row already contains a value. • The reference value exists.

Mode	Description	Exception Reasons
LINK	Adds the cross-reference value corresponding to the existing reference value. For example: `xref:populateXRefRow("customers", "Common","CM001","SBL","SB-101", "LINK")` adds the value *SB-101* in the *SBL* element of the *customers* cross-reference map and links it to the value *CM001* in the *Common* element.	Exceptions can occur for the following reasons: • The specified cross-reference map is not found. • The specified elements are not found. • The values provided are empty. • The reference value is not found. • The value being linked exists in that element for that map.
UPDATE	Updates the cross-reference value corresponding to an existing reference element-value pair. For example: `xref:populateXRefRow("customers","PS", "PS100","PS","PS1001","UPDATE")` updates the value *PS100* in the *PS* element of the *customers* cross-reference map to value *PS1001*.	Exceptions can occur for the following reasons: • The specified cross-reference map is not found. • The specified elements are not found. • The values provided are empty. • The value being updated is not unique across that element for that map. • Multiple values are found for the element being updated. • The reference value is not found. • The element for that row does not have a value.

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

Parameter	Description
mapName	The name of the cross-reference map, as string.

Parameter	*Description*
referenceDomain	The name of the reference domain, as string.
rreferencesNVP	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.

Mode	*Reference Value*	*Value to Be Added*	*Result*
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 the modes with their descriptions and exception reasons:

Mode	Description	Exception Reasons

ADD	Adds the reference value and the value to be added. For example:
	
	`xref:populateXRefRowNVP("Items", "PeopleSoft", "<Setid>SHARE</Setid>⇒`
	
	`<ItemID>1005</ItemID>","Common", "<Common>" | generate-guid() | "</Common>","ADD")`
	
	adds the reference values *SHARE/1005* in the *PeopleSoft* domain and the value <*guid1*> in the *Common* domain.
		Exceptions can occur for the following reasons:
		
		• The specified cross-reference map is not found.
		• The specified domains are not found.
		• The specified elements are not found.
		• The values provided are empty.
		• The values being added are not unique across that domain for that map.
		• The element for that row already contains a value.
		• The reference value exists.

LINK	Adds the cross-reference value corresponding to the existing reference value. For example: `xref:populateXRefRowNVP("Items", "PeopleSoft","<Setid>SHARE</Setid> <ItemID>1005</ItemID>","Retail", "<Product>RP0005</Product>","LINK")⇒` adds value *RP005* to the *Retail* domain and links it to reference values *SHARE/1005* in the *PeopleSoft* domain.	Exceptions can occur due for following reasons: • The specified cross-reference map is not found. • The specified domains are not found. • The specified elements are not found. • The values provided are empty. • The reference value is not found. • The value being linked exists in that domain for that map.

Mode	*Description*	*Exception Reasons*
UPDATE	Updates the cross-reference value corresponding to an existing reference element-value pair. For example: `xref:populateXRefRowNVP("Items", "PeopleSoft","<Setid>SHARE</Setid> <ItemID>1000</ItemID>","PeopleSoft"⇒ , "<Setid>SHARE</Setid> <ItemID>10000</ItemID>","UPDATE")` updates the value *1000* in the *ItemID* element of the *PeopleSoft* domain to value *10000*.	Exceptions can occur for the following reasons: • The specified cross-reference map is not found. • The specified domains are not found. • The specified elements are not found. • The values provided are empty. • The values being updated are not unique across that domain for that map. • Multiple values are found for the domain being updated. • The reference value is not found. • The element for that row does not have a value.

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

Parameter	*Description*
mapName	The cross-reference map name, as string.

Parameter	*Description*
elementName	The name of the element from which you want to delete a value, as string.
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

Parameter	*Description*
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</Item⇒
ID>")
```

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

Parameter	*Description*
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

Parameter	***Description***
mapName	The name of the cross-reference map, as string.
referenceDomain	The name of the reference domain, as string.
referenceNVP	NVP list of reference elements and values, as string.
targetDomain	The name of the domain to be looked up for the values, 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 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

Parameter	*Description*
mapName	The domain value map name, as string.
referenceElementName	The source element name, as string.
referenceValue	The source value (an XPath expression bound to the source document of the XSLT transformation), as string.
elementName	The target element name, as string.
defaultValue	If the value is not found, then the default value is returned, 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 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

Parameter	***Description***
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</Unique⇒
GUID>",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.
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

Parameter	***Description***
SetId	The SetId value interested in lookup.
LookupType	The lookup type is used to indicate the type of lookup. You can specify either *1* for the record group name or *2* for the record name.
LookupValue	Value should be either a record name or record group ID as determined by LookupType.
dvmTranslate	Specify *True* if translation to Common ID using DVM Name supplied is desired. Specify *False* 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","COM⇒
MON",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

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.

DVM Utility Class	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.
SetId Utility Class	Functions for PeopleCode developers to access SetId data stored in a DVM during transformations.
Xref Utility Class	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.

ExtractData

Syntax

ExtractData *(mapName, domainList, tempRecName, instance)*

Description

You can use the DVM extract data function to generate and execute set-based SQL to extract DVM data into a specified table. Prior to data extraction, all data is removed from the temp table for the specified process instance. Element values are mapped to fields in the temp record using the order of the specified domain list followed by the element order specified in the value map definition. If no domains are specified, element values are mapped to fields in the temp record using only the element order specified in the value map definition. This function is expected to be used for processing large volumes of data. It provides an abstraction layer from the physical data persistence layer of the DVM framework.

The temp table has the following design constraints:

1. The first column will be PROCESS_INSTANCE.
2. The second column will be used for UniqueGUID (EOTF_COMMONELEMENT).
3. The remaining columns correspond to the elements in the specified domain list.
4. The temp table column names do not have to match the element names in the DVM.
5. The table should be keyed by ProcessInstance, UniqueGUID.
6. There may be a performance benefit from having an index on the elements of each specified domain.

Parameters

Parameter	*Description*
mapName	The name of static value map definition (DVM), as sting.
domainList	The name of the value map domain(s) to extract data for. Provide null or an empty array in this parameter to extract data for all domains in the map.
tempRecName	The name of the temp table to extract data into, as string.
instance	The number to use when qualifying process instance, as integer.

Returns

Boolean indicating success or failure of the process.

Example

```
Local EOTF_CORE:DVM:Functions &dvm = create EOTF_CORE:DVM:Functions();
Local string &mapName = "States";
Local array of string &domainList = CreateArray("PS");
Local string &tempRecName = Record.HR_STATES_TAO;
Local boolean &success =
&dvm.ExtractData(&mapName, &domainList, &tempRecName, &instance);
```

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

Parameter	***Description***
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.
defaultValue	Default value to be returned if needAnException is false and an error occurs, as string.
needAnException	*True* to return error messages, *false* 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, &ValueNo⇒
tFound, 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

Parameter	*Description*
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 equivalent values for, as string.
needAnException	*True* to return error messages, *false* 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

Parameter	***Description***
mapName	Name of a static value map definition (DVM), as string.
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	*True* to return error messages, *false* 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(&Tes⇒
tName, &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

Parameter	*Description*
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 *True* 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.
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	*True* to cause exception to occur in case set control values are not found. *False* 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", tru⇒
e, "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

Parameter	***Description***
LookupValue	Value to use in setID lookup, as string. Value should either be a set control value if dvmTranslate is false or a common element value if dvmTranslate is true.
dvmTranslate	Specify *True* 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 *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.
needAnException	*True* to cause exception to occur in case setID is not found. *False* 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", "COMMO⇒
N", "1" ,"VENDOR", false);
```

Xref Utility Class Methods

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

BulkPopulateDomainData

Syntax

BulkPopulateDomainData (*mapName*, *domainList*, *dataRows*, *dataValuesPath*, *mode*)

Description

The XREF bulk populate domain data function populates cross-reference data from XML. It sequences through the rows of data provided in the XML nodeset and uses the supplied domain list and data values path to access the values for each domain in each row. Each row of data is then inserted into the XREF framework for the specified map utilizing bulk insert to maximize performance. This function is expected to be used for processing large volumes of data. It provides an abstraction layer from the physical data persistence layer of the XREF framework. Because inserts are done using bulk mode, duplicates are not looked for until the end of the insert sequence. If a duplicate is encountered, none of the rows of data will be inserted. When a domain contains multiple elements, each value for the domain must contain a compound value consisting of a value for each element in the domain separated by the concatenation string specified in the value map options. In this scenario, this function will unconcatenate the domain element values before inserting the data into the transformation framework.

Parameters

Parameter	***Description***
mapName	The name of a dynamic (cross reference) value map, as string.
domainList	An array of domain names identifying the order of the data values to be imported.
dataRows	An array of XML nodes where each node in the array is one row of data.

Parameter	Description
dataValuesPath	An XPath expression (relative to each row of data) used to identify each node containing a data value within a row of data. The nodeset returned when this path is evaluated must contain the same number of nodes as there are domains in the specified domainList.
mode	Only *ADD* mode is supported.

Returns

Boolean indicating success or failure of the process.

Example

```
Local EOTF_CORE:Xref:Functions &xref = create EOTF_CORE:Xref:Functions();
Local string &mapName = "VendorIDs";
Local array of string &domainList = ...;
Local array of XmlNode &dataRows = ...;
Local string &dataValuesPath = "...";
Local string &mode = "ADD";
Local boolean &success =
&xref.BulkPopulateDomainData(&mapName, &domainList, &dataRows, &dataValuesPath, &mo⇒
de);
```

BulkPopulateElementData

Syntax

BulkPopulateElementData (*mapName*, *elementList* , *dataRows*, *dataValuesPath*, *mode*)

Description

The XREF bulk populate element data function populates cross-reference data from XML. It sequences through the rows of data provided in the XML nodeset and uses the supplied pairs of elements and XPath expressions to access the values for the elements in each row. Each row of data is then inserted into the XREF framework for the specified map utilizing bulk insert to maximize performance. This function is expected to be used for processing large volumes of data. It provides an abstraction layer from the physical data persistence layer of the XREF framework. because inserts are done using bulk mode, duplicates are not looked for until the end of the insert sequence. If a duplicate is encountered, none of the rows of data will be inserted.

Parameters

Parameter	Description
mapName	The name of a dynamic (cross reference) value map, as string.
elementList	An array of element names identifying the order of the data values to be imported.

Parameter	*Description*
dataRows	An array of XML nodes where each node in the array is one row of data.
dataValuesPath	An XPath expression (relative to each row of data) used to identify each node containing a data value within a row of data. The nodeset returned when this path is evaluated must contain the same number of nodes as there are elements in the specified elementList.
mode	Only *ADD* mode is supported.

Returns

Boolean indicating success or failure of the process.

Example

```
Local EOTF_CORE:Xref:Functions &xref = create EOTF_CORE:Xref:Functions();
Local string &mapName = "VendorIDs";
Local array of string &elementList = ...;
Local array of XmlNode &dataRows = ...;
Local string &dataValuesPath = "...";
Local string &mode = "ADD";
Local boolean &success =  &xref.BulkPopulateElementData(&mapName, &elementList, &da⇒
taRows, &dataValuesPath, &mode);
```

ExtractData

Syntax

ExtractData (*mapName*, *domainList*, *tempRecName*, *instance*)

Description

You can use the XREF extract data function to generate and execute set-based SQL to extract cross-reference data into a specified table. Prior to data extraction, all data is removed from the temp table for the specified process instance. Element values are mapped to fields in the temp record using the order of the specified domain list followed by the element order specified in the value map definition. If no domains are specified, element values are mapped to fields in the temp record using only the element order specified in the value map definition. This function provides an abstraction layer from the physical data persistence layer of the XREF framework, and is expected to be used for processing large volumes of data.

The temp table has the following design constraints:

1. The first column will be PROCESS_INSTANCE.
2. The second column will be used for UniqueGUID (EOTF_COMMONELEMENT).
3. The remaining columns correspond to the elements in the specified domain(s).

Note: If the third column is named EOTF_IMPORT_FLG, it will be valued with *N* during the export.

4. The temp table column names do not have to match the element names in the map.
5. The temp table should be uniquely keyed by ProcessInstance, UniqueGUID.
6. There may be a performance benefit from having an index on the elements of each specified domain.

Parameters

Parameter	*Description*
mapName	The name of the dynamic cross reference map, as string.
domainList	The name of the value map domain(s) to extract data for. Provide null or an empty array in this parameter to extract data for all domains in the map.
tempRecName	The name of the temp table to extract data into, as string.
instance	The number to use when qualifying process instance, as integer.

Returns

Boolean indicating success or failure of the process.

Example

```
Local EOTF_CORE:Xref:Functions &xref = create EOTF_CORE:Xref:Functions();
Local string &mapName = "VendorIDs";
Local array of string &domainList = CreateArray("PS");
Local string &tempRecName = Record.HR_VENDOR_TAO;
Local boolean &success =
&xref.ExtractData(&mapName, &domainList, &tempRecName, &instance);
```

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

Parameter	*Description*
mapName	Name of a dynamic (cross-reference) value map definition, as string.

Parameter	Description
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	*True* to return error messages, *false* 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

Parameter	Description
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.

Parameter	Description
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	*True* to return error messages, *false* 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, Fals⇒
e);
```

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

Parameter	Description
mapName	Name of a dynamic (cross-reference) value map definition, as string.

Parameter	Description
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	*True* to return error messages, *false* 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(&T⇒
estName, &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

Parameter	Description
mapName	Name of a dynamic (cross-reference) value map definition, as string.

Parameter	*Description*
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

Parameter	*Description*
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);
```

PopulateData

Syntax

PopulateData (*mapName*, *domainName*, *tempRecName*, *recName*, *instance*)

Description

You can use the XREF populate data function to generate and execute set-based SQL to populate cross-reference data from a specified source record (table or view). This function provides an abstraction layer from the physical data persistence layer of the XREF framework, and is expected to be used for processing large volumes of data.

The following steps are generated and executed in this function:

1. All data is removed from the temp table for the specified process instance.
2. Existing XREF data for the specified domain is extracted to the temp table with the import flag set to *N*.

   Note: Element values are mapped to fields in the record based on map element order (not on name).

3. Distinct rows of stage data are inserted into the temp table if they do not already exist, with the update flag set to *Y*.

   Note: Fields are mapped from temp to stage based on matching field names.

4. GUIDs are generated on the temp table where they are blank.
5. XREF data is inserted from the temp table where the import flag set to *Y*.
6. The stage data is updated with GUIDs from the temp table.

The temp table has the following design constraints:

1. The first column will be PROCESS_INSTANCE.
2. The second column will be used for UniqueGUID (EOTF_COMMONELEMENT).
3. The third column must be EOTF_IMPORT_FLG (for internal use by this function).
4. The remaining columns correspond to the elements in the specified domain.
5. The temp table column names do not have to match the element names in the maps.
6. The temp table column names must match the corresponding fields on the stage table.

7. The temp table should be keyed by ProcessInstance, UniqueGUID allowing for duplicates (blanks).

8. There may be a performance benefit from having an index on the elements of each specified domain.

Note: This function does not validate any of the data, therefore, users should be sure the data being populated does not violate any of the constraints defined in the value map definition. If the data rules are violated, future usability of the map may be impacted. This function is not responsible for protecting against the potential for truncation of existing data values caused by defining temp table fields shorter than the existing data values. If this occurs, no error will then be thrown, but existing values will not be matched properly to stage values, potentially resulting in unintended data redundancy.

Parameters

Parameter	*Description*
mapName	The name of a dynamic (cross reference) value map, as string.
domainName	The name of the value map domain to populate data for, as string.
tempRecName	The name of the temp table to extract existing XREF data to, as string.
recName	The name of the stage record to populate data from, as string.
instance	The number to use when qualifying process instance, as integer.

Returns

Boolean indicating success or failure of the process.

Example

```
Local EOTF_CORE:Xref:Functions &xref = create EOTF_CORE:Xref:Functions();
Local string &mapName = "VendorIDs";
Local string &domainName = "PS";
Local string &tempRecName = Record.HR_VENDOR_TAO;
Local string &recName = Record.HR_VENDOR_STG;
Local boolean &success =
&xref.PopulateData(&mapName, &domainName, &tempRecName, &recName, &instance);
```

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

Parameter	*Description*
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, "", &AD⇒
D);
```


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

Parameter	*Description*
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, &ebs⇒
NVP1, &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, &guid⇒
NVP3, &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, &guid⇒
NVP4, &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

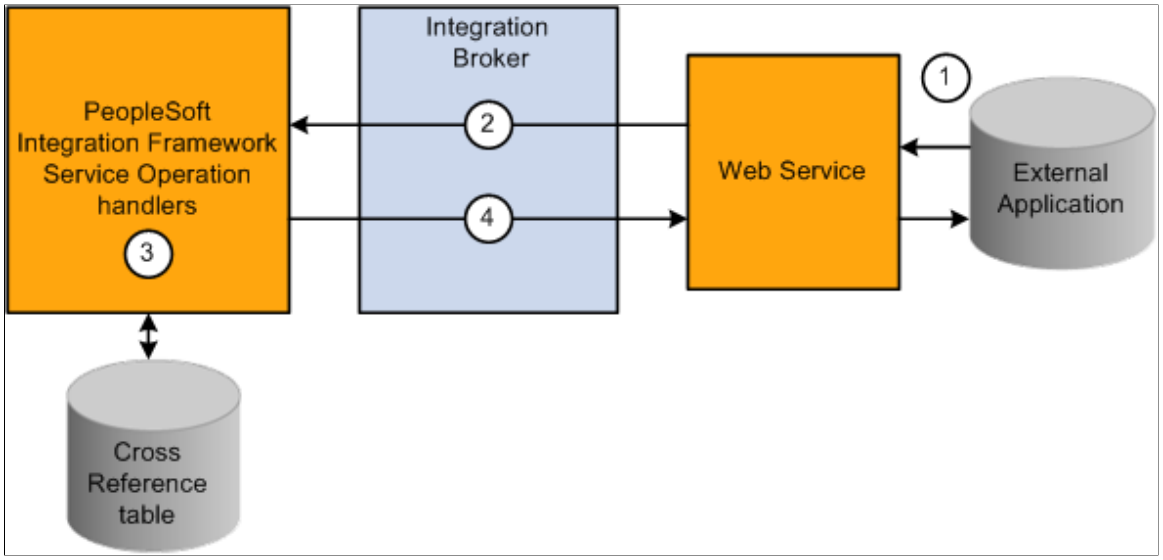
Accessing Maps Using Web Services

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.

Image: Using Application Integration Framework web service with an external application

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.



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.
4. Integration Broker sends the response to the external application.

EOTF_DVM Service

This section describes the Service Operations available as web services for DVM:

- EOTF_DVM_LOOKUP
- EOTF_DVM_IMPORT.v2
- EOTF_DVM_IMPORT.v1

EOTF_DVM_LOOKUP

This service operation is used to lookup values from a DVM.

Alias: dvmLookupValue

Type: Synchronous

Request Message: EOTF_DVM_LOOKUP_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <dvm:dvmLookupValue>
         <dvm:mapname>?</dvm:mapname>
         <!--2 or more repetitions:-->
         <dvm:Values>
            <dvm:domain>?</dvm:domain>
            <!--1 or more repetitions:-->
            <dvm:element name="?"/>
         </dvm:Values>
         <dvm:Values>
            <dvm:domain>?</dvm:domain>
            <!--1 or more repetitions:-->
            <dvm:element name="?"/>
         </dvm:Values>
         <dvm:targetDomain>?</dvm:targetDomain>
      </dvm:dvmLookupValue>
   </soapenv:Body>
```

Element Name	***Description***
mapname	The name of the domain value map to lookup values in.
Values	*This element holds the elements and values to be found in the DVM.*
domain	The name of the target domain provided on the lookup request.
element	Represents a DVM element and its value.
targetDomain	The name of the domain the lookup request should return values for.

Response Message: EOTF_DVM_RESPONSE.V1

Element Name	*Description*
status	The status of the operation. Returns *Success* or *Exception*.
reason	If there is an exception, the reason will be returned.
requestValues	*This element hold the reference values used for a lookup operation. For partially qualified lookups these values may differ from the values provided in the request.*
domain	The name of the reference domain provided on the lookup request.
element	The DVM element and its value.
responseValues	*The values returned for the lookup operation.*
domain	The name of the target domain provided on the lookup request.
element	The DVM element in the target domain and its value.

EOTF_DVM_IMPORT.v2

Use this service operation to import a Domain Value Map definition and values into the Enterprise Transformation Framework, based on the PeopleSoft schema.

Type: Asynchronous-One Way.

Alias: import_dvm

Default: Yes

Request Message: EOTF_DVM.V2

This is an example of the soap body for an import request using PeopleSoft schema:

```
<soapenv:Body>
      <dvm:dvm>
         <dvm:name>?</dvm:name>
         <!--Optional:-->
         <dvm:description>?</dvm:description>
         <!--Optional:-->
         <dvm:comments>?</dvm:comments>
         <dvm:elements>
            <!--2 or more repetitions:-->
            <dvm:element name="?" order="?" dataType="string" dataLength="?" isComm⇒
on="false"/>
            <dvm:element name="?" order="?" dataType="string" dataLength="?" isComm⇒
on="false"/>
         </dvm:elements>
         <dvm:domains>
            <!--1 or more repetitions:-->
            <dvm:domain name="?">
               <!--1 or more repetitions:-->
               <dvm:element name="?" qualifier="false"/>
```



```
            </dvm:domain>
         </dvm:domains>
         <!--Optional:-->
         <dvm:rows>
            <!--1 or more repetitions:-->
            <dvm:row>
               <!--2 or more repetitions:-->
               <dvm:cell>?</dvm:cell>
               <dvm:cell>?</dvm:cell>
            </dvm:row>
         </dvm:rows>
      </dvm:dvm>
   </soapenv:Body>
```

Element Name	***Description***
description	The DVM description. Optional
comments	Comments about the DVM. Optional
elements	*This element holds the DVM's element list.*
element	Represents a DVM element.
domains	*This elements holds the DVM's domain list.*
domain	This element represents a domain in a DVM.
element	This represents an element in the DVM domain.
rows	*This represents all the rows of data in the DVM.*
row	Each DVM row of values.
cell	This is the value for this row and the for each element in the same order as defined in elements.

EOTF_DVM_IMPORT.v1

Use this service operation to import a Domain Value Map definition and values into the Enterprise Transformation Framework, based on the FMW schema.

Type: Asynchronous-One Way.

Alias: import_dvm

Default: No

Request Message: EOTF_DVM.V1

This is an example of the soap body for an import request using FMW schema:

```
<soapenv:Body>
      <dvm:dvm name="?">
         <!--Optional:-->
         <dvm:description>?</dvm:description>
         <dvm:columns>
            <!--2 or more repetitions:-->
            <dvm:column name="?" qualifier="false" order="?"/>
            <dvm:column name="?" qualifier="false" order="?"/>
         </dvm:columns>
         <!--Optional:-->
         <dvm:rows>
            <!--1 or more repetitions:-->
            <dvm:row>
               <!--2 or more repetitions:-->
               <dvm:cell>?</dvm:cell>
               <dvm:cell>?</dvm:cell>
            </dvm:row>
         </dvm:rows>
      </dvm:dvm>
   </soapenv:Body>
```

Element Name	***Description***
description	The DVM description. Optional
columns	*This element holds the DVM's column list.*
column	Represents a DVM column.
rows	*This represents all the rows of data in the DVM.*
row	Each DVM row of values.
cell	This is the value for this row and the for each column in the same order as defined in columns.

EOTF_XREF Service

This section lists the service operations available for cross references:

- EOTF_XREF_ADD
- EOTF_XREF_LINK
- EOTF_XREF_UPDATE
- EOTF_XREF_DELETE
- EOTF_XREF_LOOKUP
- EOTF_XREF_IMPORT.v2

- EOTF_XREF_IMPORT.v1

EOTF_XREF_ADD

Use this service operation to add values to a cross-reference map.

Alias: add_xref_value

Type: Synchronous

Request Message: EOTF_XREF_ADD_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xrefAddValue>
         <xref:mapname>?</xref:mapname>
         <!--2 or more repetitions:-->
         <xref:Values>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:Values>
         <xref:Values>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:Values>
      </xref:xrefAddValue>
   </soapenv:Body>
```

Element Name	***Description***
mapname	The name of the cross-reference map to add values to.
Values	*This element holds the cross-reference elements and values to be populated.*
domain	The name of the domain the elements are a part of.
element	This represents an element and its value.

Response Message: EOTF_XREF_RESPONSE.V1

Element Name	***Description***
status	The status of the operation. Returns *Success* or *Exception*.
reason	If there is an exception, the reason will be returned.
responseValues	*The values returned for the lookup operation.*
domain	The name of the target domain provided on the lookup request.
element	Represents an element in the target domain and its value.

EOTF_XREF_LINK

Use this service operation to link values to a cross-reference map.

Alias: link_xref_value

Type: Synchronous

Request Message: EOTF_XREF_LINK_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xrefLinkValue>
         <xref:mapname>?</xref:mapname>
         <xref:ReferenceValues>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:ReferenceValues>
         <!--1 or more repetitions:-->
         <xref:TargetValues>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:TargetValues>
      </xref:xrefLinkValue>
   </soapenv:Body>
```

Element Name	***Description***
mapname	The name of the cross-reference map to add values to.
ReferenceValues	*This element holds the existing elements and values in a cross-reference to link target values with.*
domain	The name of the domain the reference elements are a part of.
element	This represents a cross-reference element and its value.
targetValues	This element holds the cross-reference elements and values to be added.
domain	The name of the domain the target elements are a part of.
element	This represents a cross-reference element and its value.

Response Message: EOTF_XREF_RESPONSE.V1

Element Name	***Description***
status	The status of the operation. Returns *Success* or *Exception*.
reason	If there is an exception, the reason will be returned.

Element Name	Description
responseValues	*The values returned for the lookup operation.*
domain	The name of the target domain provided on the lookup request.
element	Represents an element in the target domain and its value.

EOTF_XREF_UPDATE

Use this service operation to update values in a cross-reference map.

Alias: update_xref_value

Type: Synchronous

Request Message: EOTF_XREF_UPDATE_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xrefUpdateValue>
         <xref:mapname>?</xref:mapname>
         <xref:ReferenceValues>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:ReferenceValues>
         <!--1 or more repetitions:-->
         <xref:TargetValues>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:TargetValues>
      </xref:xrefUpdateValue>
   </soapenv:Body>
```

Element Name	Description
mapname	The name of the cross-reference map to update values to.
ReferenceValues	*This element holds the existing cross-reference elements and values to locate the row to be updated.*
domain	The name of the domain the reference elements are a part of.
element	This represents a cross-reference element and its value.
targetValues	This element holds the cross-reference elements to be updated and the new values.
domain	The name of the domain the target elements are a part of.
element	This represents a cross-reference element and its value.

Response Message: EOTF_XREF_RESPONSE.V1

Element Name	*Description*
status	The status of the operation. Returns *Success* or *Exception*.
reason	If there is an exception, the reason will be returned.
responseValues	*The values returned for the lookup operation.*
domain	The name of the target domain provided on the lookup request.
element	Represents an element in the target domain and its value.

EOTF_XREF_DELETE

Use this service operation to delete values from a cross-reference map.

Alias: delete_xref_value

Type: Synchronous

Request Message: EOTF_XREF_DELETE_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xrefDeleteValue>
         <xref:mapname>?</xref:mapname>
         <!--2 or more repetitions:-->
         <xref:Values>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:Values>
         <xref:Values>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?"/>
         </xref:Values>
      </xref:xrefDeleteValue>
   </soapenv:Body>
```

Element Name	*Description*
mapname	The name of the cross-reference map to delete values from.
Values	*This element holds the cross-reference elements and values to be deleted.*
domain	The name of the domain the elements are a part of.
element	This represents an element and its value.

Response Message: EOTF_XREF_RESPONSE.V1

Element Name	*Description*
status	The status of the operation. Returns *Success* or *Exception*.
reason	If there is an exception, the reason will be returned.
responseValues	*No response values are returned.*

EOTF_XREF_LOOKUP

Use this service operation to lookup values from a cross-reference map.

Alias: lookup_xref_value

Type: Synchronous

EOTF_XREF_LOOKUP_REQ.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xrefLookupValue needFault="true">
         <xref:mapname>?</xref:mapname>
         <xref:Values>
            <xref:domain>?</xref:domain>
            <!--1 or more repetitions:-->
            <xref:element name="?">?</xref:element>
         </xref:Values>
         <xref:targetDomain>?</xref:targetDomain>
      </xref:xrefLookupValue>
   </soapenv:Body>
```

Element Name	*Description*
mapname	The name of the cross-reference map to lookup values in.
Values	*This element holds the cross-reference elements and values to lookup.*
domain	The name of the domain the reference elements are a part of.
element	This represents a cross-reference element and its value.
targetDomain	The name of the domain the lookup request should return elements and values for.

Response Message: EOTF_XREF_RESPONSE.V1

Element Name	*Description*
status	The status of the operation. Returns *Success* or *Exception*.

Element Name	*Description*
reason	If there is an exception, the reason will be returned.
responseValues	*The values returned for the lookup operation.*
domain	The name of the target domain provided on the lookup request.
element	Represents an element in the target domain and its value.

EOTF_XREF_IMPORT.v2

Use this service operation to import a cross-reference map definition and values in using PeopleSoft schema.

Alias: xref

Type: Asynchronous - One Way

Default: Yes

Request Message: EOTF_XREF.V2

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xref>
         <xref:name>?</xref:name>
         <!--Optional:-->
         <xref:description>?</xref:description>
         <!--Optional:-->
         <xref:comments>?</xref:comments>
         <xref:elements>
            <!--2 or more repetitions:-->
            <xref:element name="?" order="?" dataType="string" dataLength="?" isCom⇒
mon="false"/>
            <xref:element name="?" order="?" dataType="string" dataLength="?" isCom⇒
mon="false"/>
         </xref:elements>
         <xref:domains>
            <!--1 or more repetitions:-->
            <xref:domain name="?">
               <!--1 or more repetitions:-->
               <xref:element name="?"/>
            </xref:domain>
         </xref:domains>
         <!--Optional:-->
         <xref:rows>
            <!--1 or more repetitions:-->
            <xref:row>
               <!--2 or more repetitions:-->
               <xref:cell>?</xref:cell>
               <xref:cell>?</xref:cell>
            </xref:row>
         </xref:rows>
      </xref:xref>
   </soapenv:Body>
```


Element Name	*Description*
description	The cross-reference description. Optional
comments	Comments about the cross-reference. Optional
elements	*This element holds the cross-reference element list.*
element	Represents a cross-reference element.
domains	*This elements holds the cross-reference domain list.*
domain	This element represents a domain in a cross-reference map.
element	This represents an element in the cross-reference domain.
rows	*This represents all the rows of data in the cross-reference map.*
row	Each cross-reference row of values.
cell	This is the value for this row and the for each element in the same order as defined in elements.

EOTF_XREF_IMPORT.v1

Use this service opertaion to import a cross-reference map definition and values in using PeopleSoft schema.

Alias: xref

Type: Asynchronous - One Way

Default: No

Transform: To v2

Request Message: EOTF_XREF.V1

This is an example of the soap body for the request:

```
<soapenv:Body>
      <xref:xref>
         <xref:table name="?">
            <!--Optional:-->
            <xref:description>?</xref:description>
            <!--Optional:-->
            <xref:columns>
               <!--1 or more repetitions:-->
               <xref:column name="?"/>
            </xref:columns>
            <!--Optional:-->
```

```
            <xref:rows>
               <!--1 or more repetitions:-->
               <xref:row>
                  <!--1 or more repetitions:-->
                  <xref:cell colName="?"/>
               </xref:row>
            </xref:rows>
         </xref:table>
      </xref:xref>
   </soapenv:Body>
```

Element Name	***Description***
table	*This element hold the table elements.*
description	The DVM description. Optional
columns	*This element holds the DVM's column list.*
column	Represents a DVM column.
rows	*This represents all the rows of data in the DVM.*
row	Each DVM row of values.
cell	This is the value for this row and the for each column in the same order as defined in columns.

Cross-Reference Lookup Web Service Example

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.

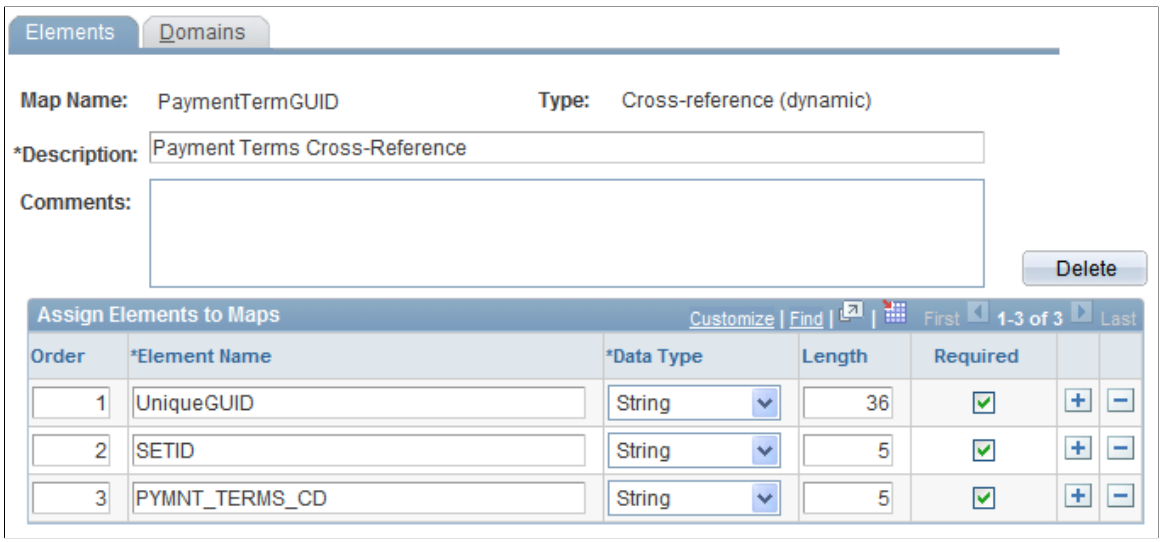
See *PeopleTools: PeopleSoft Integration Broker,* Providing Services.

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:

Image: Domain value map used in example

This example illustrates the fields and controls on the Domain value map used in example. You can find definitions for the fields and controls later on this page.

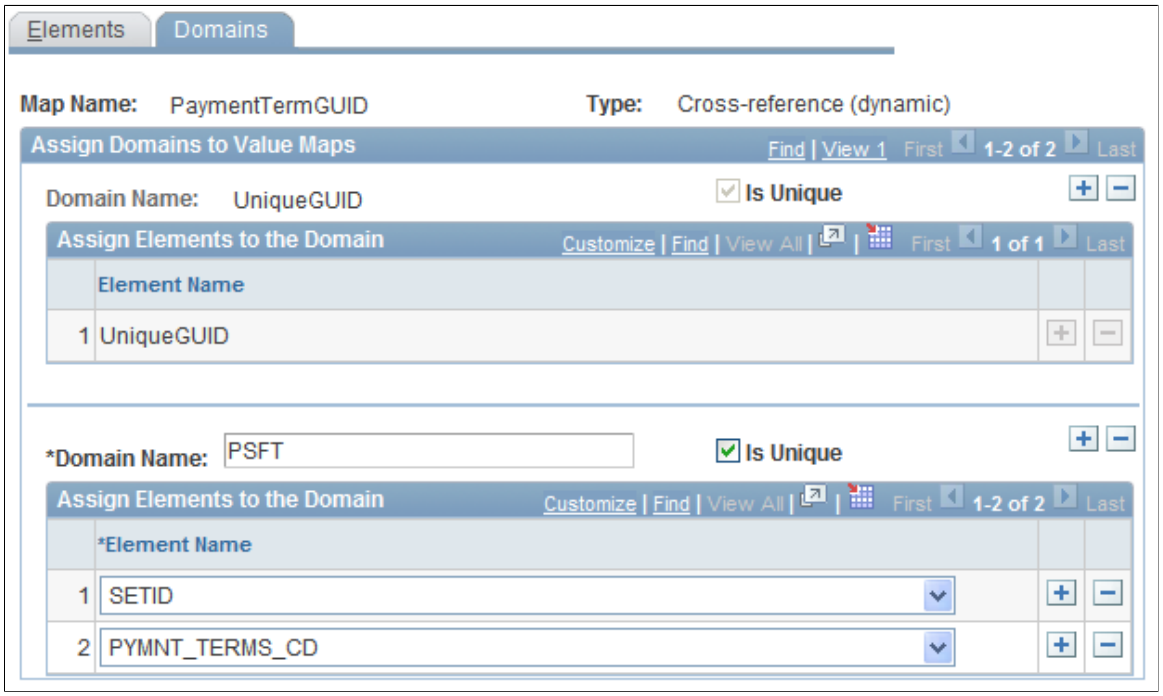


Three elements are defined: UniqueGUID, SETID, and PYMNT_TERMS_CD.

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

Image: Value Map - Domains page used in this example

This example illustrates the fields and controls on the Value Map - Domains page used in this example. You can find definitions for the fields and controls later on this page.



Two domains are defined for this map:

1. *UinqueGUID* is the domain used by the third party; it contains the UniqueGUID element.

2. *PSFT* represents the PeopleSoft application which contains the elements SETID and PYMNT_TERMS_CD.

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:x⇒
ref="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</x⇒
ref: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:s⇒
oapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/200⇒
1/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <xrefResponse xmlns="http://xmlns.oracle.com/Enterprise/tools/schema/xrefResp⇒
onse.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>
```

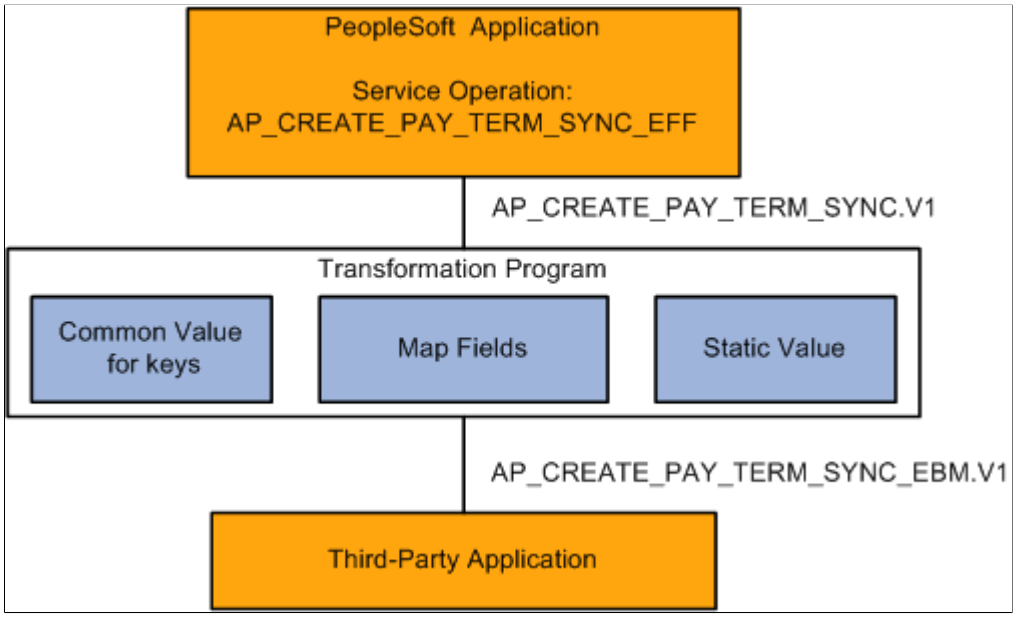
Appendix A

Application Integration Framework Example

Example Overview

Image: Payment terms integration flow

This diagram represents the scenario for this application.



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

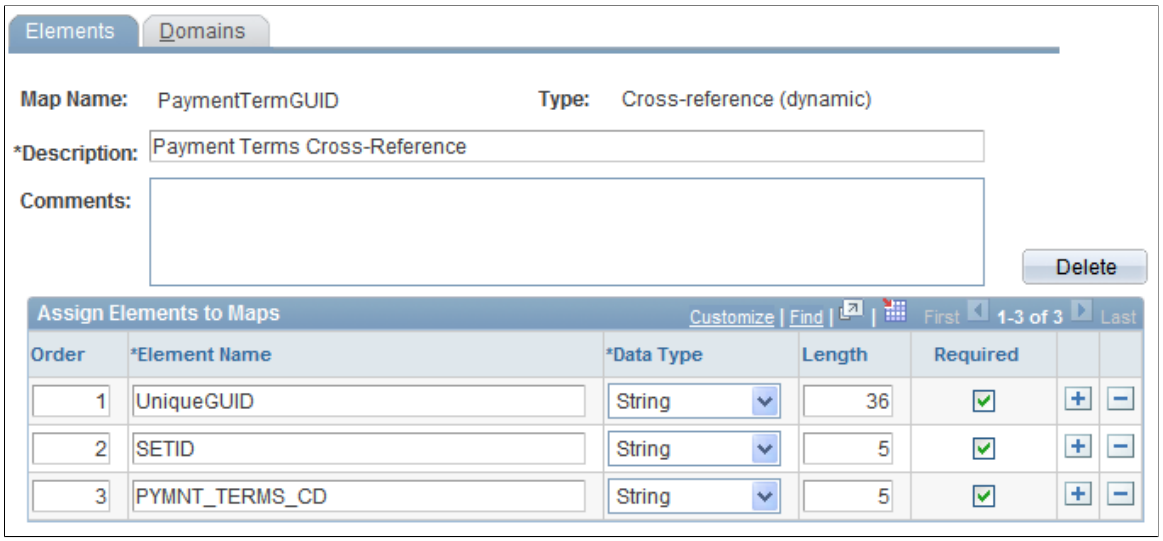
To create a new dynamic value map:

1. Select Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps.
2. Select the Add a New Value page.
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.

Image: Domain value map used in example

This example illustrates the fields and controls on the Domain value map used in example. You can find definitions for the fields and controls later on this page.



Three elements are defined: UniqueGUID, SETID, and PYMNT_TERMS_CD.

To add the domains:

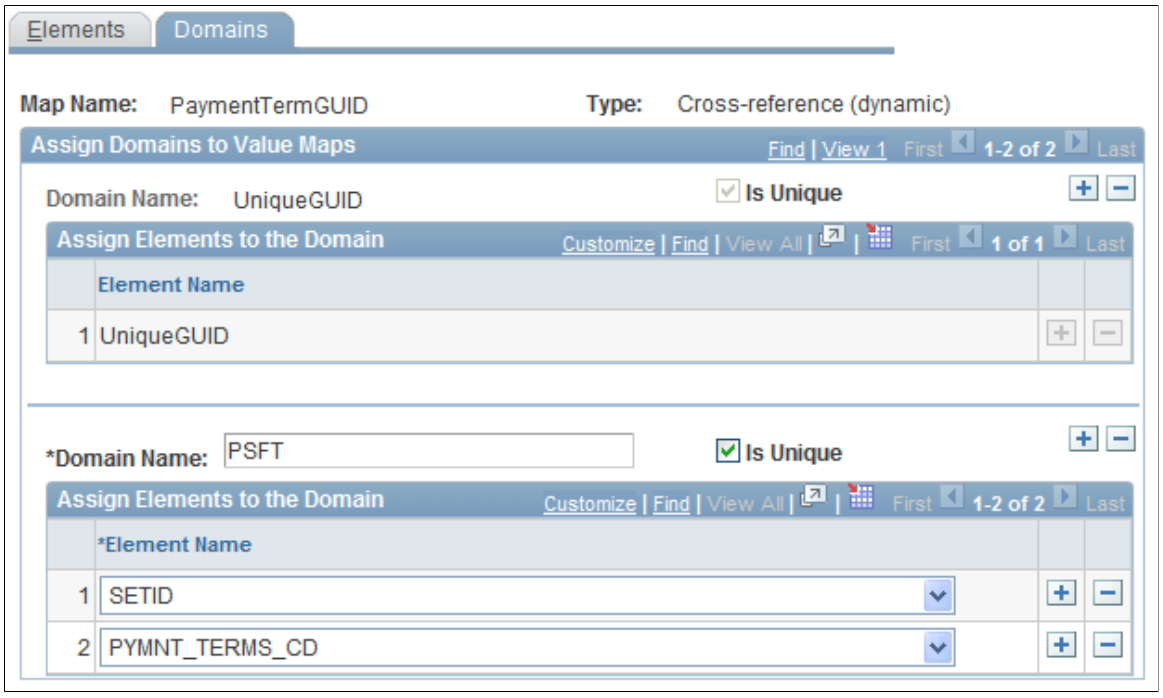
1. Select the Domains page.

   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.

Image: Value Map-Domains page used in this example

This example illustrates the fields and controls on the Value Map-Domains page used in this example. You can find definitions for the fields and controls later on this page.



Two domains are defined for this map:

- *UinqueGUID* is the domain used by the third party; it contains the UniqueGUID element.
- *PSFT* represents the PeopleSoft application and contains the elements SETID and PYMNT_TERMS_CD.

Defining and Populating a Static Value Map

To create a new domain value map:

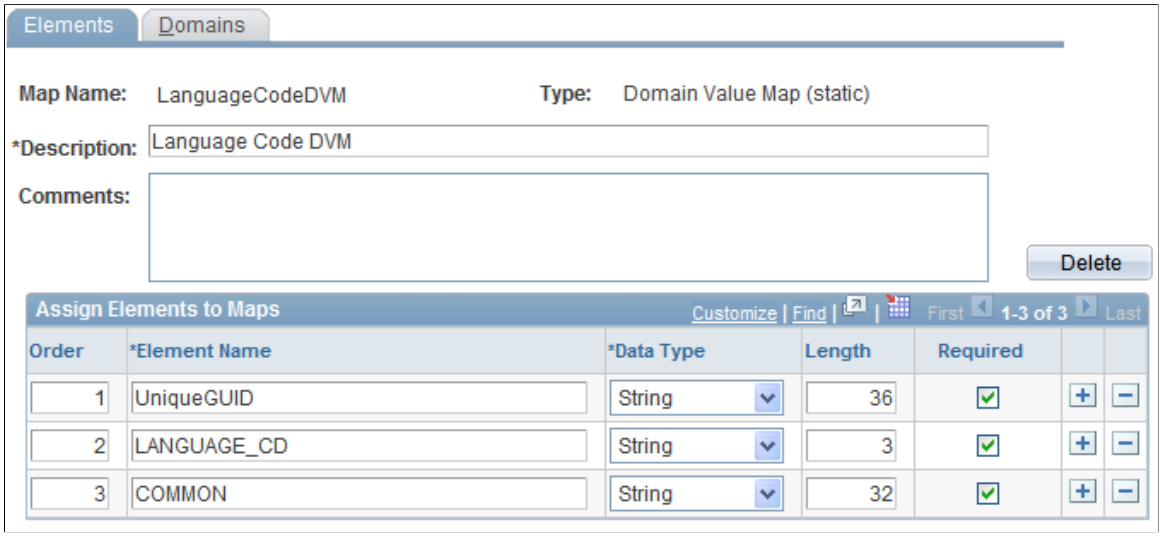
1. Select Enterprise Components > Integration Definitions > Transformation Framework > Define Value Maps.
2. Select the Add a New Value page.
3. Enter *LanguageCodeDVM* for the map name.
4. Select *Domain Value Map (static)* for the map type.
5. Click Add.

   The map will contain the element UniqueGUID.

6. Add two additional elements, LANGUAGE_CD and COMMON.

Image: Domain value map for language code

This example illustrates the fields and controls on the Domain value map for language code. You can find definitions for the fields and controls later on this page.



To add the domains:

1. Select the Domains page.

   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.

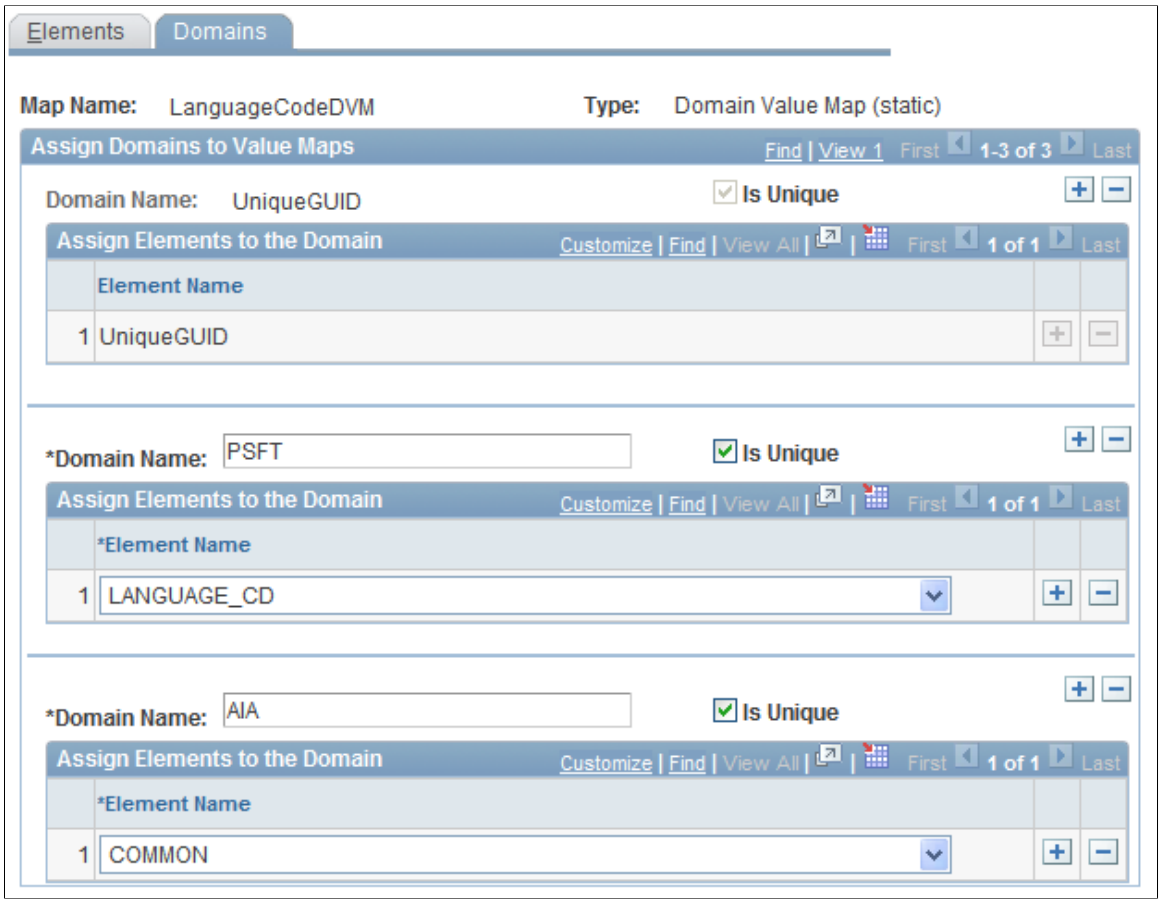
6. Enter *AIA* for the domain name.

7. Select *COMMON* for the first element.

8. Save the page.

Image: Domain value map domains for language code

This example illustrates the fields and controls on the Domain value map domains for language code. You can find definitions for the fields and controls later on this page.



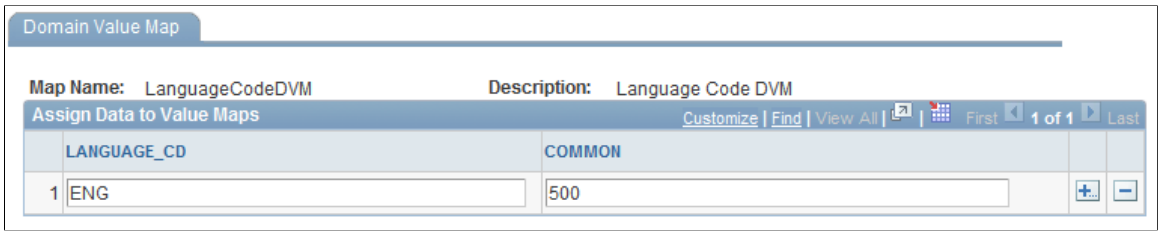
Three domains are defined: AIA, PSFT and UniqueGUID.

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.

Image: Domain Value Map page

This example illustrates the fields and controls on the Domain Value Map page. You can find definitions for the fields and controls later on this 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_T⇒
```

```
RMS_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;",util⇒
ity: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-11d⇒
d-b7dd-aaf7c4308a71</corecom:ID>
          <corecom:ContextID schemeAgencyID="PSFT" schemeID="SETID">SHARE</corecom:⇒
ContextID>
          <corecom:ContextID schemeAgencyID="PSFT" schemeID="Payment Terms Code">DI⇒
T91</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/LA⇒
NGUAGE_CD,false())'/>
<xsl:variable name="BaseLang" select='dvm:lookup-dvm("LanguageCodeDVM","LANGUAGE_CD⇒
",//MsgData/Transaction/PSCAMA/BASE_LANGUAGE_CD,"COMMON",//MsgData/Transaction/PSCA⇒
MA/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="h⇒
ttp://xmlns.oracle.com/EnterpriseObjects/Core/Custom/EBO/PaymentTerm/V1" xmlns:payt⇒
ermcreate="http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/CreatePaymentTermEBM/⇒
V1">
  <corecom:EBMHeader languageCode="500" xmlns:corecom="http://xmlns.oracle.com/Ente⇒
rpriseObjects/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 PeopleTools > Integration Broker > Integration Setup > Service Operations.
2. Select the service operation that you need to update.
3. Select the Routing page.
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.

Message.Ver into Transform 1	This is the PeopleSoft ABM message.
Transform Program 1	This is the transformation program created for this integration.

Message.Ver out of Transforms This is the EBM message that will be sent to the third party.

7. Save the routing.

Image: Service Operations Routing - Parameters page

This example illustrates the fields and controls on the Service Operations Routing - Parameters page. You can find definitions for the fields and controls later on this page.