IBM Maximo Asset Management
Version 7 Release 6

Integrating Data With External Applications

Note
Before using this information and the product it supports, read the information in "Notices" on page 373.

This edition applies to version 7, release 6, modification 0, fix pack 8 of IBM Maximo Integration Framework and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Integrating data with external applications

The integration framework helps you to integrate application data with other applications, either within your enterprise or with external systems. The framework includes predefined content that enables integration with a number of business objects, and a tool kit that you can use to extend predefined integration content and to develop new integration points.

Integration framework overview

The integration framework helps you to integrate application data with other applications, either within your enterprise or with external systems. The framework includes predefined content that enables integration with a number of business objects, and a tool kit that you can use to extend predefined integration content and to develop new integration points.

The integration framework includes the following components and features:

- Predefined integration content
- Applications to create and configure integration components
- Support for multiple communication modes including web services, hyper text transfer protocol (HTTP), and Java Message Service (JMS) messaging
- Support for different data formats, including database interface tables, XML and JavaScript Object Notation (JSON) messages, and flat files such as comma-separated text files
- Event-based, batch, program-initiated, and user-initiated processing and context-based launch of external applications
- Support for integration to operational management products (OMPs)
- Support for clustered environments
- Support for interacting with applications that support the Open Services for Lifecycle Collaboration (OSLC) integration specification. The integration framework can enable an application to be an OSLC consumer application that can integrate with an external application that has implemented OSLC provider capabilities.

The integration framework provides multiple options for sending and receiving data. Evaluate which approach is the most efficient for your requirements when you are planning an integration. Some typical integration scenarios include:

- Load files with legacy data during an implementation.
- Synchronize master data between a product application and an external ERP application.
- Use web services to enable real-time querying of product application data by an external application.
- Call an external application to validate data that is entered in a product application.

If you want to import a large number of records from an external system, import the data in batch files or use interface tables. This approach separates the data so that multiple messages are processed as separate transactions. With a single transaction that processes synchronously, such as a web service call, limit the number of messages in a single transaction to ensure that the transaction processes

in an acceptable length of time. When you are planning an integration, evaluate which integration option is most appropriate for your requirements.

Predefined integration content includes support for a number of business objects and insert, update, and delete functions on these business objects is enabled. When you use predefined content, there are certain limitations that can affect your implementation. If business rules exist within the business object that do not allow a function, for example the delete function, the function is not available for integration. In addition to support for insert, update and delete operations, the product applications support other functions that are available as actions. Predefined integration content does not support all the actions that are available. In most cases, the Change Status action is supported through integration.

Related concepts:

"Predefined integration content" on page 60
The integration framework provides predefined integration content, including object structures, publish channels and enterprise services that support importing data from external system or exporting data to them.

Architecture

The integration framework architecture includes the data, transport, communication, and security components required to exchange information between separate applications and systems.

Framework for data exchange

The framework for data exchange includes components and tools that you can use to implement different types of integration scenarios.

Components

The framework includes predefined integration components and applications you can use to configure components. The main components are described in the following table.

Table 1. Integration framework data exchange components

Component	Description
Object structures	An object structure is the common data layer that the integration framework components use for outbound and inbound application message processing. An object structure consists of one or more related business objects that define the content of an XML message (schema).
Business objects	Application business objects are available as Representational State Transfer (REST) resources for queries and updates by using the REST API component of the integration framework.
Publish channels	A publish channel is used for sending asynchronous messages, through a JMS queue, to an external system. Publish channel messages can be initiated via an event or through the Data Export feature.
Invocation channels	Invocation channels are used for sending synchronous messages to an external system and processing the response content. The channel supports the implementation of processing logic and mapping of the request and the response. An invocation channel also allows for response data to be used to update business objects and be displayed to application users.

Table 1. Integration framework data exchange components (continued)

Component	Description
Enterprise services	An enterprise service is a pipeline for querying and importing data from an external system. An enterprise service can process data synchronously (without a queue) or asynchronously (with a queue). Enterprise services can use multiple protocols, such as web services and HTTP.
External systems	An external system is defined for the external application that you plan to integrate with. The external system identifies the communication protocol to use and which enterprise services, publish channels, and JMS queues to implement for that external system.
Endpoints and handlers	An endpoint and its associated handler routes outbound messages to an external system. The combination of an endpoint and handler specify the transport protocol, such as HTTP or web service, and provide the communication data required to reach the destination, such as a URL.
Web services	You can deploy integration framework services, such as enterprise or object structure services, as web services that external systems can invoke.
Data import and export	You can load data from either XML files or flat files, such as a comma-separated text file. You can initiate data import and export from a product application and you can schedule a batch process to perform an import as a background process. You can export one or more records to a file using a Publish Channel, where filter conditions can be applied to control the content of the data that is exported.
Content	The integration framework provides predefined content that includes object structures, corresponding enterprise services and publish channels, an external system, and predefined handlers that support different communication protocols.

Processing

As integration messages flow in and out, the framework provides options, such as Java classes, XSL mapping and processing rules, to provide message transformation logic and business rules to meet your integration requirements.

Communication

The integration framework can facilitate asynchronous or synchronous data exchange. Asynchronous messages are processed through the Java Message Service (JMS) queues. JMS queues can process messages in order of priority (sequential), or in a multi-threaded manner (continuous). Synchronous messages that require a response to the sender are not processed through the JMS queues and require a direct connection between the integration framework and the external application.

You can configure multiple communication protocols, including HTTP, web services, and JMS messaging.

When integrating with multiple external applications, you can configure different channels and services to use different communication protocols, including HTTP, web services, and JMS messaging, based on the capabilities of each individual external application.

Security

The integration framework uses the product support for J2EE authentication security so that you can configure for Enterprise Java Beans (EJBs), HTTP, and web services. You can configure authorization security for applications, objects, and standard service methods.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli®'s process automation engine.

Framework for operational management product integration

Process management products can integrate with operational management products in an automated mode using integration modules. Process management products can use the launch in context feature to integrate with operational management products in an assisted mode.

The process management product provides an action Java class that initiates the call to an integration module, and subsequently the operational management product. The process management product then processes the response from the operational management product.

A logical management operation defines the action that the process management product takes on the operational management product. The logical management operation identifies the following properties:

- The name and description of the action that it supports
- Whether processing is synchronous or asynchronous
- The input (source) and output (target) objects and fields for the transaction

The integration module provides a mechanism for a process management product to call an external operational management product. When it is started by a process management product, the integration module uses data provided by the process management product to assist in calling the operational management product service. The integration module can also return any response data to the process management product.

Operational management products provide services that integration modules can call to initiate operational management product actions.

Framework for user interface integration

You can configure the integration framework to open a window in an external application and provide data to include in the context of that window.

You can configure a console URL for any application with a web-based console, and you can configure URLs for consoles that use Java Web Start. You cannot use a launch entry to open applications that are not enabled for the web. You can configure a launch point from any product application. You can provide access to launch points as actions, as hyperlinks, and in application buttons.

You can use the same approach to open a product application window from an external application.

Enabling data export and import

Before using the integration framework to exchange data with an external application, you must configure the components required for inbound and outbound communication. These procedures describe the minimum configuration steps required in order to validate that you can export data to a file and import data from a file.

Preparing the system

To enable integration, you must perform some configuration tasks related to system properties, JMS queues, and cron tasks.

Configuring JMS queues

JMS queues are configured on the application server. JMS queue configuration can be automatic or manual on the WebSphere® Application Server. JMS queue configuration is always a manual procedure on the WebLogic Server.

Procedure

1. Confirm that the JMS queues are configured.
2. Confirm that message-driven beans are enabled for the continuous queue.

Configuring integration properties

Before you enable the integration framework, modify the default settings for integration properties to settings that are appropriate to your environment.

Procedure

1. In the System Properties application, filter for the properties you want to configure. If you filter for `mxe.int` in the **Property** field, all integration properties are listed.
2. Select the `mxe.int.dfltuser` property and verify that the user account that is specified is a valid system user account. Modify the property value if necessary. The property specifies the default login ID that is used for all integration transactions that are processed by an inbound JMS queue.
3. Optional: You can configure a global directory where a single file location can hold integration-related files on the file system. The Maximo® Asset Management servers must have access to the global directory on the file system. For example, when you use a file-based endpoint that does not have a configured file location, the file that is generated by the endpoint is placed in a default directory in the global directory. You can configure the name of the global directory in the `mxe.int.globaldir` system property.

Activating the cron task for JMS queues

For asynchronous processing, integration messages are placed in JMS queues. Cron tasks poll the JMS queues at frequent intervals and then process messages that are found in the queues.

Procedure

1. In the Cron Task Setup application, search for the JMSQSEQCONSUMER cron task.
2. Verify that the cron task is configured to poll both the SEQQIN and the SEQQOUT queues.
3. Set the Active check box for each queue.
4. Click **Save**.

5. Click the **Reload Request** action.

Exporting data to a test file

To validate that outbound processing is correctly configured and enabled, run a test export.

About this task

To test outbound processing, this task uses the data export feature to export a person record to a file. The task uses the following predefined integration components:

- External system: EXTSYS1
- Publish channel: MXPERSONInterface, which uses the MXPERSON object structure
- Endpoint: MXXMLFILE

Procedure

1. In the Systems tab, ensure that the **Enabled** check box is selected for the EXTSYS1 external system.
2. In the Publish Channels tab, filter for the MXPERSONInterface publish channel.
3. In the Details section for the MXPERSONInterface publish channel, specify MXXMLFILE in the **End Point** field.
4. Click **Data Export**.
5. In the Data Export window, specify 1 in the **Count** field to limit the export to just one record.

Results

If a location is not specified in the `FILEDIR` property, the XML file is exported to the location that you specified in the **`mxe.int.globaldir`** property.

Importing data from a test file

To validate that inbound processing is correctly configured and enabled, run a test import.

Procedure

1. Open the test file that you exported in a text editor and perform the following modifications:
   a. Change the <PublishMXPERSON> tag to <SyncMXPERSON> to change the operation for inbound processing.
   b. Add a suffix to the LASTNAME value, such as LASTNAME_TEST, to make it easy to verify the change when you import the test file.
   c. Save the file.
2. On the System tab of the External Systems application, ensure that the **Enabled** check box is selected for the EXTSYS1 external system.
3. On the Enterprise Services tab, filter for the MXPERSONInterface enterprise service.
4. Select the MXPERSONInterface row, and clear **Use Continuous Queue**.
5. Click **Save**.
6. Select the MXPERSONInterface row and click **Data Import**.
7. In the Data Import window, specify the following values:

a. Select the **XML file** option.
b. In the **Specify Import File** field, navigate to the modified test file and select it.

8. Click **OK**.
9. In the People application, filter for the test record and verify that the value in the **Last Name** field includes the suffix that you added.
10. Delete the suffix to restore the record to its original value and click **Save**.

Integration components

Depending on the scope and requirements of your implementation, you can create new components or copy, modify, or extend the predefined components.

Duplicating an integration component and modifying the copy has several advantages. The copy of an integration component becomes a user-defined entity, and modification restrictions that apply to the predefined component do not apply to the copy. The original version of the component is unchanged. To avoid modifying a component while it is being used for transaction processing, perform all modifications in administrator mode.

Object structures

An object structure is the common data layer that the integration framework components use for outbound and inbound application message processing. An object structure consists of one or more related business objects that define the content of an XML message (schema).

An object structure provides the message content for channels and services and enables application-based import and export. In addition, an object structure, on its own, can be called as a service, supporting the Create, Update, Delete, Sync, and Query operations.

Object identification

Identification of what data to include in an object structure requires some knowledge of applications and their associated objects. Review the data model of an application to determine which objects, database tables, and views contain the data that you must transfer to and from an external system.

You must find out which objects populate the database tables. Generally, a one-to-one relationship exists between an object and a database table. However, in some cases, multiple objects write data to a single table.

A predefined object structure is provided for exchanging data in the person object with an external system. If no predefined object structure existed, to create one, you determine that the data that you require resides in the PERSON, PHONE, EMAIL, and SMS tables. The database tables have the same name as the corresponding objects. Include these objects in the object structure that you create. The name of the object structure must begin with an alphabetic character.

When an object structure contains multiple objects, the object structure is arranged as a hierarchy, with a root object (the top or main object) and child objects. An object structure can support any number of levels of objects in its hierarchy. You must specify a valid parent-child for relationship for all objects under the top level of the hierarchy and you cannot reference the same object more than once in the

same branch of the hierarchy. When you create an object structure, start by adding the main (top-level) object. You can then add more objects as child objects of the parent object.

Alternate keys

Inbound message processing relies on the key field(s) of an object to find an existing record in the system to support updates and deletes. The default processing relies on the primary key to retrieve existing records. Sometimes a primary key is unknown to an external application and an alternate key (known by the external system) is defined to support the updating and deleting of records by the external system.

An inbound message normally uses the primary key of an object to look up and process records that already exist in the system. However, sometimes a primary key is an internally-generated value that is not available to the external system. In such cases, you can define an alternate key for an object and the external system passes the alternate key field(s) that the integration framework uses, instead of the primary key fields, to retrieve the data for that object.

You can specify an alternate key for an object either at object level or at object structure level. If you specify an alternate key at object level, all object structures that include the object use the same alternate key. If you specify an alternate key at the object structure level, you can use different object structures to specify a different alternate key for each external data source that the object uses. You define the index that make up an alternate key for an object in the Database Configuration application. After you create the index, you can select it as an alternate key for the object.

During inbound processing, the integration framework processes the alternate key and primary key in the following order:

- Processes the alternate key of the object structure, if one is configured.
- Processes the alternate key of the object, if one is configured and an alternate key is not configured for the object structure.
- Uses the object primary key if an alternate key is not configured for either the object structure or the object.

After you specify an alternate key, inbound messages can fail if you change or drop the alternate key index.

Object fields

The integration framework and the external system exchange a subset of the data columns in the objects within the object structure. Subject to certain restrictions, you can control the content of the object fields within the message by including persistent and nonpersistent columns of individual objects.

A business object can have many fields, most of which may not be necessary for an integration scenario. When you configure an object structure, you can select which fields to include in integration messages and improve the performance of message transfer between applications. By default, the objects in an object structure include all the persistent columns of the objects. A persistent column is a data field that an object uses to write data to a database table as part of its processing. To control the content size, you can exclude persistent columns that you are not exchanging with external applications. Only the included persistent columns are part of the XML

message for outbound messages. For inbound messages, only the object columns that are included in the object structure are updated. Do not exclude any column that is part of a primary or alternate key.

By default, an object structure excludes most nonpersistent columns in the component objects. A nonpersistent column is a temporary data field that an object uses for calculations or temporary storage. You can include additional nonpersistent columns in the object structure. For example, objects that contain the persistent column DESCRIPTION also contain the nonpersistent column DESCRIPTION_LONGDESCRIPTION. Most predefined object structures include this nonpersistent column because many integration scenarios require long description fields. If this field is not included, it is not part of the integration messages. If you change the message content of an object structure that is being used for an interface table, the interface table must be regenerated to reflect the updated content of the object structure.

Interface table and flat file considerations

If you use an object structure for exchanging data with interface tables or flat files, you must ensure that the object structure does not contain duplicate column names.

You must select the **Flat Supported** check box for any object structure that you intend to use for interface table or flat file integration scenarios. When this option is set, messages are checked to ensure that every column for every object in the object structure has a unique name. If duplicate column names exist, you can create an alias field name for on the of the duplicate names. Modifying the alias ensures that all column names are unique and the system can generate the interface table or flat file without errors. Interface tables require that all columns that are included in the corresponding object have an alias name of 18 or fewer characters.

Modification of a predefined object structure

There are certain restrictions if you modify a predefined object structure.

You can add objects to a predefined object structure, but you cannot delete predefined objects from the object structure. To avoid this restriction, you can copy the predefined object structure to create a user-defined object structure, and delete objects from the copy.

You can include and exclude persistent and nonpersistent columns, subject to the standard validations that apply during integration processing. Outbound messages include the columns for objects that you add to an object structure. Test inbound messages to ensure that any columns that you added are processed successfully. If the additional object columns are not processed successfully, add an object structure processing class to handle the inbound processing.

If you use the object structure in interface tables or flat files, check for column duplications. If you use interface tables, regenerate the table for every enterprise service or publish channel that uses the modified object structure.

Configuring an object structure

You can create new object structures and during that process you can generate schema files, include or exclude fields, and resolve alias conflicts. You can also specify inbound setting restrictions, set advanced configuration properties, and configure application authorization.

Creating object structures:

When you create an object structure, you define a group of related objects to be part of an integration message that you exchange with external applications. You can identify data fields for each business object that determines the content of integration messages.

About this task

During the configuration, you can define the Java classes, if needed, that process inbound and outbound messages. You also can define an application that provides authorization rules for the integration messages processed through the object structure. If the object structure is intended for querying only and you do not want to use it for updating, select the **Query Only** option. If the main object in the object structure has a relationship to itself as a child object, set the **Self Reference** option.

Procedure

1. In the Object Structures application, click **New Object Structure**.
2. In the **Object Structure** field, specify an object structure identifier.
3. Optional: If you are using the object structure for query operations, select the **Query Only** check box.
4. Optional: If you are using interface tables or flat files to exchange data between the integration framework and an external system, select the **Support Flat File Structure** check box.
5. In the **Consumed By** field, specify the module that uses the object structure.

Option	Description
INTEGRATION	Integration framework
MIGRATIONMGR	Migration manager
REPORTING	Reporting
OSLC	OSLC

6. In the Source Objects table window, click **New Row**.
7. Enter values in the following fields to create a business object hierarchy:
   - **Object**
   - **Parent Object**
   - **Reporting Description**
   - **Relationship**
   - **Object Order**
8. Click **Save Object Structure**.

What to do next

If the **Alias Conflict** check box is selected on the object structure record, you can add or modify an alias to correct the duplicated field names in your source objects. You can also specify the persistent and nonpersistent fields that you want to exclude and include from the object structure.

Configuring an alternate key:

To configure an alternate key, create a unique index in the Database Configuration application and reference this index as the alternate key for an object structure or for a specific object.

About this task

If you set the alternate key at the object level, the key applies to all uses of the object in any object structure. If you set the alternate key within the object structure, the key applies to the object only when it is accessed through the selected object structure.

Procedure

1. Identify the field(s) of an object to use as the alternate key.
2. Select the object in the Database Configuration application.
3. Create a unique index for the field(s) in the Indexes tab.
4. Specify this index in the **Alternative Key** field in one of the following tabs:
   a. In the Object tab of the Database Configuration application if you want to apply the alternate key to the object for all external data sources.
   b. In the Object Structures tab of the Database Configuration application if you want to apply the alternate key to this specific usage of the object.

Including nonpersistent fields in the object structure:

You can include nonpersistent field data in an integration message. By default, the business object nonpersistent fields are excluded from the object structure definition. Business objects use nonpersistent fields for calculations or for temporary storage of data.

Before you begin

If you want to change a predefined object structure, make a duplicate of that object structure to create a user-defined version that is suitable for modification.

Procedure

1. In the Object Structures application, select the object structure that you want to update.
2. Select the business object that contains the nonpersistent field that you want to include.
3. From the **Select Action** menu, select **Exclude/Include Fields**.
4. Click the **Non-Persistent Fields** tab to display the nonpersistent fields in the business object.
5. Specify whether you want the nonpersistent field to be included or excluded.

Option	Included
Include the field	Selected
Exclude the field	Cleared

6. Click **OK**.

What to do next

You can generate schema files and view the XML structures of an object structure web service.

Excluding persistent fields from the object structure:

You can exclude persistent field data that you do not want to map to an integration message. By default, the business object persistent fields are included in the object structure definition. Business objects use persistent fields to write processing data to a database.

About this task

You cannot exclude a field that is part of a primary key. If you exclude a persistent field from a predefined object structure, the associated object might not function properly during inbound message processing. Test your inbound messages to ensure that an excluded persistent field does not impact the object processing.

Procedure

1. In the Object Structures application, select the object structure that you want to update.
2. Select the business object that contains the persistent field that you want to exclude.
3. From the **Select Action** menu, select **Exclude/Include Fields**.
4. Click the **Persistent Fields** tab to display the persistent fields in the business object.
5. Specify whether you want the persistent field to be excluded or included.

Option	Excluded
Exclude the field	Selected
Include the field	Cleared

6. Click **OK**.

What to do next

You can generate schema files and view the XML structures of an object structure web service.

Resolving alias conflicts:

An object structure that contains multiple objects and supports flat files or interface tables, cannot have duplicate field names for any of the fields in its objects. You must resolve any field name (alias) conflicts before you can generate interface tables and flat file records.

About this task

If an alias conflict exists, the **Alias Conflict** check box is selected on the object structure record. You can change an alias only if the **Support Flat File Structure** check box is selected on the record, indicating that the data is processed using interface tables or flat files.

Procedure

1. When an alias conflict exists, select a business object in the Source Objects table.
2. Select the **Add/Modify Alias** action. If a duplicate alias exists for a field, the corresponding **Duplicate** check box is selected.
3. To update a duplicate alias:
   a. Click **View Details** for the duplicate alias.
   b. Specify a new value in the **ALIASNAME** field.
   c. Click **OK**.

What to do next

After you resolve all alias conflicts, you can generate interface tables and flat file records. If you use interface tables, you must regenerate all tables that use the updated object structure. To regenerate interface tables, select the **Create Interface Tables** action in the External Systems application.

Setting restrictions on fields in inbound messages:

Standard integration processing sets the values in object fields with the corresponding values from an inbound message. You can set a field as restricted if you do not want the value to be updated by inbound messages, for example for a field with an internal ID or where a processing class provides the logic to set the field.

Procedure

1. From the **Select Action** menu, click **Inbound Setting Restrictions**.
2. In the Inbound Setting Restrictions window, select the object that you want to apply setting restrictions to. The Inbound Setting Restrictions table refreshes with a list of the fields that are configured for the selected object.
3. Select the **Restricted** check box for any field that you do not want to be updated with values in inbound messages.
4. You can select the **Override** check box to remove the restrictions set for a field. You cannot override the restriction set on some fields, for example for a field with a system-generated ID.
5. Click **OK**.

Setting advanced configurations for an object structure:

You can set advanced configurations for an object structure to change some default processing behavior for integration messages. Advanced configurations include configuring how key fields are processed for child objects, whether an event on a child object activates a corresponding event on the parent object, and whether auto-generated data is deleted.

Procedure

1. Select the **Advanced Configuration** action.
2. Deselect the **Exclude Parent Key Attributes** check box for any object where you want these attributes to be included for child objects. When checked (the default), key fields that exist in a child object are not included in the section of the message for the child object if the same field is part of the key of the parent object. If you deselect this option, key fields for a child object are always included and the field is included in both the child object section of the message and the parent object section.

3. Select the **Skip Key Update** check box if you want to disable updates of primary keys.
4. Deselect the **Delete Auto-generated Data** check box for any object where you want to retain this data. When checked (the default), integration processing always deletes any child-level data that is automatically created by the business object logic when the parent object is created. If you deselect this option, any additional data that is auto-generated is retained.
5. Check the **Propagate Event** option for any object where you want an event on a child object to trigger an event on the main object. When you configure a publish channel to send messages based on an object event, the event listener is configured for the main object of the object structure. In some cases, an update to a child object does not trigger an event on the main object and no message is initiated. Check this option if you want an update to the child object to trigger an event to the main object without updating the main object. If a child object includes logic that triggers an event to its parent object, you cannot enable or disable it with this configuration.

Channels and services

Channels and services reference an object structure for their message content and enable the synchronous and asynchronous exchange of data with external systems. Two types of channel process outbound messages; publish channels and invocation channels. Three types of service process inbound messages; object structure services, enterprise services, and standard services.

Publish channels

A publish channel is used for sending asynchronous messages through a JMS queue to an external system. Publish channel messages can be initiated via an event or through the Data Export feature.

The integration framework includes predefined publish channels or you can configure new publish channels. When configuring a publish channel, you must associate it with an object structure, and, optionally, enable an event listener. You must also configure the publish channel with an external system to determine where the message is delivered to.

You can also configure processing rules, Java processing classes, or XSL mapping to customize transaction processing of the publish channel.

Configuring a publish channel:

To use a publish channel for data export, you must create the publish channel, associate it with an object structure, and enable an event listener. You must also configure an endpoint that routes the transaction to a specified external system. You can also configure publishing rules, Java processing classes, or XSL mapping to customize transaction processing.

Creating publish channels:

You can create a publish channel record to send integration messages to an external system.

Before you begin

Before you create and configure a publish channel, use the Object Structures application to configure the object structure that you intend to associate with the publish channel.

Procedure

1. In the Publish Channels application, click **New Publish Channel**.
2. In the **Publish Channel** field, specify a name for the publish channel.
3. In the **Object Structure** field, specify the object structure to use with the publish channel. The Object Structure Sub-Records section refreshes with details of the objects that are contained in this object structure.
4. Optional: To restrict publish channel events to the main object of the object structure, select **Skip Different Object Types**. This option can help to avoid publishing errors when there are multiple objects associated with an object structure.
5. Optional: If you intend to use an interface table as the data source, specify its name in the **Interface Table** field. The object structure must be configured to support flat files to be used with interface tables
6. Optional: If you intend to customize outbound processing logic, specify the paths for the Java Classes and the XSL style sheet in the following fields:
   a. **Processing Class**
   b. **User Exit Class**
   c. **XSL Map**
   d. **Event Filter Class**

   Any Java classes you specify must be part of the application EAR file. An XSL file can be within the EAR file or located on an accessible file system.
7. Optional: You can configure processing rules for the publish channel.
8. Optional: If necessary, clear the **Retain Objects** check box to prevent the publish channel from processing business object-based rules.
9. Click **Save Publish Channel**.

What to do next

You can enable a publish channel listener to direct the integration framework to build and process the selected publish channel. You can also use the External Systems application to associate the publish channel with an external system and identify an endpoint for delivering messages from the publish channel.

Enabling publish channel listeners:

You enable an event listener on a publish channel to monitor for processing activity on the associated publish channel objects. Publish channel processing is initiated when an event occurs on the main object of the associated object structure.

Procedure

1. In the Publish Channels application, select the publish channel that you want to configure with an event listener.
2. Select the **Enable Event Listener** action.
3. Click **OK** to enable the publish channel listener. The details for the publish channel refresh and the **Event Listener** check box is now selected.

What to do next

Select **Disable Event Listener** action if you want to disable the event listener at any time..

Invocation channels

Invocation channels define the processing logic and mapping of inbound and outbound data, which enables the integration framework to call external applications and process responses. No predefined invocation channels are provided.

Creating invocation channels:

You can create an invocation channel record to send outbound data from an object structure to an external system and to process responses from the external system.

Before you begin

You must include the defined processing class, user exit class, and XSL mapping files in the application EAR file. You also must define an XSL mapping filename path that is accessible by the application server.

Procedure

1. In the Invocation Channels application, click **New Invocation Channel**.
2. In the **Invocation Channel** field, specify an invocation channel identifier.
3. Enter values in the following fields:
   - **Adapter**
   - **Endpoint**
4. Optional: If this invocation channel processes responses from an external application, select the **Process Response** check box.
5. In the Service Request Configuration table window, enter values in the following fields:

Option	Description
Request Object Structure	The object structure that is used to define the content for outbound data processing.
Request Processing Class	The Java class file that is used when the invocation channel requires predefined outbound processing logic.
Request User Exit	The class file that the invocation channel uses to customize the predefined outbound processing logic
Request XSL File	The XSL file that is used to customize predefined outbound invocation channel mapping.

6. Optional: If you selected the **Process Response** check box, enter values in the following fields in the Service Response Configuration table window:

Option	Description
Response Object Structure	The object structure that is used to define the content for the response.

Option	Description
Response Processing Class	The Java class file that is used when the invocation channel requires predefined inbound processing logic for the response.
Response User Exit Class	The class file that the invocation channel uses to customize the predefined inbound processing logic for the response.
Response XSL File	The XSL file that is used to customize predefined inbound invocation channel mapping for the response.

7. Click **Save Invocation Channel**.

What to do next

You can view the XML schema of the object structure using a URL with the following format:

http://*localhost*:*port*/*meaweb*/schema/service/*object_structure_name*

Configuring an action to call an invocation channel:

The integration framework provides a default action class that you can configure as a system action. By providing this action class, you can configure a user interface control, an escalation, or a workflow to invoke an external service using an invocation channel.

Procedure

1. Create an invocation channel in the Invocation Channels application.
2. Create an action in the Actions application.
3. Specify an object for the action. This object must be the same as the main object of the request object structure of the invocation channel and the main object of the application, workflow, or escalation that invokes the action.
4. Specify `Custom Class` in the **Type** field.
5. Specify the name of the custom class in the **Variable** field. You can use the name of the default class provided for this purpose, `psdi.iface.action.InvokeCustomClass`, or an alternative class name if you created your own custom class to invoke an external system.
6. Specify values in the **Parameters/Attributes** field. Specify the values in the following order, and separate each value with a comma:
   a. Required: The name of the invocation channel to use. The value must be precisely the same as the name of the invocation channel.
   b. Optional: The name of the relationship to use if the main object of the response object structure is different from the main object of the request object structure in the invocation channel. If the response object is the same as the request object, no relationship is required.
   c. Optional: If you specified a relationship, specify the action to apply. The default action is Add, which creates records. To update existing records, specify Change as the action. If the request and response object structures are the same, the objects are updated if updated fields are mapped into the response object structure.
7. Specify whether to apply the action to all applications, to workflows, or applications.

8. Save the action.

What to do next

Associate an application, workflow, or escalation with the action. The main object is passed to the action class and then to the object structure of the invocation channel to form the request XML.

Invoking an external system from an application:

After configuring an action class to invoke an external system (via an invocation channel), you can configure a button in an application to trigger the invocation action. You can also extend the action class to display the results of the transaction in an application dialog box.

Before you begin

You must create an invocation channel and an action class before you add the invocation action to an application. If you intend to show the results of the invocation, create a Results (dialog) window in advance.

Procedure

1. Open the application in the Application Designer application. The main object for this application must be same as the main object for the invocation channel and the action that you intend to call from the application.
2. Add a **Button Group** control to the workspace from the Control Palette. The **Button Group** control automatically adds a **Pushbutton** control to the workspace.
3. Click **Properties** to open the **Pushbutton Properties** window.
4. Specify a name for the button in the **Value** field, for example `Invoke External System`.
5. Specify the **Control ID** for the Results window in the **Target ID** field.
6. Specify a method in the **Value** field that calls the invocation channel and redirects the results to the Results window. For example:

   ```
   InvokeChannelCache.getInstance().getInvokeChannel(channelName)
   .invoke(metaData, mbo, mbo.getMboSet(rel), action, null);
   ```

   Where:

   - The `channelName` value is the name of the invocation channel.
   - The `mbo` value is the name of the object.
   - The `rel` value is the name of the relationship (if applicable).
   - The `action` value is Add.

Object structure services

When you configure an object structure, no additional configuration is necessary to make it available as a service or a REST resource.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Enterprise services

An enterprise service is a pipeline for querying external data and importing data from an external system. An enterprise service can process data synchronously (without a queue) or asynchronously (with a queue). Enterprise services can use multiple protocols, such as web services and HTTP.

An enterprise service has data processing layers that transform data and apply business processing rules to data before the it reaches the target objects. When the inbound message reaches the object structure layer, the XML message must be in the format of the object structure schema. The integration framework can then can process the message successfully.

You can configure an enterprise service to implement the following transaction processing:

- Processing rules
- User exit Java processing class
- Enterprise service processing class
- XSL mapping

The gateway is the entry point for enterprise service messages, excluding those that are delivered directly from an external queue to the integration queue. For some integration scenarios, it can be useful to configure gateway properties, for example if properties such as system IDs are provided within the XML message rather than in the header information. You can configure an interpreter Java class to change the external system ID, the enterprise service ID, or other JMS message header properties that are provided in an inbound message. Alternatively, you can configure gateway properties in the Enterprise Services application.

Configuring an enterprise service:

When you create an enterprise service, you associate it with an object structure. You can also configure gateway properties that are added to the header information of inbound messages.

Creating an enterprise service:

You can create an enterprise service record to receive inbound data from an external system. On the enterprise service, you can identify the data transformation and processing rules that the integration framework uses to receive inbound data from an external system.

Before you begin

Before you create and configure an enterprise service, use the Object Structures application to configure the object structure that you associate with the enterprise service.

About this task

When the **Use External Schema** check box is selected, the **Split Tag**, **External Schema File**, and **External Schema Element** fields are editable. The external schema values identify the schema location and external root element name.

Procedure

1. In the Enterprise Services application, click **New Enterprise Service**.
2. In the **Enterprise Service** field, specify an enterprise service identifier.
3. Enter a values in the following fields:
   - **Object Structure**
   - **Adapter**
4. Optional: Enter values in the following fields:

Option	Description
Operation	Determines how the Enterprise Service processes data. For example, you can synchronize objects or create new objects. You can also update, delete, or query existing objects.
Multiplication Control	Determines the cross-reference control that the enterprise service uses to multiply an inbound message for multiple organizations or sites.
Interface Table	Reflects the content of the enterprise service object structures.
Split Tag	Identifies whether the received message contains multiple instances of a document. For example, a single message can contain ten purchase orders. The split process handles each of these instances individually. The application writes multiple files to the inbound queue. The syntax that you use to identify these node values should have a fully qualified XPATH expression.

5. Optional: To use an external schema, select the **Use External Schema** check box.
6. Optional: You can customize inbound enterprise service processing logic by completing the following steps:
   a. In the **Processing Class** field, enter a class value if the enterprise service requires predefined inbound processing logic.
   b. In the **User Exit Class** field, enter a class value the enterprise service uses to customize the predefined inbound processing logic.
   c. In the **XSL Map** field, enter a value to customize the predefined inbound enterprise service mapping.
7. Click **Save Enterprise Service**.

What to do next

You can use the External Systems application to associate the enterprise service with an external system.

Configuring additional object structures for an enterprise service:

An enterprise service must be associated with a primary object structure. You can add additional object structures to an enterprise service to support updating

additional data that is not included in the primary object structure of the enterprise service. Based on the defined processing order, the enterprise service processes additional object structures before it processes its primary object structure.

Procedure

1. In the Enterprise Services application, display the service for which you want to add an object structure.
2. Select the **Add/Modify Additional Object Structure** action and click **New Row**.
3. Enter values in the following fields:
   - **Object Structure**
   - **Processing Order**
4. Optional: You can customize inbound enterprise service processing logic by completing the following steps:
   a. In the **Processing Class** field, enter a class value if the enterprise service requires predefined inbound processing logic.
   b. In the **User Exit Class** field, enter a class value the enterprise service uses to customize the predefined inbound processing logic.
   c. In the **XSL Map** field, enter a value to customize the predefined inbound enterprise service mapping.
   d. In the **Multiplication Control** field, enter a value to define the cross-reference control that the enterprise service uses to multiply an inbound message for multiple organizations or sites.
5. Click **OK**.

Adding gateway properties to an enterprise service:

The gateway is the entry point for enterprise service messages, excluding those that are delivered directly from an external queue to the integration queue. An interpreter Java class can be implemented that can change the external system (SENDER) or the enterprise service (INTERFACE) and can set other JMS header properties as needed. In the Enterprise Services application, you can set gateway properties either as hard-coded values or by specifying XML tags.

Before you begin

Review the following points about the settings in the Gateway Properties before you configure the gateway properties:

- If you select the **XML Tag** check box and leave the **Value** field null, the gateway uses the name of the root element in the XML message as the value for the corresponding property.
- If you select the **XML Tag** check box and enter a tag name in the **Value** field, the gateway uses the value for that tag as the value for the corresponding property.
- If the tag appears multiple times in the XML message, the adapter uses the value of the first occurrence of the tag. If you clear the **XML Tag** check box and enter a data value in the **Value** field, the gateway uses that data as the value for the corresponding property.

Procedure

1. In the Enterprise Services application, display the service for which you want to add a gateway property.
2. Select the **Gateway Properties** action.
3. In the Gateway Properties for Enterprise Service table window, click **New Row**.

4. In the **Property** field, specify the name of the property that you want to include in the message.
5. In the **Value** field, type the data that you want to use as the value of the property.
6. Optional: Specify whether you want to use the XML root element as the value of the corresponding property.

Option	XML Tag
Use the XML root element	Selected
Do not use the XML root element	Cleared

7. Click **OK**.

Standard services

A standard service is based on an annotated method in an application. A standard service is specific to the method annotated for an object and is not reusable for other objects. You can deploy a standard service as a web service. You can access a standard service by using the REST API.

A standard service is a service that an application provides for performing a specific operation on an object. Standard services are available only for methods that are properly annotated within the service. The service schemas that are generated for standard services are used only by the corresponding actions.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Endpoints and handlers

An endpoint and its associated handler routes outbound messages to an external system. The combination of an endpoint and handler specifies the transport protocol to use, such as HTTP or web service, and provides the communication data required to reach the destination, such as a URL.

For a data export using a publish channel, the cron task for the outbound queue invokes the handler. For a data export using an invocation channel, the invocation channel invokes the handler directly. The handler uses the metadata properties of the message to determine the external system (for a publish channel transaction) and any override values configured for the endpoint properties. The handler then sends the data to the destination that is specified by the endpoint with which the handler is associated. A single handler can have multiple endpoints, where each endpoint contains different parameters.

Endpoints and handlers are used for outbound integration only. However, the interface table endpoint and handler also support the creation of interface tables, which are needed for inbound integration.

Endpoints provide the execution parameter values that the handler uses to perform its function. The metadata included with the endpoint definition, together with the outbound integration message, are used by the handler at the time of execution. Some handlers, not all, have predefined endpoints. These can be modified or new endpoints configured using the endpoint application. The following table lists predefined endpoints.

Endpoint	Handler	Description
MXFLATFILE	FLATFILE	Writes outbound integration messages in a flat file format to a specified directory location.
MXIFACETABLE	IFACETABLE	Writes outbound integration messages to interface tables.
MXXMLFILE	XMLFILE	Writes outbound integration messages in an XML file format to a specified directory location.
MXCMDLINE	CMDLINE	Implements the CMDLINE handler. Takes a command and endpoint as input and uses the SSH protocol to invoke the command on the target system and return the results.

A number of predefined handlers are provided. For several of these handlers, corresponding endpoint definitions have been configured. The endpoint definition provides the metadata values for the handler parameters, such as a directory name for the FLATFILE handler.

Configuring an endpoint

In the End Points application, you can create an endpoint that specifies how outbound transactions are delivered and the handler that routes them.

Creating endpoints:

You create an endpoint to identify a target location and the transport mechanism that the integration framework, or deployment manager uses to publish data, or to invoke a service.

Procedure

1. In the End Points application, click **New End Point**.
2. In the **End Point** field, specify an identifier for the endpoint.
3. In the **Handler** field, specify a value. The **Consumed By** field displays the information that is associated with the specified endpoint handler.
4. In the **Properties for End Point** window, click **View Details** of the endpoint property field and perform the following actions:
   a. Add a unique value to the **Value** field to identify the endpoint property.
   b. Add a value to the **Encrypted Value** to identify whether the endpoint property needs additional security for storage and display. You can update an encrypted value only on an endpoint with a password property.
   c. Select the **Allow Override** check box to identify whether you can overwrite the code for the endpoint property. Select this check box when you use an invocation channel processing class.
5. Click **Save End Point**.

What to do next

You can add a handler to specify how to route outbound data to a specific endpoint location in a specific format. You can use the endpoint in the following integration framework applications:

- External Systems
- Publish Channels

- Invocation Channels
- Integration Modules
- Logical Management Operations

Adding a handler to an endpoint:

You can add a handler to an endpoint record to specify how to route outbound data to a specific endpoint location. You can also add a handler to define the data format that is used in the data transfers. When you create a handler, you must identify the specific Java™ class file that contains the processing logic that you need for data transfers.

About this task

You cannot modify or delete a predefined handler.

Procedure

1. In the End Points application, select the endpoint that you want to update with handler information.
2. Select the **Add/Modify Handlers** action.
3. Click **New Row**.
4. Enter values in the following fields:
   - **Handler**
   - **Handler Class Name**
   - **Consumed By**
5. Click **OK**.

Writing custom handlers:

You can write a custom handler and associate it with an endpoint, for example to support communication with an FTP server.

Procedure

1. To write a custom handler, implement the `RouterHandler` interface.
2. Specify the following method to return a list of properties that the handler uses to send data to the endpoint:
   ```
   getParameter()
   ```
   The method returns a list of RouterPropsInfo objects. The isCrypto attribute in the RouterPropsInfo object indicates whether to encrypt the property value while storing data. For password properties, the value of this attribute is True.
3. Specify the following method to send data to the specified endpoint:
   ```
   sendData(Map metaData, byte[] data, Map destinationMap)
   ```
   The method returns the following information:
   - Metadata provides information about the external system and the interface.
   - Data is the XML data.
   - DestinationMap specifies the endpoint.

Predefined endpoint handlers

Predefined handlers are provided that you can associate with an endpoint. Additionally, you can create and register custom handlers when needed.

Enterprise bean handler:

The Enterprise Java Bean (EJB) handler is a Java component that consists of enterprise bean clients. The handler publishes a set of properties that a client uses to communicate with and deliver an integration message to a target client. The target client can run on the local application server or on a remote application server.

To establish a connection, the remote Java class and the home Java class must be available in the class path of the handler. If the client is on a remote application server that is different from the handler application server, the client jar file reference must be in the class path of the handler. The handler picks up the context factory class name from the local application server when the enterprise bean client is on a remote application server that is the same as the handler application server.

CONTEXTFACTORY property

This required property specifies a J2EE context factory class name. The documentation for your application server contains the name of the default context factory to use.

The CONTEXTFACTORY uses the following property when the target client runs on an IBM WebSphere Application Server:

```
com.ibm.websphere.naming.WsnInitialContextFactory
```

EJBEXIT property

This optional property is used for customization and specifies the fully qualified name of a custom Java class that implements the `EJBExit` interface.

If you do not specify a value for this property, the `DefaultEJBExit` interface is executed and attempts to resolve the enterprise bean method signature and parameters.

If the enterprise bean client has its own method signature and parameters, create a Java class that contains your version of the `EJBExit` interface and implementations of the following methods:

```
public Class[] getClassParams()
```

The getClassParams() method returns the method signature in the form of an array of Java classes.

```
public Object[] getObjectParams(byte[] data, String interfaceName, Map
String,? metaData)throws MXException
```

The getObjectParams() method returns the parameters of the enterprise bean business method in the form of an array of Java objects.

```
public void responseOk(Object response)throws MXException
```

The responseOk() method is called after a successful enterprise bean invocation.

```
public void responseError(Exception e) throws MXException
```

The responseError() method is called with the originating exception as a parameter if an error is encountered during enterprise bean invocation.

The following code illustrates what your implementation of getClassParams() looks like when the enterprise bean client has a business method with a byte array and a string:

```
Class[] classParams = {byte[].class, String.class};
return classParams;
```

The following code illustrates what your implementation of getObjectParams () looks like when the enterprise bean client has a business method with a byte array and a string:

```
byte[] data = ...;
String ifaceType = ...;

Object[] objParams = {data,ifaceType};
return objParams;
```

Complete one of the following actions to identify the location of the package structure for the `EJBExit` class file:

- Place the class in the Java package structure in the `applications/maximo/businessobjects/classes` directory.
- Modify the `mboweb\webmodule\META-INF\MANIFEST.MF` class path to include the package structure.
- Rebuild the application EAR file and include the `EJBExit` class file.

JNDINAME property

This required property specifies the name by which the enterprise bean client is registered in the Java Naming and Directory Interface (JNDI) tree on the WebSphere Application Server. The filename is `ibm-ejb-jar-bnd.xml` and the property is set to

```
<ejbBindings xmi:id="Session_enterpriseservice_Bnd"
  jndiName="ejb/maximo/remote/enterpriseservice">
    <enterpriseBean xmi:type="ejb:Session"
      href="META-INF/ejb-jar.xml#Session_enterpriseservice"/>
</ejbBindings>
```

METHODNAME property

This required property specifies the public business method that is exposed by the enterprise bean client that is invoked by this handler.

PROVIDERURL property

This required property specifies the URL of the target application server on which the enterprise bean is running. The system then maps to the `java.naming.provider.url` property and creates the InitialContext object.

The following example is an IBM WebSphere Application Server provider URL.

`corbaloc:iiop:`*`hostname`*`:`*`iiopport`*

If the handler and the target enterprise bean are running on the same application server instance, do not specify this property because it defaults to the local server URL.

USERNAME and PASSWORD properties

The user name and password properties correspond to the `java.naming.security.principal` (USERNAME) and `java.naming.security.credentials` (PASSWORD) properties that are used to create the InitialContext object.

Flat file handler:

The FLATFILE handler converts an outbound integration message into a flat file and writes it to a directory that has a configurable location. Flat files contain ASCII data in the form of rows and columns. Each line of text constitutes one row, and a separator character separates each column in the row. The FLATFILE handler encodes outbound flat files in the standard UTF-8 format.

The FLATFILE handler can be used only with publish channels, not invocation channels. The object structure associated with the publish channel must be configured to support flat files. You must resolve all alias conflicts for the object structure and format the XML message according to the object structure schema before writing the message to a flat file.

Naming conventions

File names require the following format.

`externalsystemname_publishchannelname_uniqueidentifier.dat`

- externalsystemname is the identifier of the system (the value of MAXVARS.MXSYSID).
- publishchannelname is the name of the publish channel.
- uniqueidentifier is a number based on current system time.

The following example file name indicates that the file goes to the external system EXTSYS1 and was published through the MXASSETInterface publish channel:

`EXTSYS1_MXASSETInterface_10971102668641498.dat`

The first two lines of the file contain header information. The first line has the following format:

```
externalsystemname <separator> publish channel name <separator> [action]
<separator> langcode
```

The second line of the file contains the names of the columns, separated by the separator character. The column names are the same as the names in the corresponding interface table.

Flat file format

If the data in the flat file contains the flat file delimiter character, the data adds the text qualifier, which is " (quotation marks). If the data contains quotation marks, the handler escapes the quotation marks. You cannot use quotation marks as the delimiter character.

The following example data uses a comma (,) as a delimiter. The INVOICEDESC value, (Rotating Custom Item, No 71), contains a comma. When the flat file is written, the INVOICEDESC value is enclosed in quotation marks.

```
EXTSYS1,MXINVOICEInterface,Add
INVOICENUM,INVOICEDESC,PONUM,VENDOR,CONTACT,PAYMENTTERMS
1071,"Rotating Custom Item, No 71",1000,A0001,,
```

The following example data uses a comma (,) as a delimiter. The INVOICEDESC value (Rotating "Custom" Item No 71) contains double quotation marks. When the flat file is written, double quotation marks in INVOICEDESC data ends with quotation marks, and the entire string is wrapped in quotation marks.

```
EXTSYS1,MXINVOICEInterface,Add
INVOICENUM,INVOICEDESC,PONUM,VENDOR,CONTACT,PAYMENTTERMS
1071,"Rotating ""Custom"" Item No 71",1000,A0001,,
```

The following example data uses a comma (,) as a delimiter. The INVOICEDESC data (Rotating "Custom" Item, No. 71) contains the delimiter character and double quotation marks. When the flat file is written, the INVOICEDESC value appears in the code.

```
EXTSYS1,MXINVOICEInterface,Add
INVOICENUM,INVOICEDESC,PONUM,VENDOR,CONTACT,PAYMENTTERMS
1071,"Rotating ""Custom"" Item, No. 71",1000,A0001,,
```

Flat file properties

The FLATFILEDIR property is an optional property that specifies the location of flat files on a server. The location must exist on the server where the JMS CRON task for the outbound queue is running, or on an accessible shared network drive. The default value points to the global directory. To specify a location for the global directory, configure the mxe.int.globaldir property in the System Properties application.

In a multitenancy environment, the FLATFILEDIR property is required and is made available by the system provider.

The FLATFILESEP property is a required property that specifies the character that separates the columns in each row.

HTTP handler:

The HTTP handler is a Java component that consists of properties. The handler delivers an outbound integration message as an XML document to a URL by using HTTP or HTTPS protocols. The HTTP handler also evaluates the response code received from the external system.

HTTPEXIT property

This optional property is used for customization and specifies the fully qualified name of a Java class that interprets the HTTP response. This property also helps implement the code that is required for an external system to interpret the HTTP response.

The Java class must be available in the application EAR file and must be in the class path of the handler.

Property	Value
Java class	`DefaultHTTPExit.java`
Package	`psdi.iface.router`
HTTPEXIT property	`psdi.iface.router.DefaultHTTPExit`

If you do not specify a value for this property, the `DefaultHTTPExit` exit class is executed and implements the `psdi.iface.router.HTTPExit` interface. The Java class has the following key methods:

- processResponseData()

  This method has the following signature:

  ```
  public void processResponseData(int responseCode, String responseMsg,
  byte[] msgBodyData) throws MXException
  ```

  The default implementation compares the response code from the external system to a range of valid codes (values 200 through 299). If the response code falls outside that range, the system assumes that the message was not delivered to the external system. An exception occurs and the message remains in the queue.

  If you need additional processing for a specific implementation, extend the default implementation and override the processResponseData () method. As an alternative, you can implement the `psdi.iface.router.HTTPExit` interface. If the response that is received from the external system does not pass the validation in this class, the overriding method must issue an exception.

  If you do not define a value for this property, the default implementation of `HTTPExit` is run.

- getURLProperties()

  This method has the following signature:

  ```
  public Map String, String getURLProperties(Map String,? metaData, byte[]
  data, Map String,MaxEndPointPropInfo httpInfo)
  ```

  This method returns the map of URL properties that are added to the URL in the form *url?*`prop1=value1&...` The default implementation returns a null value.

- getHeaderProperties()

  This method has the following signature:

  ```
  public Map String, String getHeaderProperties(Map String,? metaData,
  byte[] data, Map String,MaxEndPointPropInfo httpInfo)
  ```

  This method returns a map of the HTTP header properties for the request. The default implementation returns a null value unless a header property map is associated with the metadata map that has the HEADERPROPS key.

- transformPayloadToFormData()

  This method has the following signature:

  ```
  public Map String, String transformPayloadToFormData(Map String,?
  metaData, byte[] data,Map String,MaxEndPointPropInfo destinationMap)
  ```

  This method converts the XML payload to data. The default implementation returns a null value.

CONNECTTIMEOUT property

This optional property specifies the connection timeout value in milliseconds.

READTIMEOUT property

This optional property specifies the read timeout value in milliseconds.

HTTPMETHOD property

This required property specifies a valid HTTP method that is executed by the endpoint. Valid HTTP methods are GET, POST, PUT, and DELETE.

HTTPHEADER property

This optional property can add a comma-separated list of names and values to the header section of HTTP messages. The list includes name and value information in the format of Headername1:Headervalue1, Headername2:Headervalue2. If no value is provided in the property, custom code can inject the values into the transaction context during invocation channel processing.

URL property

This optional property specifies a valid URL to which XML data can be posted or where an HTTP GET operation can be performed.

USERNAME and PASSWORD properties

If the URL requests basic authentication, these properties specify the required values. Both values are MIME encoded and are passed to the URL.

IFACETABLE handler:

The IFACETABLE handler consists of several properties. This handler writes an outbound integration message to an interface table in a local or remote database. There are no Java exit classes for this handler.

Only publish channels can use the IFACETABLE handler. Invocation channels cannot use this handler.

ISREMOTE property

This required property is a Boolean value that specifies whether interface tables are available in the local database or in a remote database. A value of 0 (false) indicates that the interface tables are available in the local database in the system schema. You do not have to enter any other handler properties. In the predefined MAXIFACETABLE handler, the value of this property is 0. A value of 1 (true) indicates the interface tables are in a remote database. If necessary, specify values for all the handler properties.

DRIVER property

This property specifies the JDBC driver to connect to a remote database that contains the interface tables. This property applies only when the value of the ISREMOTE property is 1.

URL property

This property specifies the JDBC URL and applies only when the value of the ISREMOTE property is 1. The following example contains the location, port number, and database name:

`jdbc:db2://hostname:port/maximodb`

USERNAME and PASSWORD properties

If access to the remote database requires a user name and password, these properties specify those values. These properties apply only when the value of the ISREMOTE property is 1.

JMS handler:

The JMS handler delivers outbound integration messages to a JMS-compliant messaging system that supports a JMS queue or topic.

The messaging models have the following characteristics:

- Point-to-point messaging (one to one): A sender generates messages and places them in a queue. Only one receiver can obtain the message from the queue.
- Publish-subscribe (one to many): A publisher generates messages and places them in a topic. Multiple subscribers can retrieve messages from the topic.

The messaging system represents a queue or topic that is available on the local application server, on a remote application server, or on a remote dedicated queuing system such as IBM® WebSphere MQ. To use this handler, enable the messaging systems by using JMS. The messaging system is distinct from the standard internal queues that reside on the local application server.

CONFACTORYJNDINAME property

This required property specifies a Java object that is used to create connections to a JMS provider. Before the system can connect to a queue or topic, it must obtain a reference to a connection factory.

DESTINATIONTYPE property

This optional property specifies the JMS destination type; queue or topic. The following table lists the DESTINATIONTYPE options and their associated values.

Destination	Value
Topic	javax.jms.Topic
Queue	javax.jms.Queue

DESTJNDINAME property

This required property specifies the name by which the JMS queue or topic is registered in the application server Java Naming and Directory Interface (JNDI) tree.

CONTEXTFACTORY property

This property specifies the initial context factory class name. The property is not required when the JMS handler is communicating with a JMS provider that shares the same initial context factory as the application server of the handler. When the handler and the JMS provider share a WebSphere Application Server, they share the initial context factory class. The context property value is required when the handler and the JMS provider do not share a WebSphere Application Server.

ISCOMPRESS property

This required property specifies whether the message is compressed before it is placed it into a queue or topic. Compression is an optimization technique that delivers smaller messages to a queue or topic. The following table lists the ISCOMPRESS options and their associated values.

Option	Value
Do not compress data	0
Compress data	1

Compressed messages must be extracted after they are received. Extract the messages by creating the appropriate JMS receiver or subscriber component and placing Java decompression logic within the receiver or subscriber. Use the standard Java Inflater() class that is part of the `java.util.zip` package. The default compression uses the standard Java Deflator() class.

ISTEXT property

This optional property specifies whether the JMS handler will deliver messages to another queue in text format.

Option	Value
Deliver messages in default (bytes) format	0
Deliver messages in text format	1

JMSEXIT property

In a multitenancy environment, customization by using Java classes can be implemented only by the system provider and may not be supported in your environment.

This optional property is used for customization and specifies the fully qualified name of a Java class that runs the JMSExit interface. The Java class must implement the getMessageProperties() method that is defined in the JMSExit interface. The Java class must be in the class path for the application server or in the application EAR file.

You can use this option change or add properties in the JMS message. If this property does not contain a value, the header attributes for the message are not changed when the message is delivered to the external queue or topic.

PROVIDERURL property

This required property specifies a local or remote URL where the JMS provider can be accessed. If the target JMS provider is local to the application server of the handler, the property is not required. The following property is an example of a PROVIDERURL value on a WebSphere Application Server:

`corbaloc:iiop:`*`hostname`*`:`*`iiopport`*

PROVIDERUSER and PROVIDERPASWORD properties

These properties are used for the JMS provider authentication. The properties map to the connectionFactory.createConnection(provideruser,providerpassword) API in JMS.

USERNAME and PASSWORD properties

These properties correspond to the java.naming.security.principal (USERNAME) and java.naming.security.credentials (PASSWORD) properties used for creating the InitialContext object.

Web service handler:

The web service handler is a Java client that can invoke any document-literal web service that is WS-I BP 1.1 compliant. The outbound integration message forms the payload (SOAP body) and the handler provides the SOAP headers and envelope. This handler operates independent of the container.

This web service handler supports an early implementation of web services and is provided to maintain backwards compatability with existing web services. If you are implementing new web services, configure them to use the web service handler for JAX-WS.

MEP property

This optional property specifies the message exchange pattern for the web service. The property supports the following values. If you do not provide a value, the default value, the sendreceive value is used.

Value	Web Service Operation Type
sendreceive	Request and response
sendrobust	Request with void or fault response
fireandforget	Request only, no response, or fault

ENDPOINTURL property

This required property specifies a valid web service URL on which to invoke a web service. You can use the WSEXIT class to override the value specified via the user interface just before the web service is invoked.

SERVICENAME property

This required property specifies the name of the web service deployed in the URL.

SOAPACTION property

This optional property specifies the value of the SOAPAction HTTP header to be used when invoking the web service. The default value is an empty string. To set a value for the property, view the WSDL file for a web service to determine the action and specify this value. You can use the WSEXIT class to override the value specified in the user interface before you invoke the web service.

SOAPVERSION property

This optional property specifies the version of the SOAP specification used during web service call. Valid values are SOAP11 and SOAP12.

HTTPVERSION property

This optional property specifies the version of the HTTP protocol for web service invocations. The valid values are HTTP/1.0 and HTTP/1.1. If you do not provide a value, the system uses the default value, HTTP/1.1.

HTTPCONNTIMEOUT property

This optional property specifies the connection timeout value in milliseconds. The default value for this property is 60000 milliseconds.

HTTPREADTIMEOUT property

This optional property specifies the read timeout value in milliseconds. The default value for this property is 60000 milliseconds.

USERNAME and PASSWORD properties

If the specified web service is secured (if HTTP basic authentication is enabled), specify a user name and password.

WSEXIT property

In a multitenancy environment, customization by using Java classes can be implemented only by the system provider and may not be supported in your environment.

This optional property is used for customization. It specifies the fully qualified name of a custom Java class that implements the psdi.iface.router.WSExit interface. The property defines the following methods:

The responseOk() method is called after a successful invocation of the external web service.

```
public void responseError(Exception e) throws MXException
```

If an error occurs when the web service is called, the responseError() method is called with the originating exception as a parameter.

The default implementation of the WSExit interface is `psdi.iface.router.DefaultWSExit`.

Web service handler (JAX-WS):

The WEBSERVICE-JAX-WS handler is a Java client that can invoke any document-literal web service that is WS-I BP 1.1 compliant. The outbound integration message forms the payload (SOAP body) and the handler provides the SOAP headers and envelope.

This web service handler supports the current implementation of web services. If you are implementing new web services, configure them to use this web service handler.

MEP property

This optional property specifies the message exchange pattern for the web service. The property supports the following values. If you do not provide a value, the default sendreceive value is used.

Value	Web Service Operation Type
sendreceive	Request and response
sendrobust	Request with void or fault response
fireandforget	Request only, no response, or fault

ENABLEAPPCONTEXT property

Set this property to 1 (true) when implementing support for WS-* policies.

ENDPOINTURL property

This required property specifies a valid web service URL on which to invoke a web service. You can use the WSEXIT class to override the value specified via the user interface just before the web service is invoked.

SERVICENAME property

This required property specifies the name of the target web service that the handler invokes.

SOAPACTION property

This optional property specifies the value of the SOAPAction HTTP header to be used when invoking the web service. The default value is an empty string. To set a value for the property, view the WSDL file for a web service to determine the action and specify this value. You can use the WSEXIT class to override the value specified in the user interface before you invoke the web service.

SOAPVERSION property

This optional property specifies the version of the SOAP specification used during web service call. Valid values are SOAP11 and SOAP12.

HTTPCONNTIMEOUT property

This optional property specifies the connection timeout value in milliseconds. The default value for this property is 60000 milliseconds.

HTTPREADTIMEOUT property

This optional property specifies the read timeout value in milliseconds. The default value for this property is 60000 milliseconds.

HTTPHEADER property

This optional property can add a comma-separated list of names and values to the header section of HTTP messages. The list includes name and value information in the format of Headername1:Headervalue1, Headername2:Headervalue2. If no value is provided in the property, custom code can inject the values into the transaction context during invocation channel processing.

USERNAME and PASSWORD properties

If the specified web service is secured (if HTTP basic authentication is enabled), specify a user name and password.

WSEXIT property

In a multitenancy environment, customization by using Java classes can be implemented only by the system provider and may not be supported in your environment.

This optional property is used for customization. It specifies the fully qualified name of a custom Java class that implements the psdi.iface.router.WSExit interface. The property defines the following methods:

The responseOk() method is called after a successful invocation of the external web service.

```
public void responseError(Exception e) throws MXException
```

If an error occurs when the web service is called, the responseError() method is called with the originating exception as a parameter.

The default implementation of the WSExit interface is `psdi.iface.router.DefaultWSExit`.

CFGXMLPATH property

This property is obsolete. Do not use it.

XML file handler:

The XML file handler is a Java component that writes an outbound integration message into a file in XML format.

FILEDIR property

This optional property specifies where the handler creates the XML files. The default value is mxe.int.globaldir/xmlfiles. The file location must be accessible by the JMS CRON task for the outbound queue for publish channel messages. For Invocation Channel messages, the file location must be accessible by any Maximo Asset Management servers where an invocation channel can be initiated.

PRETTYPRINT property

This required property specifies whether the handler formats the XML file. The valid values are 0 and 1. A value of 1 prompts the handler to pretty print format the xml file. Publish channel, invocation channel, and invocation API file names have the following formats:

externalsystemname_publishchannelname_uniqueidentifier.xml

invocationchannelname_uniqueidentifier.xml

- externalsystemname is the identifier of the system (the value of MAXEXTSYSTEM.EXTSYSNAME).
- publishchannelname is the name of the publish channel.
- uniqueidentifier is a number based on current system time.

For example, the file name MX_MXASSETInterface_10971102668641398.xml indicates that the file was generated to send data to the external system EXTSYS1. The file name also indicates that the file contains the MXASSETInterface publish channel.

Command line handler:

The CMDLINE handler takes a command and an endpoint as input. The CMDLINE handler uses the SSH protocol to run the command on the target system and return the results.

A metadata parameter is passed during a system invocation when the handler is called. The parameter is a map that contains the name of the endpoint that represents the target system. The caller can target any system at run time and pass the endpoint to the command handler. The caller uses whatever configuration the endpoint has at the time of the invocation.

CMDLINE handler properties

The CMDLINE handler has the following properties:

- CMDTIMEOUT – The timeout value for command execution
- CONNTIMEOUT – The timeout value for the connection
- USERNAME – The user name for the connection
- PASSWORD – The password for corresponding user name
- HOST – The host name of target where command is run
- PORTNO – The port number of target where the command is run
- IGNORESETUPERR – A boolean value that indicates whether to ignore an error running the setup command
- RETRYINTERVAL – The time to wait between retrying a command
- MAXRETRY – The number of attempts to run a command before returning an exception
- SSHEXIT – The Java exit class that can be implemented to customize processing of the handler

Command data parameters

The data parameter is a byte array representation of an XML document. The data parameter contains the following information:

- The tags that correspond to the setup command

- The working directory
- The command to run
- Any substitution parameters

Available tags

The following tags are available:

- CLWORKINGDIR – A directory to change (`cd`) to on the remote system before the command is run.
- CLSSETUPCMD – A setup command to be run before the main command. Use this tag for any environmental setup that must occur on the remote system before the main command is issued.
- CLCMDPATTERN - A string that defines the pattern of the command to be run. The format of this pattern is similar to the `java.text.MessageFormat` class. An example is `ls -l {0}`, where {0} represents a parameter that is substituted.
- CLSUB0 - The value to substitute into positions that are marked by {0} in the CLCMDPATTERN.
- CLSUB1- The value to substitute into positions that are marked by {1} in the CLCMDPATTERN.
- CLSUBn - The value to substitute into positions that are marked by {n} in the CLCMDPATTERN. A CLSUBn tag must correspond to each substitution position in the CLCMDPATTERN tag.

Command results

The return byte array representation of an XML document contains the results of the command. The XML document contains tags that correspond to the return value, STDOUT and STDERR.

The following tags are available:

- CLRETURNCODE – The return code from the remote command.
- CLRESPONSEOUT – The data that is returned by the remote command in STDOUT tag.
- CLRESPONSEERR – The data that is returned by the remote command in STDERR tag.

Integration web services

External systems can use integration framework web services to send messages or queries to the Maximo database. You can use an integration framework service as the source for a web service and then deploy this service to communicate with external systems.

The web services are based on the Java API for XML Web Services (JAX-WS). In earlier releases, the integration framework also supported AXIS2 web services but these are no longer supported. During upgrade, any existing AXIS2 web services are automatically converted to JAX-WS web services.

Web service sources

The source of a web service can be an object structure service, a standard service, or an enterprise service.

Object structure services

You can create an object structure web service from a predefined or a user-defined object structure service. Object structure web services support create, update, delete, sync, and query operations. Create and query operations also support responses. The response to the create operation can be based on the primary or alternate key of the primary object that is defined in the object structure. The response to the query operation is provided in the XML schema format of the object structure.

Standard services

You can create a standard web service from methods that are annotated in application services. To use methods as web services, annotated methods, such as the ChangeStatus method, must exist within an application. A single standard web service is created for each application service and all annotated methods within the service are the web service operations. The input and output parameters of the methods are associated with the input and output parameters for the standard web service.

Enterprise services

You can create an enterprise web service from a predefined or a user-defined enterprise service. Enterprise web services support additional exit processing, business rules, and transformations. A service exists for each operation that is contained in an enterprise service record (one service per operation). With the exit processing layer, you can map an external schema XML to the object structure XML for both the invocation and the response. Enterprise web services provide response content for the create and query operations that are processed in the queue. The enterprise service definition has a ProcessResponse flag that indicates whether the service supports a response.

JMS queue settings for enterprise web services:

Enterprise web services can use the Java Message Service (JMS) queue to process XML messages, or you can specify that the service bypasses the JMS queue.

JMS queue-based processing provides asynchronous message processing. For enterprise services that support synchronous processes, you must specify non-queue-based message processing. The integration framework processes XML messages from external applications based on the queue settings that are defined for the enterprise web service:

Queue-based message processing
A queue-based enterprise web service processes XML messages and writes these messages into the configured, inbound JMS queue. When the message is moved into the JMS queue, the external application is released from the invocation. The integration framework processes the messages and saves them in the Maximo database.

An enterprise web service cannot use the JMS queue if the service includes create or query operations that require a response to the external application.

Non-queue-based message processing
A non-queue-based enterprise web service processes messages from the integration framework to the object processing layer and saves information, when applicable, to the Maximo database. After the enterprise web service

completes message processing, the integration framework sends a response to the external application. You must specify non-queue-based processing if the enterprise web service supports query or create operations that require a response to the external application.

Web service deployment options

After you create a web service, you have a choice of deployment options. If you deploy a web service to the product web service container, the deployment is automatic. If you deploy a web service to an application server web service container, the deployment is manual.

You can deploy web services to one container type only. You cannot deploy some web services to a product web service container and deploy other web services to an application server web service container. Deployment to the web service container for the product is the quicker option, because the deployment is automatic. This type of deployment does not require redeployment of the Maximo EAR files or a restart of the application server.

To invoke a web service on the application server, you must deploy the service to the application server web service container. Integration with an application server can provide a web service with access to additional services, such as enhanced security policies. If you choose this type of deployment, the information is added to the `deplmodule.dar` file. You must then add this file to the application server deployment directory, rebuild the EAR files, and restart the application server.

You can convert existing web services that are deployed to the product web service container to redeploy them to the application server web service container.

Web service deployment actions

When you deploy a web service, a number of events occur, including the generation of the XML schema and the web services description language (WSDL) file for the web service.

When you deploy a web service, the following events occur:

- The XML schema is made available for a new web service or the schema is regenerated for an existing web service.
- The WSDL file is made available for the service interface.
- The web service is deployed for the selected service.
- If Universal Description Discovery and Integration (UDDI) registry properties are configured, the web service is registered in the UDDI registry.

The deployed web service is available at the following URL:

`http://hostname:port/meaweb/services/web service name`

- `host:port/meaweb` is the value of the integration web application URL property.
- `web service name` is the name of the service for which the web service is deployed.

The list of deployed web services is available in the Web Services Library application.

Schema generation

You can generate the schema and view the XML for any web service in the Web Services Library application. You can also regenerate the schema to ensure that the schema is updated to reflect any changes that you make to the service.

A web service data structure is based on its associated object structure (for object structure services and enterprise services) or method signature (for standard web services). The web service data structure is provided in a standard XML representation, as a schema. This schema is used to create the WSDL file for the service.

If you change the data structure that is associated with a web service, regenerate the schema to update its schema and the WSDL file. The following changes to the data dictionary change the object structure:

- Adding new fields to a table
- Changing the type of a field
- Dropping fields from a table
- Changing a field from optional to required

The XML does not contain fields that are marked for exclusion in the object structure.

Before you deploy a web service, you must generate the schema to provide updated schema information to the WSDL file. When you generate a schema, input elements are displayed in the XML Data window and output elements are displayed in the Response XML window.

To access a generated schema, you can use a URL.

Enterprise service and object structure schemas

Use a URL that is similar to the following example in which MXSR is the object structure name that is associated with the service:

`http://localhost:port/meaweb/schema/service/MXSR`

For compatibility with version 7.5, URLs that are similar to the following example are still supported:

`http://localhost:port/meaweb/schema/service/MXSRService.xsd`

Standard service schemas

Use a URL that is similar to the following example in which ITEM is the name of the service.

`http://localhost:port/meaweb/schema/service/ss/ITEM`

For compatibility with version 7.5, URLs that are similar to the following example are still supported:

`http://localhost:port/meaweb/schema/service/ss/ITEMService.xsd`

If you set the `mxe.int.resolveschema` global property to true in the System Properties application, all include files are resolved. The entire schema content resides in a single file.

Generation of a Web Services Description Language file

A WSDL file is generated during the deployment of a web service. The WSDL file describes the web service, provides its location, and specifies the operations that the service makes available.

The WSDL file specifies the XML structure of the input and output messages for the operation, based on the XML schema. The file also specifies the URL for the web service, and the operations that the web service makes available.

A client program needs the schema definitions and WSDL file to generate client stubs. The client program uses a programming language, such as Java or C#, to call the web service.

You can view a generated WSDL file in a browser at `http://localhost:port/meaweb/wsdl/service_name?wsdl`, where *service_name* is the name of the service, such as MXASSET.

UDDI registration

You can register deployed web services in a UDDI registry. UDDI is an XML-based registry for publishing and locating WSDL files that describe web service applications.

You can register deployed web services in a UDDI registry by configuring the following global properties in the System Properties application:

System property	Description
mxe.int.uddipuburl	UDDI registry publish URL
mxe.int.uddiinqurl	UDDI registry inquiry URL
mxe.int.uddiname	UDDI registry user ID
mxe.int.uddipassword	UDDI registry password

If you specify values for the publish URL property and the inquiry URL property, the web service is registered in the UDDI registry. To bypass UDDI registration, do not specify any values for these properties. Only the model for the WSDL file is registered in the UDDI registry. The businessEntity, businessService, and bindingTemplate values are not registered. The UDDI registration entry contains the URL to the WSDL file.

Creating and deploying web services

You can create a web service based on any object structure service, standard service, or enterprise service that is defined for the system. You then deploy the web service to the product web service container or to the application web service container.

Creating a web service:

When you create a web service, an external system can send web-based messages and queries to the associated service without configuring additional communication protocols or services.

Before you begin

Perform the following checks to ensure that the service you want to use is available to create a web service:

- For an object structure service, verify that the **Consumed By** field is set to `INTEGRATION` in the Object Structures application.
- For a standard service, ensure that a method is annotated in the application service for each operation that you want to use in the web service. Only those methods that are properly annotated are accessible in the Web Services Library application
- For an enterprise service, ensure that the service is associated with an external application in the External Systems application.

Procedure

1. In the Web Services Library application, select the appropriate **Create Web Service** action.
2. Choose the service to use by selecting the corresponding **Source Name** check box.
3. Optional: In the **Name** field, specify an identifier for the web service.
4. For a web service that is based on an enterprise service, specify whether you want the web service to be queue-based or to bypass the JMS queue.
5. Click **Create**.

What to do next

You must deploy the web service before you can use it to process inbound messages and queries. You can also generate the schema and view the XML structures of any selected web service.

Deploying a web service to the product web service container:

After you create a web service, you must deploy it before it can start processing XML messages. When you deploy a web service to the product web service container, the deployment process occurs automatically and does not require a server restart.

Procedure

1. In the Web Services Library application, select the web service that you want to deploy.
2. Select the **Deploy to Product Web Service Container** > **Deploy Web Service** action.
3. Click **OK**.

Deploying a web service to the application server web service container:

A deployment file is generated when you deploy a web service to the application server web service container. You must rebuild the Maximo EAR file to include the deployment file and restart the application server to activate the web service.

About this task

The `deplmodule.dar` deployment file is in the directory that is specified by the **`mxe.int.globaldir`** system property.

Procedure

1. In the Web Services Library application, select the web service that you want to deploy.
2. Select the **Deploy to Application Server Web Service Container** > **Generate Deployment File Entries** action. This action generates an entry in the deployment file, `deplmodule.dar`.
3. Click **OK**.
4. Copy the `deplmodule.dar` file from the integration framework global directory to the product deployment directory.
5. In the deployment directory, build the `maximo.ear` file.
6. In the administrative console for the application server, deploy all web services in the `deplmodule.dar` file:

a. Stop the application server.
b. Redeploy the `maximo.ear` file.
c. Restart the application server.

Updating schema information:

If you change the data structure of an integration service, update the schema information to the WSDL file to update the web service. If you do not regenerate the schema, the schema used by the web service can differ from the structure defined for the associated integration service.

Procedure

1. After modifying the data structure of a web service, click **Generate Schema, WSDL, and View XML**. If you do not regenerate the schema, the web service is not aware of changes that you make.
2. For web services that are deployed to the application server web service container, you can automate schema updates in the System Properties application.
   a. Set the value of the mxe.int.containerdeploy property to 1 to deploy web services to the application server web service container.
   b. Set the value of the mxe.int.wsdlincludeschema property to 1 to ensure that schema information is included as part of WSDL files.
   c. Set the value of the mxe.int.resolveschema property to 1 to ensure that the system resolves all included fields into a single file.
   d. Set the value of the mxe.int.wsdlcurrentschema property to 1 to provide up-to-date schema content as part of a WSDL file.

Sending Maximo Asset Management error messages to integration users

When an external application invokes a Maximo Asset Management web service and encounters a business logic error such as Invalid Site, the text of the Maximo Asset Management error message is not included in the exception that is sent to the client. To include the error message text in the web service response, you can add a webservices.unify.faults property to WebSphere Application Server.

Procedure

1. Log in to WebSphere Application Server administrative console and navigate to **Servers** > **Server Types** > **Websphere application servers**.
2. Select the server name.
3. In the server infrastructure section, select **Java and Process Management**.
4. Select **Process definition**.
5. In the Additional properties section, select **Java virtual machine** and then **Custom properties**.
6. Create the webservices.unify.faults property.
   a. Click **New**.
   b. Set the name to `webservices.unify.faults`.
   c. Set the value to **false**. This setting enables the error message text to be propagated from the error condition that occurs during integration.
7. Save your changes.

Web service interactions overview

An interaction can start a web service and send data to it from an application. The interaction can then display data returned from the web service and save this data to the application database.

To implement interactions requires knowledge of:

- XML schemas
- Web services
- The integration framework
- Customizing applications and application user interfaces

An interaction can manage the following processes:

- Prepare a request for a web service.
- Invoke a web service from an application.
- Retrieve results from the web service in the form of a response.
- Display the results in an application.
- Optionally, apply the response data into the system database.
- Report any errors that occur during the request or response processes.

Two applications are provided to help you create and manage interactions:

- Create and configure interactions in the Create Interaction application.
- Review, modify, and delete interactions in the Interactions application.

After you create an interaction, users can perform the following tasks from the application user interface:

- Start the interaction.
- View and, if configured, change the parameters of the request to the web service.
- Invoke the web service by sending the request.
- Optionally, view data received from the web service and commit this data to the database.

Creating interactions:

The Create Interactions wizard application guides you through all the steps required to create and configure an interaction. After you complete the wizard process and test the interaction, users can start working with it immediately with no requirement for additional configuration or deployment.

Before you begin

Before you run the Create Interactions application, ensure that you have all the information required for the configuration. During the process, you must specify the web service, define the structure of messages, and map the data between the application and the web service.

A logging tool is provided to log configuration activity and maintain a log file of the items that are configured. To generate a detailed log of the configuration, in the Logging application, set the interaction logger to DEBUG mode.

Procedure

1. To configure the web service for the interaction, in step 1 of the process:
   a. Specify the URL for the WSDL file for the web service.
   b. When the screen refreshes with information from the WSDL file, specify one port for the interaction.
   c. Specify one operation for the interaction and check **Process Response** if you want the web service to return data to the application during the interaction.
2. Review the contents of the request to the web service in step 2, and modify the request object structure by removing any unnecessary elements.
3. Optional: Review the contents of the response from the web service in step 3, and modify the response object structure by removing any unnecessary elements. This step is included only if you checked the **Process Response** option in step 1.
4. To configure the application for the interactions, in step 4 of the process:
   a. Specify the application that uses the interaction.
   b. Configure the application binding for the interaction, including the main object, the signature option, the interaction mode, and the user interface components.
   c. Specify the security groups that are authorized to initiate the interaction.
5. Configure the Request tab of the Interactions window in step 5, including specifying the fields that users can see and whether they can edit them.
6. Optional: Configure the Response tab of the Interactions window in step 6, including specifying the fields that users can see and whether they can edit them. This step is included only if you checked the **Process Response** option in step 1.
7. You can map information from the application to the web service in step 7. Mapped information is entered automatically into the request when users start the interaction.
8. Optional: You can map information from the web service to the application in step 8. If you check the **Commit Response** option, the mapped information is saved automatically to the database.
9. Review the configurations in the final step of the process.

What to do next

To monitor the time it takes to execute the interaction, in the Logging application, set the integration logger to either INFO mode or DEBUG mode.

You can view the interaction and modify mapping information in the Interactions application. If you intend to modify an interaction, select the **Deactivate Interaction** action before you make any changes and reactivate it after you complete the modification.

External systems

You can configure the external system that the integration framework communicates with. You can configure external systems for external applications within or outside your enterprise. If you copy a predefined external system, it copies the channels and services that are configured for it. You can configure a new external system to use existing JMS queues.

Configuring an external system

To configure an external system, identify the external system and associate it with the channel or service used to process transactions. You must also configure the JMS queues that the external system uses and you can configure integration controls to support customization with processing rules.

Creating an external system:

You create an external system to exchange data with external applications. When you create an external system, the application copies the integration controls that are defined for the corresponding publish channels and enterprise services. You then can specify default control values that apply to a particular external system.

Before you begin

Before you create an external system, define the queues and endpoint that the external system uses.

Procedure

1. In the External Systems application, click **New External System**.
2. In the **System** field, specify an external system identifier.
3. Optional: If the external system sends outbound messages, complete the following steps:
   a. Specify a value in the **Outbound Sequential Queue** field.
   b. Specify a value in the **End Point** field.
4. Optional: If the external system receives inbound messages, complete the following steps:
   a. Specify a value in the **Inbound Sequential Queue** field.
   b. Specify a value in the **Inbound Continuous Queue** field.
5. Click **Save External System**.

What to do next

You must enable at least one publish channel or enterprise service before message processing can occur.

Enabling an external system:

You can enable an external system after you configure the external system record and when you are ready to begin integration framework message processing. You also can disable an external system to stop all inbound and outbound message processing.

Before you begin

You must enable at least one publish channel or enterprise service before any message processing can occur.

About this task

When you disable an external system, the integration framework does not accept inbound messages or send outbound messages. Additionally, you cannot use the external system data export and data import features. Only messages that are in the queues are processed.

Procedure

1. In the External Systems application, select the system that you want to update.
2. Specify whether you want the external system to be enabled or disabled:

Option	Enabled
Enabled	Selected
Disabled	Cleared

3. Click **Save External System**.

Enabling a publish channel:

You must enable a publish channel that is associated to an external system before it can be used to publish event-based messages to an external system. By default, the publish channel records that are associated with an external system are disabled. A disabled publish channel prevents the external system from processing outbound integration framework messages.

Procedure

1. In the External Systems application, select the system that you want to update.
2. On the Publish Channels tab, specify whether you want the publish channel to be enabled or disabled.

Option	Enabled
Enabled	Selected
Disabled	Cleared

3. Click **Save External System**.

Associating a publish channel with an external system:

You can associate a publish channel with an external system to synchronize the asset management object data with the external application data. The channels that you create and associate to the external system contain the outbound message processing logic.

About this task

You can define the endpoint that the channel uses. If you do not define an endpoint at the publish channel level, data is moved to the endpoint location that is defined at the external system level. You also can enable the associated publish channel when you are ready to perform outbound integration framework message processing.

Procedure

1. In the External Systems application, select the external system that you want to update.
2. On the Publish Channels tab, click **New Row**.
3. In the **Publish Channel** field, enter a value.
4. Optional: In the **End Point** field, specify a value.
5. Optional: Specify whether you want the publish channel to be enabled or disabled:

Option	Enabled
Enabled	Selected
Disabled	Cleared

6. Click **Save External System**.

Selecting publish channels for the external system:

You can associate multiple publish channels with an external system to synchronize the asset management object data with the external application data. The channels that you create and associate to the external system contain the outbound message processing logic.

Procedure

1. In the External Systems application, select the system that you want to update.
2. On the **Publish Channels** tab, click **Select Channel**.
3. Choose one or more publish channels by selecting the corresponding **Publish Channel** check boxes.
4. Click **OK**.
5. Click **Save External System**.

What to do next

You can define the endpoints that the channels use. If you do not define an endpoint at the publish channel level, data is moved to the endpoint location that is defined at the external system level. You must enable the associated publish channels before you can perform integration framework message processing.

Adding an endpoint to a publish channel:

You can define the endpoint that a publish channel uses to determine where the outbound data is published. Endpoints identify a target location and the transport mechanism for outbound data publication. If you do not define an endpoint at the publish channel level, data is moved to the endpoint location that is defined at the external system level.

Before you begin

You must associate a publish channel to an external system.

Procedure

1. In the External Systems application, select the external system that you want to update.
2. On the Publish Channels tab, select the publish channel for which you want to add an endpoint.
3. In the **End Point** field, specify a value.
4. Click **Save External System**.

What to do next

You can enable the associated publish channel when you are ready to perform outbound integration framework message processing.

Importing file-based data:

You can use the data import feature to load data from either XML or flat, delimited files, to update the Maximo database. You can preview and validate the data prior to loading it and committing it to the database. You can choose to manage errors with the Message Reprocessing application or by extracting errors to a file format that is the same as the imported file format.

Before you begin

Before data can be imported, if you plan to import data from a flat file, such as a .csv file, the enterprise service object structure must support flat file structures. Ensure that the **Support Flat File Structure** check box is selected on the associated object structure record in the Object Structures application. You also must enable both the external system and the enterprise service before you can import data.

About this task

The data that you import must be in a delimited flat file, such as comma separated, or an XML file format. The data import process can use a predefined or user-defined enterprise service.

Procedure

1. In the External Systems application, display the external system that contains the enterprise service from which you want to import data.
2. On the **Enterprise Services** tab, select the enterprise service from which you want to import data.
3. Click **Data Import**.
4. Optional: Select the **Import Preview** check box to examine the data before importing and committing the data to the database. Use the preview option to sample data records. This feature is not intended to support a large file containing hundreds of records. Processing synchronously processes the file to the business objects and returns any error messages encountered, without committing any updates to the database.
5. Specify the type of file that you want to use for the file import.

Option	Description
XML File	Imported data is in XML format.
Flat File	Imported data is in a delimited flat file. If necessary, modify the **Delimiter** and **Text Qualifier** values.

6. In the **Specify Import File** field, enter the file name path the imported file uses for identification and storage.
7. Select the **File-based Error Management** check box if you want to manage any errors you encounter through a file in the same format as the file being imported. This option is an alternative to managing errors with the Message Reprocessing application
8. Click **OK** to begin the import data process.

What to do next

When the data import is executed, the file that is selected for import is formed into multiple messages and dropped into the inbound queue that is configured for the

enterprise service and its corresponding external system. The messages are then processed from the inbound queue to the application objects for updating. The processing of messages from an inbound queue requires the enablement of the JMS cron task when the sequential queue is used or the enablement of Message Driven Beans for the continuous queue. If errors occur when processing a file, you can manage and view the data import messages that are flagged with an error in the Message Reprocessing application.

Enabling an enterprise service:

You must enable an enterprise service that is associated to an external system before it can be used to receive inbound external application data. By default, the enterprise service records that are associated with an external system are disabled. A disabled enterprise service prevents the external system from processing inbound external application messages.

Procedure

1. In the External Systems application, display the system that you want to update.
2. On the Enterprise Services tab, specify whether you want the enterprise service to be enabled or disabled.

Option	**Enabled**
Enabled	Selected
Disabled	Cleared

3. Click **Save External System**.

Associating an enterprise service with an external system:

You can associate an enterprise service with an external system to synchronize inbound external application data with asset management objects. The services that you create and associate to the external system contain the inbound message processing logic.

About this task

You can specify whether the associated enterprise services receive data from the sequential or continuous queue. You also can enable the associated enterprise service when you are ready to perform inbound integration framework message processing.

Procedure

1. In the Enterprise Systems application, display the system that you want to update.
2. On the **Enterprise Services** tab, click **New Row**.
3. In the **Enterprise Service** field, enter a value.
4. Optional: Specify whether you want the enterprise service to be enabled or disabled:

Option	**Enabled**
Enabled	Selected
Disabled	Cleared

5. Specify whether you want the service messages to receive data from the continuous or sequential queue.

Option	Use Continuous Queue
Continuous queue	Selected
Sequential queue	Cleared

6. Click **Save External System**.

Selecting enterprise services for the external system:

You can associate multiple enterprise services with an external system to synchronize inbound external application data with asset management objects. The services that you create and associate to the external system contain the inbound message processing logic.

Procedure

1. In the External Systems application, display the system that you want to update.
2. On the **Enterprise Services** tab, click **Select Service**.
3. Choose one or more enterprise services by selecting the corresponding **Enterprise Service** check boxes.
4. Click **OK**.
5. Click **Save External System**.

What to do next

You must enable the associated enterprise services before you can perform integration framework message processing. You also can specify whether the associated enterprise services receive data from the sequential or continuous queues.

Selecting an enterprise service queue type:

You can specify whether the enterprise service that you associate to the external system receives data from the continuous or sequential queue. The queue selection you make for the enterprise service determines how the Java Message Service (JMS) queue processes inbound messages.

Before you begin

You must associate an enterprise service to an external system.

About this task

The continuous queue continues to process messages that are in the queue even when message processing results in an error. Conversely, the sequential queue stops processing messages that are in the queue until the processing error is cleared. A sequential queue also processes messages on a strict first-in-first-out basis.

Procedure

1. In the External Systems application, select the external system that you want to update.

2. On the Enterprise Services tab, select the enterprise service for which you want to select a queue.
3. Specify whether you want the service messages to receive data from the continuous or sequential queue.

Option	Use Continuous Queue
Continuous queue	Selected
Sequential queue	Cleared

4. Click **Save External System**.

What to do next

You can enable the associated enterprise service when you are ready to perform inbound integration framework message processing.

Exporting file-based data:

With the data export feature, you can perform a bulk export of message data from a file to an external system. You can initiate the export process for each publish channel that is associated to an external system.

Before you begin

In a multitenancy environment, you can use the data export feature only if the system provider provides you with access to a file server that is accessible by the application server. You must then configure a file-based endpoint to point to the location of this file server.

You must enable both the external system and the publish channel before you can export data. The data for export must either be in an XML file format that adheres to the object structure schema or in a delimited flat file, such as comma separated, that is a flattened version of the object structure schema format.

About this task

The optional SQL query that you enter in the **Export Condition** field, can affect the size of the exported XML message. You can filter the content to limit the amount of data that is being exported. The export process performs the standard outbound processing on the result set of the query for the selected publish channel.

Procedure

1. In the External Systems application, click the **Publish Channels** tab and select the publish channel that you want to export.
2. In the **End Point** field, specify a file-based endpoint handler, for either XML file or flat file format.
3. Click **Data Export**.
4. Optional: Enter a SQL query in the **Export Condition** field. The query must be against the primary or top-level object in the publish channel object structure.
5. Optional: Specify an integer value in the **Export Count** field to limit the number of records that are contained in the exported file. If the result of the query contains more records than the number you specify, those records are not included in the exported file.
6. Click **OK** to begin the data export process.

What to do next

When the data export is executed, the selected data is formed into a message and dropped into the outbound queue that is configured for the publish channel and its corresponding external system. The message is then processed from the outbound queue to the configured endpoint. If an error occurs when delivering a message to the endpoint, you can manage and view the data export messages that are flagged with an error in the Message Reprocessing application.

Adding queues to an external system:

You can use Java Message Service (JMS) queues to exchange enterprise service and publish channel data with an external application. When messages are received or sent, they are written to the JMS queue. These messages remain in the queues until they are successfully processed or deleted.

Before you begin

Before you can add a queue to an external system, you must create the queue on the application server. If you add a sequential queue, you must set up a cron task to periodically poll the queue for messages.

About this task

Each external system can have its own inbound and outbound queues, or you can configure multiple systems to share queues. You can either add your own user-defined message queues, or modify the existing queues when the predefined inbound and outbound message queues do not meet your needs.

Procedure

1. In the **External Systems** application, select the system for which you want to add a JMS queue.
2. Select the **Add/Modify Queues** action.
3. Click **New Row**.
4. Enter values in the following fields:

Option	Description
Queue JNDI Name	The name of the JMS queue.
Queue Connection Factory	The connection factory that is used for accessing the queue. The default value is `jms/mro/int/queues/sqin`.
Maximum Try Count	The number of times a message is processed before it is written to the error log and an e-mail notification is sent to the system administrator.

5. Optional: Enter values in the following fields:

Option	Description
Initial Context Factory	The class used to connect to the JMS server.
Provider URL	The URL of the JMS server.
User ID	The user ID that is used to access the new queue.

Option	Description
Password	The password that is use to access the queue.
E-mail Address	The e-mail address of a user who receives notices when transaction errors occur in the queue. This value is typically the e-mail address of a system administrator.

6. Optional: If the queue delivers inbound messages and functions as a continuous queue, clear the **Sequential** check box.
7. Optional: If the queue delivers outbound messages, clear the **Inbound** check box.
8. Click **OK**.
9. On the **System** tab, enter values in the following fields:
   - **Outbound Sequential Queue**
   - **Inbound Sequential Queue**
   - **Inbound Continuous Queue**
10. Click **Save External System**.

Creating interface tables:

You can create an interface table to integrate with external systems that use database tables for data exchange. Interface tables reflect the content of publish channel or enterprise service object structures. You must recreate existing tables when you change the definition of the corresponding object structure.

Before you begin

You cannot recreate an interface table when unprocessed messages for that interface table exist in the MXIN_INTER_TRANS queue table. If you do not back up data before you recreate an interface table, the system loses the data. Ensure that alias conflicts are resolved and the **Support Flat File Structure** check boxes are selected on the associated object structure records in the Object Structures application.

About this task

The Create Interface Table dialog box displays the interface tables that are associated with publish channels and enterprise services that have the following characteristics:

- Registered to the selected external system.
- Interface table name is not null.
- Corresponding object structure supports a flat file representation.

Procedure

1. In the External Systems application, select the **Create Interface Tables** action.
2. In the Create Interface Tables dialog box, select the interface table that you want update and create.
3. Enter a value in the **End Point** field.
4. Optional: Select the **Rename Existing** check box to create a copy of the selected interface table. The asset management system stores copy of the interface table and adds a BAK suffix to its name.

5. Click **Create**.
6. Click **OK** to create the table or **Cancel** to stop the process. Depending on the number of interface tables you are creating, this process can take some time.
7. Click **OK** to close the Create Interface Tables dialog box.

Working with integration controls:

If integration channels or services contain processing rules or exit classes that are based on integration controls, the values of the integration controls can be configured for the external system. You can configure boolean, list, and value integration controls that you can incorporate in processing rules.

Setting up a Boolean control:

You can set up a Boolean control on an external system when you need a control that specifies a value of true or false. You also can add specific organization-level and site-level values to a Boolean control. An enterprise service or a publish channel can use this Boolean control in its processing rule evaluations. The true or false value that you assign to the control in the external system determines whether an enterprise service or publish channel applies a processing rule.

Before you begin

The boolean value that you assign to a control must already exist in the control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, any organization-level or site-level values that you assign must exist in that domain.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the **Set Up Integration Controls** action.
3. Select the boolean control that you want to update.
4. Click **New Row**.
5. Perform one of the following actions:
   - Enter a value in the **Organization** field.
   - Enter a value in the **Site** field.
6. Specify whether you want the Boolean control to have a default value of true or false.

Option	Default True
True value	Selected
False value	Cleared

7. Click **OK** to close the Boolean Control dialog box.
8. Click **OK** to close the Set Up Integration Controls dialog box.

Setting up a cross-reference control:

You can set up a cross-reference control when you need a control that replaces one value with another. A cross-reference control can map a value in the asset management system to a value in the external system.

Before you begin

The value that you assign for translation must first exist in the control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, the organization-level or site-level values that you assign must exist in that domain.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the **Setup Integration Controls** action.
3. Select the cross-reference control that you want to update.
4. Click **New Row**.
5. In the **Maximo Value** field, enter the application value that the asset management system converts to or from an external system value.
6. In the **External Value** field, enter the external system value the asset management system converts to or from the **Maximo value**.
7. Click **OK** to close the Cross-Reference Control dialog box.
8. Click **OK** to close the Setup Integration Controls dialog box.

Setting up a list control:

You create a list type integration control when you need a control that contains a list of values. You also can add specific organization-level and site-level values to a list control. An enterprise service or publish channel can use this list control in its processing rule evaluations. The value that you assign to the control in the external system determines whether an enterprise service or publish channel applies a processing rule.

Before you begin

The values that you add to a list control must already exist in the control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, the organization-level or site-level values that you assign must exist in that domain.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the Set Up Integration Controls action.
3. Select the list control that you want to update.
4. Click **New Row**.
5. In the **Value** field, enter a value that exists in the domain.
6. Click **OK** to close the List Control dialog box.
7. Click **OK** to close the Set Up Integration Controls dialog box.

Setting up a value control:

You can set a value type integration control on an external system when you need a control that contains a single value. You also can add specific organization-level and site-level values to a value control. An enterprise service or a publish channel can use this value control in its processing rule evaluations. The value that you

assign to the control in the external system determines whether an enterprise service or publish channel applies a processing rule.

Before you begin

The value that you assign first must exist in the control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, the organization-level or site-level values that you assign must exist in that domain.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the **Set Up Integration Controls** action.
3. Select the value control that you want to update.
4. Click **New Row**.
5. Perform one of the following actions:
   - Enter a value in the **Organization** field.
   - Enter a value in the **Site** field.
6. In the **Value** field, enter a value that exists in the domain.
7. Click **OK** to close the Value Control dialog box.
8. Click **OK** to close the Set Up Integration Controls dialog box.

Overriding values for a cross-reference control:

You can set up a cross-reference control to override values for sites and organizations. You can choose to override the cross-reference values that were previously defined at the enterprise service or publish channel level. The value override can be configured according to each external system to maintain valid asset management system and external system mappings.

Before you begin

The value that you use for translation must first exist in the cross-reference control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, the organization-level or site-level values that you assign must exist in that domain.

About this task

If you use synonyms, enter the external value as the control value, not the internal application value.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the **Setup Integration Controls** action.
3. Select the cross-reference control that you want to update.
4. Click **Override**. The Override Values for Cross-Reference Control dialog boxh displays any organization-level and site-level values that exist for the control.
5. Click **New Row**.
6. Perform one of the following actions:

- Enter a value in the **Organization** field.
- Enter a value in the **Site** field.

7. In the Values for Organization/Site table window, click **New Row**.
8. Enter the values in the **Default Value** and **External Value** fields. To use the control as a multiplication control, enter multiple records that have the same external value and different organization or site values.
9. Click **OK** to close the Override Values for Cross-Reference Control dialog box.
10. Click **OK** to close the Cross-Reference Control dialog box .
11. Click **OK** to close the Set Up Integration Controls dialog box .

Example

Five external systems are configured to exchange data with the integration framework. Four of these external systems use the same site values, but one site value differs. A cross-reference control that performs the translation between the four mismatched values and the asset management system value can be overwritten at an external system level. The value override can be configured to translate the remaining mismatched external system site value to an asset management system storeroom value.

A cross-reference control in an enterprise service can translate the external system site value `EX001` to an asset management system site `MX001`. An override cross-reference control in external system can override the predefined `EX001` value and use an `EX002` value in its value translation.

Overriding values for a list control:

You can set up a list control to override values for sites and organizations. You can choose to override the list control values that were previously defined in the enterprise service or publish channel level. The rule could skip the processing of the transaction when the data field value does not match any of the overwritten list control values.

Before you begin

The value that you assign first must exist in the control definition in the Publish Channels or Enterprise Services application. If that definition points to a domain, the organization-level or site-level values that you assign must exist in that domain.

About this task

You must use a period (.) as the decimal placeholder when you enter decimals as a control value, regardless of the locale settings of the application server or database. Numbers to the left of the placeholder are not formatted. This format applies to inbound and outbound data. For example, $1,738,593.64 must be 1738593.64.

Procedure

1. In the External Systems application, display the system that you want to update.
2. Select the **Set Up Integration Controls** action.
3. Select the list control that you want to update.
4. Click **Override**. The Override Values for List Control dialog box displays any organization-level and site-level values that exist for the control.

5. Click **New Row**.
6. Perform one of the following actions:
   - Enter a value in the **Organization** field.
   - Enter a value in the **Site** field.
7. In the Values for Organization/Site table window, click **New Row**.
8. Enter a value in the **Value** field. If you use synonyms, enter an external value, not the internal application value.
9. Click **OK** to close the Override Values for List Control dialog box.
10. Click **OK** to close the Cross-Reference Control dialog box .
11. Click **OK** to close the Set Up Integration Controls dialog box .

Example

Work orders are sent to an external system based on their statuses. The processing rule that is defined on an enterprise service or publish channel can check the status of a work order against a list control that contains two status values: `APPR` (approved) or `COMPLETE`. This list override can be configured to evaluate two different work order status values: **WAPPR** (waiting on approval) or `WSCH` (waiting to be scheduled). If the status of a work order does not match the overwritten list control values, the work order transaction is not sent to the external system.

Predefined integration content

The integration framework provides predefined integration content, including object structures, publish channels and enterprise services that support importing data from external system or exporting data to them.

In some cases, only the object structure is provided without a related publish channel or enterprise service. The integration framework, by design, provides support for inserting, updating, deleting and querying data according the business rules defined in the business objects within the object structure. It does not provide all the functionality that is available through the applications, including those that are available from actions or the processing button on the screen. Some object structures may provide support for some actions or the processing button on the screen.

Master data objects

The integration framework provides a selection of predefined integration content for master data, which generally consists of accounting, people, storeroom, labor, classification, and vendor data.

Asset object:

The MXASSET object structure allows for bidirectional synchronization of asset information, including attributes that define meters.

Purpose

The MXASSET object structure synchronizes individual assets, but not the asset hierarchy as a whole. It supports the addition and update of meters tied to an asset, but not the update of meter reading values. Inbound processing of ASSETMETER is restricted to attributes that define the meter, not meter reading values or other information pertaining to meter readings.

Prerequisite

You must synchronize applicable operating locations, storerooms, meters, and items before loading assets.

The MBO relationship used to retrieve the ASSETMETER MBO is different from the MBO relationship used in the Assets application. Therefore, the changed attribute in the XML (outbound processing) is not set for any values from this MBO.

This object structure does not support status changes or asset movement. A standard service for assets is provided to support moving assets that reside in operating locations. The standard service, Asset, provides an operation, assetmoveSingleAsset, to support this functionality.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutAssetProcess - Populates the Hierarchypath field if a classification is associated with the asset.
Publish Channel	MXASSETInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxAssetProcess - If the Hierarchypath is provided, the processing class populates the corresponding classstructureid field of the asset, which creates the association of the classification with the asset.
Enterprise Service	MXASSETInterface
Processing Rules	None
Integration Controls	None

Chart of Account object:

The MXCOA object structure allows for inbound synchronization of chart of accounts data.

Prerequisite

You must load general ledger components before loading the chart of accounts. There is no predefined publish channel for this object structure.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	None
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxCOAProcess - Can process as a combination of delimited segments or as individual components that are part of the GLACCOUNT datatype definition.
Enterprise Service	MXCOAInterface
Processing Rules	None
Integration Controls	None

Examples of inbound processing

Processing as a combination of delimited segment is available for XML and interface tables. You can specify the general ledger account this way in all object structures.

```
<GLACCOUNT>
  <VALUE>6400-2-10</VALUE>
</GLACCOUNT>
```

The object structure validates each segment, then creates the chart of account record in database with the following values:

- GLACCOUNT=6400-2-10
- GLCOMP01-6400
- GLCOMP02=2
- GLCOMP03=10

You can also process the object as individual components that are part of the GLACCOUNT datatype definition.

```
<GLACCOUNT>
<GLCOMP glorder="0">6400</GLCOMP>
<GLCOMP glorder="1">2</GLCOMP>
<GLCOMP glorder="2">10</GLCOMP>
</GLACCOUNT>
```

The object structure validates the components and creates the account, using the delimiter defined in the GLCONFIGURE table for each segment. This option is available for XML only.

Classification item object:

The MXCLASSIFICATION object structure allows for bidirectional synchronization of classifications, including Class Use With, Class Specs, and Class Spec Use With.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXCLASSInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxClassificationProcess - Processing logic to find classification and parent using Hierarchypath. Because the system allows duplicate classifications (hierarchypaths), an exception is generated when attempting to update a classification if duplicates exist.
Enterprise Service	MXCLASSInterface
Processing Rules	None
Integration Controls	None

Craft object:

The MXCRAFT object structure allows for bidirectional synchronization of craft information. Craft information includes crafts, craft skills and craft rates.

Prerequisite

If the craft references a skill or contracts, that information must exist prior to loading crafts.

A craft can have a standard rate, a rate for each skill level for the craft, or different rates for each contract that provides the craft, with an optional skill-level rate for each contract.

The CRAFT and CRAFTSKILL records have a STANDARDRATE field for the hourly rate for each craft or skill associated with the craft.

The CRAFTRATE record specifies rates for vendors that supply the craft and, optionally, different rates for each combination of skill, craft, and vendor.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXCRAFTInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None
Enterprise Service	MXCRAFTInterface
Processing Rules	None
Integration Controls	None

Financial project object:

The MXPROJ object structure allows for bidirectional synchronization of financial project information. This object structure synchronizes individual tasks and projects as separate messages. It does not synchronize a project and all its child tasks in a single message.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXPROJInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None

Predefined integration components	Value and description
Enterprise Service	MXPROJInterface
Processing Rules	None
Integration Controls	None

General ledger (GL) component object:

The MXGLCOMP object structure allows for inbound synchronization of GL components.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	None
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None
Enterprise Service	MXGLCOMPInterface
Processing Rules	None
Integration Controls	None

Labor object:

The MXLABOR object structure allows for bidirectional synchronization of labor information, including person and labor craft rates.

Prerequisite

You must synchronize any associated craft, work location, or storeroom location before loading labor. Each person record can have only one labor record.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutLaborProcess - Processing logic disabled.
Publish Channel	MXLABORInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxLaborProcess - Supports status changes. Sets the ORGID for LaborCraftRate MBO from the parent MBO. Also sets the PERSONID on the PERSON record, based on the PERSONID in the LABOR record
Enterprise Service	MXLABORInterface
Processing Rules	None
Integration Controls	None

Person object:

The MXPERSON object structure allows for bidirectional synchronization of person information, including phone, email, and SMS data.

Purpose

This object structure supports synchronizing individual person data. It does not support updating availability information for a person record. The object structure also supports status changes.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXPERSONInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxPersonProcess - Requires that data for all child objects (PHONE, EMAIL, SMS) must be provided with every person update. Due to a lack of a unique key on these objects, the logic always deletes all of the child objects and adds them again. If a person has three email records in the system and a person message is received with two email records, the email record not included in the person message will be deleted. The object structure supports status changes.
Enterprise Service	MXPERSONInterface
Processing Rules	None
Integration Controls	None

Person/user object:

The MXPERUSER object structure allows for bidirectional synchronization of person/user information. This object structure supports status changes.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXPERUSERInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxPersonUserProcess - Processing is required to save the person data prior to saving of the user data. This object structure supports status changes.
Enterprise Service	MXPERSUSERnterface
Processing Rules	None
Integration Controls	None

Storeroom location object:

The MXSTORELOC object structure allows for bidirectional synchronization of storeroom, labor, and courier locations.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutLocProcess - provides filtering to send only storeroom, courier, or labor locations.
Publish Channel	MXSTORELOCInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxLocProcess - provides filtering to accept only storeroom, courier, or labor locations.
Enterprise Service	MXSTORELOCInterface
Processing Rules	None
Integration Controls	None

Vendor (Companies) master object Structure object:

The MXVENDORMSTR object structure allows for bidirectional synchronization of vendor master data.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXVENDORMSTRInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None
Enterprise Service	MXVENDORMSTRInterface

Predefined integration components	Value and description
Processing Rules	None
Integration Controls	None

Vendor (Companies) object:

The MXVENDOR object structure allows for bidirectional synchronization of organization-level vendor data, including contacts.

Prerequisite

To enable this object structure to create the company master record, select the **Automatically Add Companies to Company Master** option for the set associated with the organization to which the vendor record is being added. The MXVENDOR object structure supports status changes.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXVENDORInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxComProcess - Provides logic to support creating the company master record.
Enterprise Service	MXVENDORInterface
Processing Rules	None
Integration Controls	None

Item and inventory objects

The integration framework provides a selection of predefined integration content for item and inventory objects, such as service items, tool items, inventory vendors, and issues.

Item object:

The MXITEM object structure allows for bidirectional synchronization of item data, including conversions, item specs and conditions. This object structure supports status changes.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutItemProcess - Filter to not set tool or service items. Supports retrieval of the hierarchypath for the classstructureid of the item.
Publish Channel	MXITEMInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxItemProcess - If the inbound transaction is an item update and the capitalized flag of the item has changed, a method is called to change the capitalized status of the item. If the item type is not ITEM or a valid synonym. Supports status changes.
Enterprise Service	MXITEMInterface
Processing Rules	None
Integration Controls	None

Service item object:

The MXSERVITEM object structure allows for bidirectional synchronization of service item data. The main object of the object structure, SERVICEITEMS, is a nonpersistent object.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	None
Publish Channel	MXSERVITEMInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None
Enterprise Service	MXSERVITEMInterface
Processing Rules	None
Integration Controls	None

Tool item object:

The MXTOOLITEM object structure allows for bidirectional synchronization of tool item data, including tool item specs.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutToolItemProcess - Supports retrieval of the hierarchypath for the classstructureid of the item.
Publish Channel	MXTOOLITEMInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxToolItemProcess - If the inbound transaction is an item update and the capitalized flag of the item has changed, a method is called to change the capitalized status of the item.
Enterprise Service	MXTOOLITEMInterface
Processing Rules	None
Integration Controls	None

Inventory object:

The MXINVENTORY object structure allows for bidirectional synchronization of inventory (item-storeroom) data, including inventory costs.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutInvProcess - Sets the value of the ITEMTYPE field from the ITEM object.
Publish Channel	MXINVENTORYInterface
Processing Rules	SKIPINVENTORY - Skips the record if its ITEMTYPE is in the SKIPITEMTYPE integration control.
Integration Controls	SKIPITEMTYPE

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxInvProcess - Supports status changes.
Enterprise Service	MXINVENTORYInterface
Processing Rules	None
Integration Controls	None

Inventory balance object:

The MXINVBAL object structure allows for bidirectional synchronization of inventory balance data. The balance change information in this object structure is for the lowest level (BIN or LOT) within the application.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutInvBalancesProcess - Sets the value of the ITEMTYPE field from the ITEM object.
Publish Channel	MXINVBALInterface
Processing Rules	SKIPINVBALITM - Skips the record if its ITEMTYPE is in the SKIPITEMTYPE integration control.
Integration Controls	SKIPITEMTYPE

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxInvBalancesProcess - The inbound inventory balance object structure internally calls the current balance adjustment method, and this creates a CURBALADJ financial transaction in INVTRANS.
Enterprise Service	MXINVBALInterface MXINVBALQInterface - A second enterprise service that is configured for operation query.
Processing Rules	None
Integration Controls	None

Item vendor object:

The MXINVVENDOR object structure allows for bidirectional synchronization of vendor-item data.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutInvVendorProcess -Sets the value of the ITEMTYPE field from the ITEM object and set the value of the CURRENCYCODE from the vendor (companies) object.
Publish Channel	MXINVVENDORInterface
Processing Rules	INVVITEMTYPE - Skips the record if its ITEMTYPE is in the ITEMTYPEFORINV integration control.
Integration Controls	ITEMTYPEFORINV

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxInvVendorProcess - If the inbound message contains a currency code and that code differs from the vendor's currency code, an error is reported.
Enterprise Service	MXINVENDORInterface
Processing Rules	None
Integration Controls	None

Inventory reservations object:

The MXINVRES object structure provides for the bidirectional synchronization of inventory (storeroom) reservations. This object structure does not process direct issue reservation records created by the system.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutRSVProcess - Skips reservations that are for direct issues.
Publish Channel	MXINVRESInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	None
Enterprise Service	MXINVRESInterface
Processing Rules	None
Integration Controls	None

Inventory issues object:

The MXINVISSUE object structure provides for the bidirectional synchronization of inventory (storeroom) issues and returns.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	MoutISUProcess - Skips reservations that are for direct issues.
Publish Channel	MXINVISSUEInterface - Filters out direct issues from MATUSETRANS (PONUM is not null and ISSUETYPE is ISSUE or RETURN) because they are handled by the receipts (MXRECEIPTInterface) object structure. Filters out variance transactions that are written to MATUSETRANS by the invoice approval process.
Processing Rules	None

Predefined integration components	Value and description
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	MaxISUProcess - If the action provided is not an Add action, then an exception is generated. Validates ISSUETYPE to ensure that it is ISSUE, RETURN, or null. If it is any other value, an error is reported. If it is null, the value is set to ISSUE by default.
Enterprise Service	MXINVISSUEInterface
Processing Rules	None
Integration Controls	None

Documents objects

The integration framework provides a selection of predefined integration content for documents, such as purchasing records, invoices, and work order records.

The STATUSIFACE field and processing:

All purchasing and work order documents have a status. The STATUSIFACE field identifies whether related transactions contain new or updated records, or status changes only.

In general, the field STATUSIFACE applies to all object structure subrecords that are created from stateful MBOs, which are MBOs that have a STATUS field and support status change actions. The STATUSIFACE field is provided as a nonpersistent field to those objects where integration supports a status change.

The STATUSIFACE field and outbound processing

While processing an object structure subrecord that has been created from a stateful MBO, the outbound integration processing looks for a STATUS field in the MBO and a STATUSIFACE field in the corresponding object structure subrecord. If both fields exist, the processing sets the value of the STATUSIFACE field to the value of the changed attribute for the corresponding STATUS field. A value of 0 indicates that the status was not changed. A value of 1 indicates that the status was changed.

This processing applies only to event-generated outbound messages, not to messages exported via the data export feature or through a programmatic invocation.

The STATUSIFACE field and inbound processing

When processing an object structure where the primary (top) MBO is stateful, the inbound processing looks for a STATUSIFACE field on the corresponding object

structure subrecord to determine if the inbound message is to be processed as both a document update and a status change, or as a status change only.

The following table describes inbound processing using object structures with a stateful primary (top) MBO. It does not apply to any stateful MBO that is included as a child object in an object structure.

Table 2. Actions taken during inbound processing of the STATUSIFACE field

Value of STATUSIFACE	Document exists in the database	Document does not exist in the database
Not provided, or 0	• Updates the document in the database • If the status of the inbound document is different from the status in the database, updates the status in the database	• Adds the document to the database • Sets the status in the database to the status of the inbound message
1	Updates the status in the database	Error

Purchase contracts object:

The MXPC object structure allows for bidirectional synchronization of purchase contract information. This object structure currently only supports the purchase contracts types of blanket and price.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 3. Purchase contracts object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MoutPCProcess - Skips reservations that are for direct issues.
Publish channel	MXPCInterface - Uses processing rules so that the complete purchase contract document is sent only for a status change.
Processing rules	1. SKIPPC - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the PCSEND control. 2. SKIPPCUPDATE -Skips the sending of the document when updated but no status change. 3. SETSTATUSIFACE - Sets the value of the STATUSIFACE field to 0 (false) for all status values except those listed in the PCSEND control. 4. CHECKSTATUS - Sends only the purchase contract header data if status changes and the new status is not in the PCSEND control. (In this case, the full document was previously sent and this is sending only the status change notification).
Integration controls	PCSEND

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 4. Purchase contracts object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxPCProcess - Supports status changes. Creates a new revision when the purchase contract already exists and the revision does not. Only supports contract types of blanket and price.
Publish channel	MXPCInterface
Processing rules	None
Integration controls	None

Purchase requisitions object:

The MXPR object structure allows for bidirectional synchronization of purchase requisition information and supports status changes.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 5. Purchase requisitions object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXPRInterface – Uses the processing rules. The complete purchase request document is sent only for status changes.
Processing rRules	1. SKIPPR - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the PRSEND control. 2. SKIPPRUPDATE - Skips the sending of the document when updated but the status is not changed. 3. SETSTATUSIFACE - Sets the value of the STATUSIFACE field to 0 (false) for all status values except those listed in the PRSEND control. 4. CHECKSTATUS - Sends only the purchase requisition header data if status changes and the new status is not in the PRSEND control. (In this case, the full document was previously sent and this is sending only the status change notification).
Integration controls	PRSEND

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 6. Purchase requisitions object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxPRProcess – Supports status changes. If the current status of the purchase requisition is approved, to support updates the processing class will change the status back to waiting for approval, perform updates and then return the status to what is identified in the XML message.
Publish channel	MXPRInterface
Processing rules	None
Integration controls	None

Purchase order object:

The MXPO object structure allows for bidirectional synchronization of purchase order information and supports status changes.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 7. Purchase order object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXPOInterface - Uses processing rules, the complete purchase order document is only sent only for status changes.
Processing rules	1. SKIPPO - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the POSEND control. 2. SKIPPOUPDATE - Skips the sending of the document when updated but the status is not changed. 3. SETSTATUSIFACE - Sets the value of the STATUSIFACE field to 0 (false) for all status values except those listed in the POSEND control. 4. CHECKSTATUS - Sends only the purchase order header data if status changes and the new status is not in the POSEND control. (In this case the full document was previously sent and this is sending only the status change notification).
Integration controls	POSEND

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 8. Purchase order object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxPOProcess - supports status changes. If the current status of the purchase order is approved, to support updates, the processing class changes the status back to waiting for approval, performs the updates, and then returns the status to what is identified in the XML message.
Publish channel	MXPOInterface
Processing rules	None
Integration controls	None

Invoice object:

The MXINVOICE object structure allows for bidirectional synchronization of invoice information and supports status changes.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 9. Invoice object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXINVOICEInterface - Uses processing rules, the complete invoice document is sent only for status changes.
Processing rules	1. SKIPINVOICE - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the IVSEND control. 2. SKIPINVIOCEUPDATE - Skips the sending of the document when updated but the status is not changed. 3. SETSTATUSIFACE - Sets the value of the STATUSIFACE field to 0 (false) for all status values except those listed in the IVSEND control. 4. CHECKSTATUS – Sends only the invoice header data if the status changes and the new status is not in the IVSEND control. (In this case the full document was previously sent and this is sending only the status change notification).
Integration controls	IVSEND

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 10. Invoice object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	The inbound processing creates and updates INVOICECOST lines only if one of the following conditions is met: • The invoice line does not reference a purchase order line. • The corresponding purchase order line does not have a distribution. • The purchase order line does not reference a storeroom. In all other cases, the inbound processing ignores INVOICECOST information when creating or updating the invoice. If the inbound message provides INVOICETERMS, the inbound processing deletes the old terms and adds the new ones. Users can optionally specify a price variance in the PRICEVAR field on each invoice line, if invoice matching is done in the external system and price variances determined in the external system are sent to create variance transactions to update work orders and storerooms. In order to process these variances, a value must exist for OWNERSYSID in the inbound invoice, and it cannot be the same as the value of MAXVARS.MXSYSID. Users can optionally specify a price variance in the PRICEVAR field on each invoice line, if invoice matching is done in the external system and price variances determined in the external system are sent to create variance transactions to update work orders and storerooms. In order to process these variances, a value must exist for OWNERSYSID in the inbound invoice, and it cannot be the same as the value of MAXVARS.MXSYSID. The IVMATCH collaboration switch requires the following values: • OWNER1SYSID value is always “THISMX”. • OWNER2SYSID value is the value of INVOICE.OWNERSYSID. If the evaluation is false, line level invoice variances are ignored in the invoice processing. The following are the typical scenarios for using the invoice interface: • Invoice Matching in the system (AP Outbound): Invoices received from third parties, either electronically or manually, are processed by the system; that is, they are matched against receipts, if applicable, and then approved. Alternately, payment schedules created in the system will result in approved invoices being created based on the schedule. These approved invoices result in a payment advice being sent to an external AP system. The process of matching also results in accounting entries being posted to the general ledger. • Invoice Matching in external system (Variances Inbound): The system does not do the invoicing, but accepts matched invoices from external systems and applies any variances back to the respective work orders, storerooms, and so on. The accounting entries related to the accounts payable and/or variances must be recorded in the external system; they will not be sent out. The primary intent of this interface is to provide the system with any variance information necessary for updating the work order costs.
Publish channel	MXINVOICEInterface

Table 10. Invoice object predefined components for inbound integration processing (continued)

Predefined integration components	Value and description
Processing rules	None
Integration controls	None

The following table shows an example of possible INVOICE.OWNERSYSID values, the evaluation that is generated, and the default result of the evaluation.

Table 11. Evaluation example

Value of INVOICE.OWNERSYSID	Evaluation	Result (default)
Null	OWNER1SYSID="THISMX" and OWNER2SYSID="THISMX"	False
MXSYSID	OWNER1SYSID="THISMX" and OWNER2SYSID="THISMX"	False
EXTSYSID	OWNER1SYSID="THISMX" and OWNER2SYSID="EXT"	True
Any other value	OWNER1SYSID="THISMX" and OWNER2SYSID="EXT"	True

Outbound processing rules for work order interfaces:

Unlike purchasing document interfaces, work order document interfaces do not have a STATUSIFACE field and do not send status change notifications. A work order document interface is first sent out when the status of a work order changes to WOSTART and on every subsequent update, regardless of the status of the work order.

Users can configure the start value in the WOSTART control. Any status that you specify in the control is the value, not the MAXVALUE. If multiple synonym values exist for a status, list all applicable synonyms.

Work order object:

The MXWO object structure allows for bidirectional synchronization of work order information and supports status changes.

Purpose

The system first sends out a work order when it reaches the status in the WOSTART control, then on all updates thereafter. The entire work order document is always sent.

The work order interface contains all the information defined by system on the work order, but it does not provide additional information about projects or financial control data, equipment, and locations that is not part of the standard work order. If necessary, you can add the additional information via user fields.

The system treats work order tasks as work orders. Both have similar properties and they are stored in the same table. If a user creates a work order, adds tasks/child work orders to the work order, and then approves the work order, the **Inherit Status Changes** flag on the work order indicates whether approval of the work order also results in approval of all the tasks/child work orders of that

particular work order. The default is Y, so when a work order is approved, any child tasks or work orders that inherit the parent's approval based on this flag are also approved.

The outbound event listener on the work order MBO receives multiple independent events, one for each work order, and they are processed and sent out independently. Therefore, a work order with three tasks and two child work orders results in six independent outbound work orders.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 12. Work order object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXWOInterface - Uses processing rules, the complete work order document is sent only for status changes.
Processing rules	SKIPWO – Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the WOSTART control.
Integration controls	WOSTART

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 13. Work order object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	StatefulMicSetIn - Generic class that supports status changes.
Publish channel	MXWOInterface
Processing rules	None
Integration controls	None

Work order detail object:

The MXWODETAIL object structure allows for bidirectional synchronization of work order information, including planned material, labor, service, and tools.

Purpose

The system first sends out a work order when it reaches the status in the WOSTART control, then on all updates thereafter. The entire work order document is always sent.

The MBO relationship that is used to retrieve the reservation MBO (INVRESERVE) is different from the relationship that is used in the Work Order Tracking application. Therefore, the changed attribute in the XML is not set for any values from this MBO

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 14. Work order detail object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	MoutWORsvProcess - Provides the code required to update the sendersysid column on the invreserve MBO.
Publish channel	MXWODETAILInterface - Uses processing rules, the complete work order document with plan information is sent only for status changes.
Processing rules	SKIPWO - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the WOSTART control.
Integration controls	WOSTART

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 15. Work order detail object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	None
Publish channel	None
Processing rules	None
Integration controls	None

Work order hierarchy object:

The MXWOHIER object structure allows for bidirectional synchronization of work order information, including child work orders, and also supports status changes. The system first sends out a work order when it reaches the status in the WOSTART control, then on all updates thereafter. The entire work order document is always sent.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 16. Work order hierarchy object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXWOHierInterface - Contains an event filter class, WOHierarchyEventFilter, that prevents the child MBOs from being sent out when they are being sent out via a parent MBO (where both parent and child MBOs are initiated by the same event action).
Processing rules	SKIPWO - Skips the sending of the document if both of the following conditions are true: • The document has not been sent out before. • The new status is not listed in the WOSTART control.
Integration controls	WOSTART

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 17. Work order hierarchy object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxWOHierarchyProcess - Supports status changes and commit processing to support loading multiple related work orders in a single message.
Publish channel	MXWOHierInterface
Processing rules	None
Integration controls	None

Transaction interface objects

The integration framework provides a selection of predefined integration content for transaction interface objects, such as receipts, general ledger transactions, labor time reporting, and meter readings.

Receipts object for materials and services:

The MXRECEIPT object structure allows for bidirectional synchronization of purchase order receipt information for material and service receipts and also supports status changes. The object structure uses a nonpersistent object to support both material (MATRECTRANS) and service (SERVRECTRANS) receipts.

Purpose

In the outbound direction, this object structure processes purchase order receipts, transfers (movements against receipts or receipts against internal POs), and returns (returns to vendors after receipt inspection, or returns to vendor after acceptance and goods movement from the inspection holding location).

For receipts that require inspection, the user-defined field INSPECTED indicates if the receipt line was inspected in the external system. The interface does not process transfers independently; each transfer is associated with a receipt.

You do not have to specify whether an inbound receipt is a material receipt or a service receipt. The integration processing uses the POLINE to make the determination.

All quantities, including return quantities, must be positive.

The two types of transfer records are identifiable by the following values in the RECEIPTREFID field:

- Null: a receipt against an internal purchase order
- Not null: movement against a receipt

Do not specify a RECEIPTFREF value for returns. Returns are processed independently of the corresponding receipt.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 18. Receipts object predefined components for outbound integration processing

Predefined integration components	**Value and description**
Object structure definition class functionality	MoutProcess - Processes receipts with issue type of RECEIPT or RETURN. If no issue type is specified, it is treated as RECEIPT. Processing for RECEIPT issue type and Inspection Required = N: Material and service receipts: • Uses inbound RECEIPTQUANTITY (for material receipts) or QTYTORECEIVE (for service receipts) to create the receipt. • Maps inbound REJECTEDQTY to the REJECTEDQTY field in the receipt. • Ignores any other quantities. • Does not look at the inbound INSPECTED field. Material receipts (MATRECTRANS records): • Sends out only RECEIPT and RETURN type records and TRANSFER type records containing a PONUM (not a storeroom transfer). Sends out new receipts only, not updates to existing receipts. Service receipts (SERVRECTRANS records): • Sends out RECEIPT and RETURN type records. Sends out new receipts and updates to existing receipts.
Publish channel	MXRECEIPTInterface
Processing rules	Service receipts: Sends out records when the status is equal to a value in the SERVRECSTAT control (default COMP). This occurs under the following conditions: • A record that does not require inspection is inserted. • An existing record is updated in database and the status field is changed to COMP.
Integration controls	SERVRECSTAT control identifies all statuses at which the system will send out service receipt transactions. It can have multiple values. By default, the value is COMP.

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 19. Receipts object predefined components for inbound integration processing

Predefined integration components	**Value and description**
Object structure processing class functionality	MaxRCVProcess - Processing for RECEIPT issue type and Inspection Required = Y: Material receipts: INSPECTED = N: • Uses only the inbound RECEIPTQUANTITY field to create the receipt; ignores accepted and rejected quantity values. • Creates a receipt with STATUS = WINSP (waiting inspection) and quantity derived from RECEIPTQUANTITY. INSPECTED = Y: • Uses the inbound RECEIPTQUANTITY, ACCEPTEDQTY, and REJECTEDQTY fields to create the receipt. • Does not allow partial inspections or acceptances. RECEIPTQUANTITY must equal ACCEPTEDQTY + REJECTEDQTY. • Creates a receipt with STATUS = WASSET (if rotating item) or COMP (all other items). • Depending on quantities specified, can create up to three transactions—one RECEIPT, one TRANSFER, and one RETURN. Service receipts: INSPECTED = N: • Uses the inbound AMTTORECEIVE (if POLINE order quantity is null) or QTYTORECEIVE (in other cases) to create a receipt; ignores all other quantity values. • Creates a receipt with STATUS = WINSP (waiting inspection) and quantity derived from QTYTORECEIVE. INSPECTED = Y: • Uses the inbound QTYTORECEIVE, ACCEPTEDQTY, and REJECTEDQTY fields to create a receipt. • Does not allow partial inspections or acceptances. QTYTORECEIVE must equal ACCEPTEDQTY + REJECTEDQTY. • Creates a single transaction of type RECEIPT, with STATUS = COMP. Processing for RETURN issue type: Material and service receipts: • Accepts return transactions for a POLINE only if there was an earlier receipt for the same line; if Inspection Required = Y for the POLINE, the receipt must have been approved. Otherwise, reports an error. • Uses only the inbound RECEIPTQUANTITY (for material receipts) or QTYTORECEIVE (for service receipts) field to create the receipt; ignores all other quantity values. • Creates a single transaction with issue type RETURN and the credit GL account as the RBNI account.
Publish channel	MXRECEIPTInterface
Processing rules	None
Integration controls	None

Material and rotating item receipt object:

The MXRCVROTITM object structure allows for inbound synchronization of receipt information for items, including rotating items.

Purpose

This object structure supports inbound processing only.

This interface does not let you specify a status for the receipt; the status is always assumed to be COMP.

This interface differs from the MXRECEIPTInterface in that it processes material receipts exclusively and lets you identify serialized rotating assets to be created in the case of rotating item receipts.

You can receive rotating items with or without asset numbers. If you receive them without asset numbers, you must manually specify the asset numbers by using the Receive Rotating Items dialog box in the Purchasing Receiving application.

For rotating items, the number of inbound transactions is one more than the number of rotating items. One transaction exists for the total receipt quantity, and one transaction exists for each rotating item associated with the receipt. For example, for a receipt of ten rotating items produces eleven transactions.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 20. Material and rotating item receipt object predefined components for outbound integration processing

Predefined integration components	**Value and description**
Object structure definition class functionality	None
Publish channel	None
Processing rules	None
Integration controls	None

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 21. Material and rotating item receipt object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxRcvRotItmMProcess - Reports an error if the purchase order line being processed has LINETYPE = SERVICE or STDSERVICE. Processing is the same as the inbound integration point processing class functionality for material receipts using the MXRECEIPTInterface, with the following additional processing: • Receipts: If the line item is a rotating item and Inspection Required = N, or INSPECTED = Y, the processing checks for asset information that corresponds to the item provided in the interface. If the information is available, the processing validates the number of asset records to ensure it equals the RECEIPTQTY or ACCETPTEDQTY (whichever applies). If it does, invokes the receiving functionality and creates asset as required. • Returns: If the item being returned is a rotating item, ignores any asset information. In the system, the return of a rotating type item does not affect the assets created by the original receipt. The asset records remain unchanged; only the item balances are updated (if applicable).
Publish channel	MXRCVROTITMInterface
Processing rules	None
Integration controls	None

General ledger (GL) object:

The MXGLTXN object structure allows for outbound synchronization of GL transactions. This interface allows for the posting of site-level transactions to an external general ledger application for accounting reconciliation.

Purpose

This interface uses a nonpersistent MBO with data from the following subrecords:
- SERVRECTRANS
- MATRECTRANS
- INVTRANS
- INVOICETRANS
- MATUSETRANS
- LABTRANS
- TOOLSTRANS

The event filter class, GLEventFilter, is used to set the events to the applicable persistent objects listed.

The SOURECEMBO field identifies the database table in which the transaction originated. Its value is derived from the GLSOURCEMBO synonym domain.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 22. General ledger object predefined components for outbound integration processing

Predefined integration components	Value	Description
Object structure definition class functionality	MoutGLProcess - Service receipts	Service receipts include the following entries: • Accounting entries for non-distributed service receipts created by the receiving application for services ordered on purchase orders (ISSUETYPE = RECEIPT, COSTINFO=1) • Accounting entries for distributed service receipts created by the receiving application for services ordered on purchase orders (ISSUETYPE = POCOST) • Accounting entries for purchase order services with Receipt Required = N that are directly invoiced instead of being received (ISSUETYPE = INVOICE) • Accounting entries for services that are not against purchase orders and are invoiced directly (ISSUETYPE = INVOICE) • Invoice variance transactions recorded against service receipts (ISSUETYPE = INVOICE) Service receipts (SERVRECTRANS) processing: • Sets SOURCEMBO to SERVRECTRANS and ISSUETYPE to the value listed. • INVOICE type transactions: sends out on insert. • POCOST and RECEIPT type transactions: if Inspection Required = N for the corresponding POLINE, sends out on insert. If inspection required = Y, sends out when status is changed to COMP.
	MoutGLProcess - Material receipts	Material receipts include the following: • Accounting entries for non-distributed material receipts created by the receiving application for items/ tools ordered on purchase orders (ISSUETYPE = RECEIPT, COSTINFO = 1) • Accounting entries for distributed material receipts created by the receiving application for items/ tools ordered on purchase orders (ISSUETYPE = POCOST) • Accounting entries for Item transfers between storerooms (ISSUETYPE = TRANSFER and PONUM=NULL) • Accounting entries for receipt inspection transfers of items between the receipt inspection storeroom and the purchase order line storeroom (ISSUETYPE = TRANSFER and RECEIPTREF!=NULL and PONUM!=NULL) • Accounting entries for receipts against internal purchase orders (ISSUETYPE = TRANSFER and RECEIPTREF=NULL and PONUM!=NULL) • Accounting entries for receipt inspection goods return of items and materials (ISSUETYPE = RETURN) • Accounting entries for return to vendor from a storeroom or direct issue purchase order lines (ISSUETYPE = RETURN) • Invoice variance transactions recorded against material receipts (ISSUETYPE = INVOICE) • Accounting transaction for increasing the kit item's INVENTORY control account value when kits are made (ISSUETYPE = KITMAKE). transactions for increasing the INVENTORY control account for each constituent item of a kit when a kit is disassembled (ISSUETYPE = KITBREAK) Material receipts (MATRECTRANS) processing: • Sets SOURCEMBO to MATRECTRANS and ISSUETYPE to value listed. • Sends out INVOICE, RECEIPT, TRANSFER, RETURN, KITMAKE and KITBREAK type transactions on insert. • For POCOST type transactions on insert if status is COMP (that is, on insert if Inspection Required = N, and when status changed to COMP if Inspection Required = Y).

Table 22. General ledger object predefined components for outbound integration processing (continued)

Predefined integration components	Value	Description
	MoutGLProcess - Inventory adjustment transactions	Inventory adjustment transactions include the following: • Inventory current balance adjustments (ITTYPE = CURBALADJ) • Inventory standard/ average cost adjustments (ITTYPE = STDCOSTADJ/ AVGCOSTADJ) • Cost difference when a kit is disassembled and there is a difference between the value of the kit and the sum of the kit component values (ITTTYPE = KITCOSTVAR) • Physical count reconciliation (ITTYPE = RECBALADJ) • Capitalized cost adjustment (ITTYPE = CAPCSTADJ) • Standard cost receipt adjustment (ITTYPE = STDRECADJ) Inventory adjustment (INVTRANS) processing: • Sets SOURCEMBO to INVTRANS and ITTYPE to value listed. • Does not send out transactions with ITTTYPE = INSERTITEM, CREATEASSET and PHYSCNT, as they are considered audit records rather than accounting transactions. • Sends out all other transactions on insert.
	MoutGLProcess - Invoice transactions	Invoice transactions include the following transactions created by invoice approval: • Invoice TOTAL transaction (amount payable to invoice vendor, TRANSTYPE = TOTAL) • Invoice line tax transactions (tax accounting for each TAX code for an invoice line, TRANSTYPE = TAX1:TAX5) • Invoice currency variance transaction (TRANSTYPE = CURVAR) • Invoice price variance transactions (TRANSTYPE = INVCEVAR) Invoice transaction (INVTRANS) processing: • Sets SOURCEMBO to INVOICETRANS and TRANSTYPE to the value listed. • Sends out all transactions when they are created.
	MoutGLProcess - Material issue and return transactions	Material issue and return transactions include the following: • Accounting entries for items issued from a storeroom in the system (ISSUETYPE = ISSUE) • Accounting entries for items returned to a storeroom (ISSUETYPE = RETURN) Material issue and return (MATUSETRANS) processing: • Sets SOURCEMBO to MATUSETRANS and ISSUETYPE to value listed. • Does not send out direct issue transactions that are created by PO receiving/ invoice variances in MATUSETRANS, as they are accounted for in MATRECTRANS. Identifies such transactions by their PONUM, so all MATUSETRANS transactions that have a PO reference are not sent out by this interface. • Sends out all other transaction at time of creation.
	MoutGLProcess - Labor transactions	Labor Transaction (LABRTRANS) processing: • Sets SOURCEMBO to LABTRANS. • If approval not required, send out labor actuals against work orders on insert. • If approval required, sends out transaction after it is approved (when GENAPPSERVRECEIPT is Y).
	MoutGLProcess - Tool transactions	Tool Transaction (TOOLTRANS) processing: • Sets SOURCEMBO to TOOLTRANS. • Sends out usage of tools on work orders on creation of TOOLTRANS
Publish channel	MXGLTXNInterface	
Processing rules	SKIPGL	Skips the sending of the general ledger transaction based on the value in SOURCEMBO existing in the GLSOURCE control.
Integration controls	GLSOURCE	

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 23. General ledger object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	None
Publish channel	None
Processing rules	None
Integration controls	None

Labor time reporting object:

The MXEMPACT object structure allows for posting site-level actual time reported in the system to external applications. The approved labor actuals are sent out and all inbound labor actuals are accepted regardless of status. The status of existing records are not updated when processing inbound transactions.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 24. Labor time reporting object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	MXEMPACTInterface
Processing rules	SKIPEMPACT - Skips the sending of the labor pay (LABTRANS) if the transaction is not approved (GENAPPRSERVRECEIPT = 0).
Integration controls	

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 25. Labor time reporting object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxEmpactProcess - Allows add actions only. A null action is processed as an add action. LABTRANSID is not allowed to be passed with a value because the LABTRANSID is program-generated during the add action.
Publish channel	MXEMPACTInterface
Processing rules	None
Integration controls	None

Meter reading object:

The MXMETERDATA object structure allows inbound synchronization of meter reading data. This object structure supports only inbound processing. This interface does not support the processing of meter readings for work order tasks.

Outbound integration processing

The following table shows the predefined values used in outbound integration processing.

Table 26. Meter reading object predefined components for outbound integration processing

Predefined integration components	Value and description
Object structure definition class functionality	None
Publish channel	None
Processing rules	None
Integration controls	None

Inbound integration processing

The following table shows the preconfigured values used in inbound integration processing.

Table 27. Meter reading object predefined components for inbound integration processing

Predefined integration components	Value and description
Object structure processing class functionality	MaxMeterDataProcess - The processing class verifies that the following attributes are provided: • Site ID • Asset or location • Meter ID or condition monitoring point • Meter reading value, reading date time, inspector When a work order is specified, the following processing occurs: • If a measurement point is specified, the processing class identifies the asset or location meter corresponding to the site and measurement point, and creates a meter reading for that asset or location meter. • If a meter is specified, then either asset or location must be specified. If both are specified, the processing class assumes the meter belongs to the asset, and processes the meter accordingly. If only the asset or location is specified, then the meter on the corresponding entity is updated. If no work order is specified, the following processing occurs: • If a measurement point is specified, the reading is recorded for the measurement point. • If a meter is specified, then either asset or location must be specified. If both are specified, the processing class assumes the meter belongs to the asset, and processes the meter accordingly. If only asset or location is specified, then the meter on the corresponding entity is updated.

Table 27. Meter reading object predefined components for inbound integration processing (continued)

Predefined integration components	Value and description
Publish channel	MXMETERInterface
Processing rules	None
Integration controls	None

System objects

System objects are MBOs that are generally used for application or metadata configuration. Updating data using system objects may require specific post-processing activities, for example, reconfiguring the database.

The following restrictions apply to the use of enterprise services, publish channels, and object structures:

- You cannot enable listeners for publish channels.
- You cannot process system interfaces via interface tables or flat files.
- You must specify an action code on inbound system interfaces.

Object structures object:

The MXINTOBJECT object structure allows for inbound synchronization of the definition of object structures, including column aliases. System validations that apply to users adding, deleting and modifying predefined object structures apply to the modification of object structures via the Object Structure object structure service.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Enterprise service object:

The MXENTSRV object structure allows for inbound synchronization of the definition of enterprise services and their corresponding processing rules and control values.

Prerequisite

Before enterprise service creation, make sure all controls used by the enterprise service exist.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Publish channel object:

The MXIFACEOUT object structure allows for inbound synchronization of the definition of publish channels their corresponding processing rules and control values. Before publish channel creation, make sure all controls used by the publish channel exist.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

End point object:

The MXENDPOINT object structure allows for inbound synchronization of the definition of end points.

Prerequisite

Before end point creation, make sure handlers used by end points exist.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

External system object:

The MXEXTSYSTEM object structure allows for inbound synchronization of the definition of an external system, the enterprise services and publish channels used by the external system, and their corresponding control values.

Prerequisite

Before external system creation, make sure all enterprise services, publish channels, end points, and controls used by the external system exist.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Integration control object:

The MXIFACECONTROL object structure allows for inbound synchronization of Integration Controls and their system level default values.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Invocation channel object:

The MXIFACEINVOKE object structure allows for inbound synchronization of an invocation channel definition.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Integration queue object:

The MXQUEUE object structure allows for inbound synchronization of integration queue definitions.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Message definition object:

The MXMESSAGE object structure allows for inbound synchronization of system error and warning messages.

Prerequisite

In addition to the new messages added via this object structure, the `messages.xml` file must be updated with the corresponding message text.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None

Predefined integration components	Value and description
Integration Controls	None

MBO configuration object:

The MXOBJECTCFG object structure allows for inbound synchronization of MBO definition information.

Prerequisite

Use this object structure with caution.

Use only add, addchange, or change action codes when synchronizing inbound data through this object structure. Do not use the replace action unless you completely replace the MAXOBJECTCFG and MAXATTRIBUTECFG data.

The delete action presents the risk of deleting predefined records in the database tables associated with MAXOBJECTCFG and MAXATTRIBUTECFG MBOs.

An inbound setting restriction exists on the CHANGED, EAUDITENABLED, EAUDITFILTER, EAUDITTBNAME, IMPORTED, STORAGEPARTITION, MAXOBJECTID column, so the XML value is not set to the MBO.

After synchronizing inbound data with this interface, for your changes to take effect, use the Database Configuration application and select the **Apply Configuration Changes** action.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	psdi.iface.app.configure.MaxObjcfgProcess - If the event that MAXOBJECTCFG is creating a view (MAXOBJECTCFG.VIEW=1), the processing class skips any MAXATTRIBUTECFG MBOs associated with the referred MAXOBJECTCFG. This class also delays the SAVE validation on MAXOBJECTCFG until all its associated attributes are successfully added into the database.
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Domain object:

The MXDOMAIN object structure allows for inbound synchronization of domain information.

Purpose

Some fields in the database are associated with select value lists. These lists of defined values are known as domains (sometimes referred to as value lists).

This object structure synchronizes ALN, numeric, numeric range, table, and crossover domains definitions.

Prerequisite

Update and delete operations require external systems to provide valid DOMAINID values.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Communication template object:

The MXCTEMPLATE object structure allows for bidirectional synchronization of communication templates that users can leverage to standardize frequently used e-mail communications and notifications.

Purpose

A communication template is a definition of a mail message with subject, message, and recipient information that is processed when certain nodes become current, or along specified workflow routing paths between nodes.

This object structure synchronizes application, change status, custom class, command line, and set value actions, but not action group.

Prerequisite

Update and delete operations require external systems to provide a valid ACTIONID value.

This object structure does not support the creation of action groups.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	MXCTEMPLATEInterface
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None

Predefined integration components	Value and description
Integration Controls	None

Action definition object:

The MXACTION object structure allows for inbound synchronization of workflow action definitions that can be used with escalation, service level agreement (SLA), and workflow processes.

Purpose

An action is an event that you want the system to trigger when it encounters records that meet the conditions defined by an escalation point, service level agreement, or workflow process.

This object structure synchronizes application, change status, custom class, command line, and set value actions, but not action group.

Prerequisite

Update and delete operations require that external systems provide a valid ACTIONID value.

This object structure does not support the creation of action groups.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

System properties object:

The MXPROP object structure allows for inbound synchronization of system properties and values.

Purpose

Prerequisite

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Integration module object:

The MXIM object structure allows for inbound synchronization of an integration module definition including properties and LMO relationships.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Logical management operations (LMO) object:

The MXLMO2 object structure allows for inbound synchronization of LMOs including attributes.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Operational management product (OMP) object:

The MXOMP object structure allows for inbound synchronization of OMPs, including relationships to CIs and configurations with IM and LMOs.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Launch entry object:

The MXLAUNCH object structure allows for inbound synchronization of Launch Entries and context information.

Outbound Integration Processing

The following table shows the predefined values used in outbound integration processing.

Predefined integration components	Value and description
Object Structure definition class functionality	N/A
Publish Channel	N/A
Processing Rules	None
Integration Controls	None

Inbound Integration Processing

The following table shows the preconfigured values used in inbound integration processing.

Predefined integration components	Value and description
Object Structure processing class functionality	N/A
Enterprise Service	N/A
Processing Rules	None
Integration Controls	None

Data loading order

When you use the integration framework to load multiple sets of data into the application database, you must maintain the dependencies so that the data is loaded correctly. Use the following sequence of predefined object structures as a guide to determine the order in which you load the data.

Object structures

MXGLCOMP
MXCOA
MXVENDORMSTR
MXVENDOR
MXPERSON
MXPERUSER
MXCRAFT
MXLABOR
MXSTORELOC
MXCLASSIFICATION
MXITEM
MXINVENTORY
MXINVBAL
MXINVVENDOR
MXSERVITEM
MXPROJ
MXASSET
MXWO
MXWODETAIL
MXWOHIER
MXINVRES
MXEMPACT
MXINVISSUE
MXGLTXN
MXTOOLITEM
MXPC
MXMETERDATA
MXPR
MXPO
MXRECEIPT
MXRCVROTITM
MXINVOICE

System object structures

MXINTOBJECT
MXIFACECONTROL
MXENTSRV
MXIFACEOUT
MXENDPOINT

MXQUEUE
MXEXTSYSTEM
MXIFACEINVOKE
MXIM
MXLMO2
MXOMP
MXLAUNCH
MXMESSAGE
MXOBJECTCFG
MXDOMAIN
MXCTEMPLATE
MXACTION
MXPROP

Integration data processing

You can configure integration components in different ways to meet your integration requirements. You can integrate with multiple external applications, and each application requires a different integration approach that is based on the integration support that is provided by that application.

Planning to process data for integration

Before you initiate an integration transaction, you must decide which integration components to use. You must also plan how to implement these components so that you can achieve a successful integration with the external destination for your integration.

The choice of the components that you use and how you implement them is often determined by the external application that you are integrating with. These choices include the following considerations:

Data type
Define the data, for example work orders and person data. When defined, determine whether there are predefined object structures to support each set of data or must you create more object structures.

Direction
For each set of data, which direction will the integration scenario implement, sending outbound transactions, receiving inbound transactions, or both? The direction of the integration can vary for each set of data.

Message exchange
This factor depends on the capabilities of the external application and options can include integrating data by using files (XML or flat), calling web services, and posting XML over HTTP. The method of exchanging data and how often that occurs can vary for each set of data.

Customization
As data is exchanged, is customization within the application needed to transform message content or to apply integration business rules? If customization is required, does it use integration framework capabilities such as Java exit classes, automation scripts, XSL maps, or processing rules, or is customization implemented in the external application?

Depending upon the answers to these questions, you need different types of component and API configurations, which can include:

- Object structures
- Publish channels and invocation channels
- Enterprise services
- Object structure services, enterprise services, and standard services
- Web services
- REST and OSLC APIs
- External systems
- Endpoints

Inbound data processing

The integration framework supports asynchronous and synchronous processing of inbound integration messages with the following types of service; object structures, enterprise services, or standard services.

For asynchronous processing, the external system establishes and maintains a connection until the message is passed into a JMS queue. For synchronous processing, the external system establishes maintains a connection until message processing is complete.

Asynchronous processing of inbound messages

During asynchronous processing of inbound messages, an external application calls an enterprise service and maintains a connection until the message is saved to a JMS queue and the connection ends. If an error occurs during the save of the message to the queue, the external application is responsible for reprocessing the message.

Asynchronous processing is managed by sequential or continuous JMS queues. Processing in a sequential queue guarantees the order of delivery of messages, based on a first in, first out order. Processing in a continuous queue delivers messages in any order.

You can use enterprise services to support the following transactions:

- Importing data from a flat file or an XML file
- Loading data from interface tables
- Receiving an XML message from an HTTP POST request
- Calling a web service that is configured to use a JMS queue
- Calling an EJB
- Using a direct JMS connection

All of these options support the use of an enterprise service, when the integration message is passed into a JMS queue. When the message is in the queue, a separate process that uses a cron task or a message-driven bean (MDB) picks up the message from the queue. The message is then processed through the enterprise service layer, the related object structure layer, and the business objects are updated. Because messages are processed through a JMS queue, no response is sent to the application that called the service and the Query operation is not supported in this processing model.

Synchronous processing of inbound messages

You can implement synchronous inbound integration with an object structure, enterprise, or standard service. These transactions require the establishment and maintenance of a direct connection from the external application to the integration framework for the duration of the transaction.

The external application maintains the connection to the service until business objects are updated in the Maximo database or until business objects are returned in response to a Query operation. The integration framework returns a response to the external application to confirm the success or failure of message processing.

The following options are available for sending synchronous integration messages to services:

- Posting of an XML message by using HTTP
- Starting a web service that is configured to bypass the JMS queue
- Starting an EJB

Object structure and enterprise services can use any of these options to support Create, Update, Delete, Sync, and Query operations.

You can also access integration object structures, standard services, and application business objects by using the REST API or the OSLC REST API.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Initiation of asynchronous processing of inbound data

To initiate integration processing, the external system uses one of the supported methods to establish a connection. After the external system and enterprise service are validated, messages are placed in the JMS queue that is specified for the enterprise service.

The procedure for establishing the initial connection varies depending on the integration method that is used for the transaction. The following table describes the available methods and the connections that these methods use.

Method	Connection procedure
HTTP or HTTPS Post method	Use the following URL: *hostname*:*port*/meaweb/esqueue/*extsysname*/*entservname* where: • *extsysname* is the name of the external system • *entservname* is the name of the enterprise service

Method	Connection procedure
EJB	Start an EJB from a Java Platform, Enterprise Edition client by using the following code: `public byte{} processExternalDataAsync(byte[] extData, String serviceName, String sender)` On the client, specify the JNDI name of the Enterprise JavaBean, `ejb/maximo/remote/enterpriseservice`, to look up the Enterprise JavaBean reference and to start the method. The following configurations are required on the Java Platform, Enterprise Edition client: • Access to the `Home` and `Remote` class files • Access to the Java Platform, Enterprise Edition JAR files for the server • The URL of the server that hosts the Enterprise JavaBeans • The class name of the context factory The client code must instantiate the default `IntialContext` object. The context derives the provider URL and the context factory from the environment.
Data import feature that uses XML or flat files	The user clicks **Data Import** in an application.
Data import cron task	The XMLFILECONSUMER cron task or the FLATFILECONSUMER cron task polls the source directory that is configured for data import.
Interface tables	The external system writes message data to the appropriate interface tables and updates the MXIN_INTER_TRANS queue table with information about the sequence for processing the records in the interface table. A cron task polls the MXIN_INTER_TRANS queue table for records to be processed.
Web services	The external system starts an enterprise service that is deployed as a web service that is configured not to bypass the JMS queue.
JMS direct	The external system passes an XML message directly into the JMS queue from another queuing system, such as WebSphere MQ.

When a connection is established, or when a cron task runs and identifies that data is ready for import, the following steps occur:

1. The integration framework checks that the external system and the enterprise service are valid and enabled.
2. If the messages are in flat file format, the integration framework checks that the object structure supports flat structures.
3. If verification fails, the integration framework notifies the sender of the error and does not process the data.
4. If the verification is successful, the integration framework identifies the inbound JMS queue that is assigned to the enterprise service and writes the message to the queue.
5. The integration framework updates the JMS message header with the names of the external system and the enterprise service.

If a transaction that contains multiple instances of a record encounters errors, error-handling varies depending on the exchange method used. The following error-handling occurs, depending on the exchange type:

- For HTTP or EJB transactions only: If a transaction contains multiple instances of a document, for example, if it contains 10 person records, a single message is written to the JMS queue, not 10 individual messages. If one of the records has a processing error, a total message processing exception occurs and none of the records is committed to the database.
- For data import transactions only: If a transaction contains multiple instances of a document, for example, if it contains 10 person records, the application writes 10 individual messages to the JMS queue. If one of the messages has a processing error, an error transaction is logged for that message and the remaining 9 messages continue to process in the application.

Initiation of synchronous processing of inbound data

To initiate synchronous processing, the external system uses one of the supported methods to establish a connection. During the connection process, the integration framework checks that the external system and the referenced enterprise service are valid and enabled.

The procedure for establishing the initial connection varies, depending on the service that is used for the transaction. The following table describes the available methods and the connections that these methods use.

Method	Connection procedure
HTTP or HTTPS Post method	• Use the following URL for transactions that use an enterprise service: *hostname*:*port*/meaweb/es/*extsysname*/*entservname* Where: – *extsysname* is the name of the external system – *entservname* is the name of the enterprise service • Use the following URL for transactions that use an object structure service: *hostname*:*port*/meaweb/os/*osname* Where *osname* is the name of the object structure service. • Use the following URL for transactions that use a standard service: *hostname*:*port*/meaweb/ss/*application service name* Where *application service name* is the name of the standard service.

Method	Connection procedure

EJB	• For enterprise service transactions, start an EJB from a Java Platform, Enterprise Edition client by using the following code:
	`public byte{} processExternalDataSync(byte[] extData, String serviceName, String sender)`
	The client can use the JNDI name of the Enterprise JavaBean, `ejb/maximo/remote/enterpriseservice`, to look up the Enterprise JavaBean reference and to start the method. The following configurations are required on the Java Platform, Enterprise Edition client:
	• For object structure service transactions, start an EJB from a Java Platform, Enterprise Edition client by using the following code:
	`public byte{} processMOS(byte[] reqmosData, String mosName)`
	The client can use the JNDI name of the Enterprise JavaBean, `ejb/maximo/remote/mosservice`, to look up the Enterprise JavaBeans reference and to start the method. The following configurations are required on the Java Platform, Enterprise Edition client:
	• For object structure service transactions, start an EJB from a Java Platform, Enterprise Edition client by using the following code:
	`public byte{} action(byte[] actionData, String maxServiceName)`
	The client can use the JNDI name of the Enterprise JavaBean, `ejb/maximo/remote/actionservice`, to look up the Enterprise JavaBean reference and to start the method. The following configurations are required on the Java Platform, Enterprise Edition client:
	The following configurations are required on the Java Platform, Enterprise Edition client:
	• Access to the `Home` and `Remote` class files
	• Access to the Java Platform, Enterprise Edition JAR files for the server
	• The URL of the server that hosts the Enterprise JavaBeans
	• The class name of the context factory
	The client code must instantiate the default IntialContext object. The context derives the provider URL and the context factory from the environment.

Web service	Use the following URL for transactions that start a web service: `http://hostname:port/meaweb/services/web service name` where *web service name* is the name of a deployed web service.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Processing sequences

The same processing sequence is applied to asynchronous and synchronous messages that use an enterprise service. The processing sequences are different for synchronous messages that use object structure or standard services.

Enterprise service processing sequences:

An enterprise service message processes through a number of layers to query and update business object data. You can use these processing layers to customize and control processing to satisfy the integration requirements with the external application.

For asynchronous processing, the data can originate from the following exchange methods:

- Interface tables
- HTTP Post of an XML message
- Starting a web service that is configured to use a JMS queue
- Starting an EJB
- Direct JMS connection
- Data import of a flat file or an XML file

For synchronous processing, the data can originate from the following exchange methods:

- HTTP Post of an XML message (HTTP)
- Starting a web service that is configured to use a JMS queue
- Starting an EJB

The same processing sequence applies for asynchronous and synchronous processing.

Enterprise service processing occurs in the following sequence:

1. The external system calls the enterprise service.
2. Custom processing can occur at several points during the processing of enterprise services. You can run custom Java classes or automation scripts to manipulate the message data or you can apply an XSL map. Custom processing occurs in the following order:
   a. If a preprocessing method is defined in a user exit class, this method is applied first.
   b. If an external exit class is specified, this processing is applied.
   c. If a postprocessing method is defined in a user exit class, this method is applied.
   d. If an XSL map is specified for the enterprise service, the map is applied to convert enterprise service format to object structure format.

   At this point in the processing, the message must match the format of the object structure that is associated with the enterprise service.
3. Object structure processing occurs in the following order:
   a. You can run custom Java classes or automation scripts to manipulate data during object structure processing.
   b. If processing rules are configured for the object structure, these rules are run next.
   c. If processing rules result in the multiplication of data for sites or organizations, the object structure message is duplicated for the additional sites or organizations.
   d. Objects are created and object processing starts.
4. Object processing occurs in the following order:

a. If a preprocessing method is defined in a user exit class, this method is applied.
b. If an external exit class is specified, this processing is applied.
c. If a postprocessing method is defined in a user exit class, this method is applied.
d. The business objects are processed, and data is saved to the database.

5. For synchronous transactions only, for Create, Update, Delete, and Sync operations, the enterprise service provides a response to the starting application that includes the following information:
   - Indicates whether the message was processed successfully.
   - The value of the internal ID column of the main object, regardless of whether the column is part of the primary key of the main object.
   - The values of the primary key fields of the main object.
   - The values of any alternate key fields for the main object, which is configured for the object or for the object structure.
6. For synchronous transactions only, for Query operations, the entire object structure is returned. You can run custom Java classes or automation scripts to manipulate the response data, or you can apply an XSL map.

Synchronous integration with an object structure service:

An object structure service is accessible by using HTTP Post, EJB, and web service invocation methods to receive synchronous object structure service messages. The operations Create, Update, Delete, Sync, and Query are supported.

Integration processing that is based on an object structure service occurs in the following sequence:

1. The external system uses one of the following invocation methods to initiate communication with the integration framework:
   - HTTP invocation
   - EJB invocation
   - Web service invocation
2. The integration framework receives the inbound message that identifies the object structure that is associated with the message
3. The integration framework builds the objects, based on the object structure and the content of the inbound message.
4. The integration framework applies any predefined logic to the object structure.
5. Objects are processed and data is committed to the database if the message contains an update operation, or the data is returned to the requester if the message contains a query operation.
6. If a processing error occurs, the response with an error message is returned to the caller of the service.

Synchronous integration with a standard service:

A standard service is accessible by using HTTP Post, EJB, and web service invocation methods to receive synchronous standard service messages.

Integration processing that is based on a standard service occurs in the following sequence:

1. The external system uses one of the following invocation methods to initiate communication with the integration framework:
   - HTTP invocation
   - EJB invocation
   - Web service invocation
2. The objects are passed to the system, and standard system processing is applied.

Outbound data processing

Publish channels provide asynchronous processing of outbound messages that do not require a response from the external system. For outbound messages that require a response from the external system, you can use an invocation channel to provide synchronous processing.

Asynchronous integration with a publish channel

The integration framework uses publish channels to process asynchronous outbound messages. The sending of a publish channel message can be initiated by an object event, or you can use the data export feature to send publish channel messages on demand.

Publish channels are configured for an object structure and can enable event-based integration when the main object of the object structure is updated. An update occurs, for example, when a user saves a record in an application or completes a stage in a workflow process. When the main object of an object structure is updated, an event is initiated to all publish channels that are associated with the object structure and are configured to listen for events. Users can use the data export feature in the External Systems application to initiate publish channel messages at any time.

When a publish channel message is initiated by an event or by data export, the integration framework performs a sequence of tasks:

1. Identifies the object structure and the component objects that are associated with the publish channel.
2. Constructs the object structure for the transaction and forms an XML message that is based on the content of the objects.
3. Applies any processing rules that are defined in the Publish Channels application.
4. If the publish channel is associated with multiple external systems, creates copies of the object structure, one for each publish channel and external system combination.
5. If any custom processing Java classes, automation scripts, or an XSL map are associated with the publish channel, processes them in the following order:
   a. Runs the event filter class, if one is provided.
   b. Runs the preprocessing method in the user exit processing class, if one is provided.
   c. Runs a publish channel processing class, if one is provided.
   d. ,
   e. Runs the postprocessing method in the user exit class, if one is provided.
   f. Applies an XSL map, if one is provided.
6. Writes the XML message to the outbound queue specified for the external system.

7. A cron task picks up the publish channel message and delivers it to the external system, which is based on the endpoint that is configured.

Publish channel processing overview:

The object structure that is configured for a publish channel provides the message content of the channel. The external system determines how and where the integration message is delivered. Within the publish channel layer there are optional processing layers that support the transformation of message content and the application of business rules.

Event-based integration:

A publish channel can initiate an integration message in response to the processing of a system transaction, such as the update of an object.

You can configure publish channels to listen to events that occur on the primary object of the related object structure by selecting the **Enable Listener** check box in the Publish Channels application. When the main object of an object structure is updated, a publish channel message sends the updated information to the external system.

When an update to a child object initiates an update to the main object, this information is also sent to the external system. However, some updates to a child object do not initiate an update to the main object and, as a result, the updated information is not sent to the external system. For example, if an object structure is configured to support an attachment as a child object, when the user adds an attachment, the external system is not notified of this event.

To propagate an event from a child object to the primary object of the object structure, in the Object Structures application,you use the **Advanced Configurations** action to configure event propagation. When you configure an object structure to propagate events on child objects to the main object, the main object then sends the updated information to the external system.

If you configure event propagation for an object structure, you must monitor message activity carefully, because the configuration can initiate more messages than the external system is designed to process. For child objects that have embedded logic to update the parent object, this configuration has no impact in terms of enabling or disabling the event propagation from the child to the parent object.

Data export feature:

The publish channel supports batch processing of integration data by using the data export feature.

You initiate the data export feature by running a SQL query on the primary object of an object structure, and the number of records to send. If the query is successful, the export starts.

You initiate the data export feature by providing a SQL WHERE clause on the primary object of an object structure in the Data Export window. You can also configure the number of records to send. If the query is successful, the export starts and delivers the data based on the endpoint that is configured for the publish channel.

Synchronous integration with an invocation channel

The integration framework uses invocation channels to process synchronous outbound messages that require a confirmation or response content from the external application. The export is started by an action class that calls the invocation channel.

Sending an invocation channel message is initiated by an action Java class that is coded to start the channel. Invocation channels are used, for example, to call an external application to validate data, to start a process on an external application, or to retrieve data. An invocation channel can be triggered by a user interface control that is configured to call the action class that starts the channel.

When an invocation channel message is started by an action class, the integration framework performs a sequence of tasks:

1. Identifies the object structure that is associated with each invocation channel and identifies the component objects.
2. Constructs the object structure for the transaction and forms an XML message that is based on the content of the objects.
3. If any custom processing Java classes, automation scripts, or an XSL map are associated with the invocation channel request, processes them in the following order:
   a. Runs the preprocessing method in the request user exit processing class, if one is provided.
   b. Runs an invocation channel request processing class, if one is provided.
   c. Runs the postprocessing method in the request user exit class, if one is provided.
   d. Applies an XSL map, if one is provided.
4. Writes the XML message to the destination that is specified by the endpoint.
5. The endpoint handler specifies the transport mechanism to use.
6. When the response is received from the end point, if any custom Java classes, automation scripts, or XSL maps are associated with the response, processes them in the following order:
   a. Runs the preprocessing method in the response user exit processing class, if one is provided.
   b. Runs an invocation channel response processing class, if one is provided.
   c. Runs the postprocessing method in the response user exit class, if one is provided.
   d. Applies an XSL map, if one is provided.
7. Objects are built from the data in the response object structure and are passed to the system where standard processing is applied.
8. If all the objects that are created from the response object structure process successfully, they are committed to the database.

The response content can be managed by the invoker based on integration requirements, for example to update business object data or display the response to a user. A default action class is provided that you can use or extend to implement an invocation channel integration.

Customization of metadata properties in an invocation channel:

Within the external exit classes or the user exit classes of the invocation channel, you can update additional data, such as the override values for the parameters that are configured for an endpoint.

You can use this type of customization, for example, to derive an endpoint parameter, such as the URL, from the data in the object structure of the invocation channel. You can override the values for a number of metadata properties.

Value	Description
ENDPOINT	You can override this value to use a provided endpoint in place of the endpoint that is configured through the External System.
ENDPOINTPROPS	This value points to a hash map that can contain one or more endpoint parameters with a corresponding override value. If the endpoint configuration defines that you cannot override a property any override value in the integration context is ignored.
TARGETOBJECT	This value is the target object structure of the invocation channel.
SOURCEOBJECT	This value is the source object structure of the invocation channel.

Configuring integration processing

Processing of integration messages occurs at several points during the movement of integration messages for inbound and outbound transactions. When you configure integration processing, you must ensure that source data and destination data are in compatible data structures and formats.

Configuring asynchronous processing of inbound messages by using enterprise services

Enterprise services use asynchronous processing for inbound messages that do not require a response from Maximo Asset Management. To enable enterprise service processing, you configure several integration components, including an object structure, enterprise service, external system, and JMS cron task.

About this task

An external system initiates a connection and sends a message that includes the names of the external system and the enterprise service. The external system maintains the connection while the integration framework checks that the enterprise service is enabled for this transaction. Valid transactions are placed into the inbound sequential JMS queue, and the connection with the external system ends. A cron task polls the JMS queue, forwards the message for enterprise service processing, and sends the message onwards for object structure processing before it is committed to the database.

Procedure

1. In the Object Structures application, identify a predefined object structure to use or create one.

2. In the Enterprise Services application, identify a predefined enterprise service to use or create one.
3. Configure the enterprise service to use your object structure and specify a valid operation. The Query operation is not valid for asynchronous processing with enterprise services.
4. In the External Systems application, identify an external system to use or create one and configure it for your enterprise service:
   a. Configure the external system to use inbound JMS queues and enable the external system.
   b. Configure the external system to use your enterprise service and enable the service.
5. Optional: Configure the source for the integration message.

Message source	Configurations
Messages from files by using a cron task	In the Cron Task Setup application, set the data import cron task to active.
Messages from a web service call	In the Web Services Library application, create a web service that is based on your enterprise service.
Messages from interface tables	In the Cron Task Setup application, set the interface table cron task to active.

6. Configure a queue to receive the integration messages.

Queue type	Description
Continuous	On the application server, configure message-driven beans to consume queue messages.
Sequential	In the Cron Task Setup application, set the JMS cron task to active so that it can consume messages from the queue.

Configuring asynchronous processing of outbound messages by using publish channels

Publish channels process outbound integration messages that do not require a response from the message destination. To enable publish channel processing, you configure several integration components, including an object structure, publish channel, external system, and cron task.

About this task

An object structure provides the message content to the publish channel. Publish channel processing transforms the message, and the external system then forwards it to its destination queue. A cron task polls the queue on a regular schedule and delivers the message to its destination. Publish channel transactions can be initiated by an event, such as a change to the status of an object, or by starting a data export transaction in the External Systems application.

Procedure

1. In the Object Structures application, identify a predefined object structure to use or create one.
2. In the Publish Channels application, identify a predefined publish channel to use or create one.
3. Optional: To enable event-based integration, configure a listener for the event:

a. In the Publish Channels application, select the Enable Publish Channel Listener action. If the listener is enabled, when the primary object of the object structure is updated, publish channel transactions are initiated.
b. If you want events on child objects to start publish channel transactions, in the Object Structures application, select the **Advanced Configurations** action and configure event propagation.

4. Optional: In the Publish Channels application, review the processing rules and add or change rules as necessary. Processing rules are run in the order indicated in the **Sequence** field. If you change the sequence values, rules run in a different order, which can have unintended results.
5. Optional: Specify any automation scripts , custom Java class files, or XSL maps to provide custom logic during the transformation process.
6. In the External Systems application, add the publish channel to the external system and specify the filepath to the outbound sequential JMS queue.
7. In the Cron Task Setup application, enable the SEQOUT instance of the JMSQSEQCONSUMER cron task. The cron task delivers messages from the JMS queue to their destinations in the same order as the messages are received into the queue.

Rule-based customization

You can use processing rules to change the behavior of predefined integration processing without writing Java classes, automation scripts, or XSL maps. You can implement processing rules only on publish channels or on enterprise services.

Processing rules can access and evaluate the values in XML and object fields, object sets, and integration and system controls. Processing rules also can change the values in XML and object fields, or stop or skip processing all or part of a message.

Rule definitions for objects and records

An object structure consists of one or more object records. When an object is created, the object fields are populated from the corresponding record fields before standard application processing is applied.

During outbound processing, the original object is populated with the record fields from the corresponding fields. Except for certain generic integration fields, system objects are not updated in outbound messages.

Use the following guidelines to apply a rule to an object structure record or to an object:

- For outbound processing, you can apply processing rules to object structure records only.
- For inbound processing, you can apply processing rules to object structure records or to objects.
  - If an inbound rule changes the key field value of an object, apply rules to the object structure record.
  - If an inbound rule does not evaluate or manipulate an object or object set, apply rules to the object structure record.
  - If an inbound rule evaluates or manipulates a user-defined field, apply rules to the object structure record.
  - If an inbound rule evaluates or manipulates an object or object field, apply rules to the object.

Apply all rules for enterprise services to either objects or to object structure records. Avoid applying rules to both objects and object structure records. If you apply processing rules to both object structures and objects, the processing time for inbound transactions increases.

Processing rule definitions

A processing rule performs an action on a field in a record or object, or on the record or object itself. To define a processing rule, in the service or channel used for the transaction, you can specify the record or object to which the rule applies.

Processing rule initiation:

For an outbound transaction, a business object event on the primary object of an object structure initiates a processing rule. For an inbound transaction, the event that initiates a processing rule is identified by the value of the Action attribute on the primary object of the inbound XML message.

When you configure publish channels and enterprise services, you specify which events to use to initiate each processing rule. You can apply a processing rule to the primary record, a child record, or an object, but the event that initiates the rule must be initiated on the primary object.

You can, for example, implement a Stop rule on the PERSON object that prevents users from changing any attribute on the person record by identifying that the rule applies on update. With this configuration, users can create or delete person records but users cannot update person records.

For outbound transactions, in the Publish Channels application you can configure one or more of the following actions on the primary object to initiate the rule:

- Insert
- Delete
- Update

When an outbound message is generated using the Data Export feature or by a programmatic call to the publish channel, all enabled processing rules are run, regardless of the event settings.

For inbound transactions, in the Enterprise Services application you can configure one or more of the following actions on the primary object to initiate the rule:

- Add
- Change or Replace (equivalent of Update)
- Delete

You can set one of these values as the Action attribute of the primary object for the inbound transaction.

Processing rule actions:

A processing rule can act on an enterprise service or a publish channel as a whole. For example, a rule can bypass a message, or it can manipulate the value in a data field within the message.

Three processing rule actions act on a service or channel message: stop, skip, and skip children. Four processing rule actions transform the value in a field within a service or channel message: combine, split, set, and replace.

Message processing actions:

Message processing actions stop or skip an entire enterprise service or publish channel message, or skip entire records within the message.

Skip action

The skip action bypasses a message that meets specified criteria. When a skip action is applied to an inbound message, the message is not processed and is cleared from the inbound queue. When a skip action is applied to an outbound message, nothing is written to the queue and the message is not sent to an external system. Skip processing does not generate an error, but the system log file is updated with the rule that caused the skip action.

A skip action has some predefined rules. These rules look up integration control values to ensure that outbound messages have a valid status before being sent to the external system.

Stop action

The stop action halts the processing of a message that meets specified criteria. An outbound message is rolled back and an error message is displayed. For inbound transactions, the message remains in the inbound JMS queue. If the error was the result of a synchronous invocation of the enterprise service, the calling program is notified about the error.

Predefined rules are not provided with a stop action. This option is a utility for users to customize the behavior of a service or channel.

When possible, use the skip action rather than the stop action for inbound enterprise services. The stop action results in a processing error and the message remains in the inbound queue or the initiator receives an error response. These results do not occur when you use the skip action.

If a processing rule with a stop action is applied to a publish channel that is generated by the Data Export feature, the stop action is treated as a skip action. If the stop action evaluates as true, the message is skipped.

Skip children action

You can apply the skip children action only to outbound messages in a publish channel. Apply the processing rule on the record or object whose child level records are skipped.

If the person structure has the person object and child objects of phone and email, you can use the skip children action on a person to strip the phone and email data from the message. Use the skip children action when a status change occurs and the external system does not need the accompanying phone and email information.

Skip record action

You can apply the skip record action only to outbound messages in a publish channel. The skip record action deletes a record or object that contains your applied rule. Apply the processing rule on the record or object, and all of child records and objects, that you want to skip.

If the person object structure has a person object and a child object of phone, you can use the skip record action to strip a specific phone record from the message. Use the skip record action, for example, when you want to send the work phone number but not the home phone number for a person. The skip record rule needs a condition that identifies a home phone record to ensure that the record is skipped.

Field transformation actions:

Apply the field transformation rule to a record or object that contains the field to be transformed. A field transformation rule can be applied to a single field or multiple fields in the selected record.

Set action

The set action writes a value to a specified data field. When you define the rule, you specify the data to be set and the source of the new value. Indicate whether the rule always writes the new value to the target field or writes the new value only when the field is null (the default action). You can use this action to initialize the value in a data field. If the rule always writes the new value to the target field, any existing value in the field is overwritten.

The source can be one of the following values:

- A value integration control
- A hard-coded value
- A system control (in the MAXVARS database table)
- Another field in the specified record or object
- A field in a related object

Replace action

The replace action replaces a value in a data field with another value. When you define the rule, specify the data field that you want to update. The control that you use must be a cross-reference control. You specify the name of a cross-reference control that contains the original and replacement values for the data field.

Use this action when the database and the external system use different identifiers for the same entity. You can, for example, replace the SITEID value in a publish channel with an external PLANTID value, and replace the external PLANTID value in an enterprise service with the SITEID value.

Combine action

The combine action concatenates values from multiple source fields into a single target field. When you define the rule, identify the target field and the source fields and the sequence in which the source data is to be written. The source data can be a data field or an integration control that contains a data value. You can also specify an integration control that contains the delimiter to separate the segments in the target field.

Use this action in an enterprise service processing rule when a mismatch exists between the system definition and the external system definition of an entity. An enterprise service processing rule can, for example, combine a vendor ID and a vendor location field from an external system into the COMPANY field. A publish

channel processing rule can then use the split action to separate the combined field into separate values when data is sent to the external system.

The source and target fields must be in the same object. This action always overwrites the existing value in the target field. Ensure that the source and target fields are alphanumeric fields, or processing errors may occur.

Split action

The split action is the reverse of the combine action. The split action separates the value in one field into multiple fields. When you define the rule, you identify one source field, one or more target fields, and how the rule processor identifies segments of the source field.

The fields can have the following sources:

- A field in the selected record or object
- An integration control that contains the delimiter that separates the segments in the source field

The source and target fields must exist in the same object. This action always overwrites the existing value in the target fields. Ensure that the source and target fields are alphanumeric fields, or processing errors might occur.

If you combined multiple fields in an inbound message, split the combined field into individual fields in the outbound direction. There are two ways to identify how to split the field. You can specify the length of each segment of the source field, or you can identify a delimiter that separates the segments.

If the field length of each segment of source data is constant, the rule processor breaks up the source field from left to right, based on the field length, sequence, and values that you specify. For example, target field A with a character length of 6 holds positions 1-6 of the source field. Target field B with a character length of 3 holds positions 7-9 of the source field.

If the length of the source field segments is variable, but the source field contains a distinct delimiter that identifies the segments, use the separator option. The separator option identifies an integration control that defines the separator. The same separator must delimit all the segments. The rule processor parses the source field from left to right. The processor looks for the delimiter, breaks up the string into multiple values, and moves each value into the designated target field.

Processing sequence:

Processing rules are applied sequentially for each record or object within an object structure, starting with the primary object and moving down to the child objects.

If you define multiple processing rules for a single record or object, you can modify the default processing sequence. Your modification is especially important if a rule depends on the successful result of an earlier rule. If a rule with a stop or skip action is successfully applied, no further checking occurs.

Conditions and evaluations

Processing rules are applied conditionally. Any conditions must be met before the processing or action that is specified in the rule can be performed.

Conditions can involve evaluating or comparing XML field data, an object field, an object set, or an integration or system control.

Condition specifications:

A condition is a grouping of one or more evaluations. Multiple conditions can be specified, and their sequence is identified by the condition number.

Each evaluation returns a value of true or false. If an evaluation checks whether the values of two fields are equal, for example, it returns a value of true if the fields are equal and a value of false if they are not equal. Conditions also return a value of true or false. If every evaluation within a condition is true, the condition is true. If any evaluation within the condition is false, the condition is false. If a processing rule contains multiple conditions, only one condition must be true for the action that is associated with the processing rule.

Evaluation category specifications:

Before you define the specifics of an evaluation, select the type of data that must be evaluated.

The following table describes the categories that you can use in your evaluations.

Category	Use
XML field	Evaluate a value in an integration object record field, or compare the values in two record fields.
Object field	Evaluate the value in an object field, or compare the values in two fields in the related objects. The object field can be part of the object structure definition. The object field can also be part of an object that is accessed in a relationship with an object in the object structure definition.
Object set	Check for the existence of records in a related object.
Control	Evaluate a value or boolean integration control or a system control.

Because enterprise service processing rules are applied before objects are built, the processing rules cannot evaluate object fields or object sets. You can use the following combinations of categories, processing direction (outbound or inbound), and record types (record or object) in your evaluations.

Direction of processing rule	XML field evaluation	Object field evaluation	Object set evaluation	Control evaluation
Outbound	Available	Available	Available	Available
Inbound (record)	Available	Not available	Not available	Available
Inbound (Object)	Available	Available	Available	Available

Field to evaluate:

For XML field and object field evaluations, you specify the field that you evaluate.

For an object evaluation, you specify the object and the relationship to access the field. If the field value is a derivative of the object, which matches the record, no relationship is required.

Type of evaluation:

Evaluations generally involve the comparison of two values or a check for the existence of an object set or a null value.

The user interface displays a subset of the types depending on the category of evaluation (XML field, object field, object set, or control). The following table lists the possible types of evaluations that you can use.

Type of evaluation	Description
EQUALS	The value in the specified field is equal to the value of a second field (the comparison field).
NOTEQUALS	The value in the specified field is not equal to the value of a second field (the comparison field).
GREATER	The value in the specified field is greater than the value of a second field (the comparison field).
GREATEROREQUAL	The value in the specified field is greater than or equal to the value of a second field (the comparison field).
LESS	The value in the specified field is less than the value of a second field (the comparison field).
LESSOREQUAL	The value in the specified field is less than or equal to the value of a second field (the comparison field).
LIKE	The value contains the expected value.
NOTLIKE	The value does not contain the expected value.
ISNULL	The specified field contains no value or a null value.
ISNOTNULL	The specified field contains a value.
NONE	This option is available only if the **When to Evaluate** field is configured as Changed or Not Changed. If NONE is selected, no further evaluation is necessary.
EXISTS	Records exist in the specified object set.
NOTEXISTS	No records exist in the specified object set.

When to evaluate a field:

For XML field and object field evaluations, the processing rule first determines whether to evaluate the specified data.

The system evaluates the data by checking the Evaluate When field, which can have one the following values:

Value	Action
CHANGED	The evaluation continues only if the activity that generated the message changes the specified field.
NOT CHANGED	The evaluation continues only if the activity that generated the message does not change the specified field.
ALWAYS	The evaluation continues whether or not the value of the activity that generated the message (default) changes the specified field. If you specify this option, you cannot specify a comparison type of None.

When a record is updated, a changed attribute (changed="1") appears on the corresponding field in the outbound message. This attribute determines whether the field meets the criteria in the **Evaluate When** field.

This attribute does not appear in messages generated by the Data Export feature. Evaluations that are applied when a value has changed, might not provide the right output in a data export scenario.

The changed attribute does not apply to inbound messages.

Comparison field specifications:

The user interface displays subsets depending on the type of evaluation (XML field, object field, object set, or control). If a processing rule uses one of the first eight evaluation types, it must specify the field (comparison field) with which it is making the comparison.

The following table lists the possible types of comparison fields that you can use in the field comparisons. Comparison of an alphanumeric source field is case sensitive.

Field	Use
Integration control	Compare the value in the specified field with the values in a list or value integration control. If a list control has multiple matching values, the evaluation is true. The true evaluation occurs only if the field value matches any one of the values in the list control. Example: Validate the STATUS of a purchase order. The current value in a STATUS field is WAPPR and the possible acceptable values that satisfy the condition are in a list control called POSEND. The values in POSEND are WAPPR, APPR, and CLOSE. If the evaluation type is EQUALS, the evaluation returns a true value.

Field	Use
Value	Compare the value in the specified field with a predefined value. This option is available for user-defined conditions. Regardless of the locale setting of the application server or the database, all decimal fields must use a period (.) as the decimal placeholder. Numbers to the left of the placeholder are not formatted. This format applies to inbound and outbound data. For example, $1,738,593.64 must be in the following format: 1738593.64. Example: A processing rule compares the value of the POLIN1 field with the value SPARE. If the evaluation type is EQUALS and the two values are the same, the evaluation returns a true value.
MAXVAR	Compare the value in the specified field with the value in a system control (a value in the MAXVARS database table). Example: Evaluate the OWNERSYSID on any enterprise service or publish channel to determine if it is the same as MAXVARS.MXSYSID.
Boolean	Compare the value in the specified field with a Boolean value (true or false).
Comparison field	Compare the value in the specified field with another field in the same object. Example: Compare the GLDEBITACCT value and GLCREDITACCT value on a PO line or a journal entry to determine if they are equal.
Object, relationship, and field	Compare the value in the specified field with a field in a different object. Example: Check the OWNERSYSID of inventory in the system for the item-storeroom values on a receipt line or a PO Line.

Integration controls

Integration controls give you the ability to configure the behavior of any enterprise service or publish channel according to the requirements of individual organizations and sites. Processing rules and Java classes can access integration controls for evaluation purposes.

Integration controls are defined at the system level. You can assign controls to one or more enterprise service and publish channel. The control values can be configured at the external system level. Two external systems that process the same enterprise service can share the same processing logic, class files, and processing rules, yet they process the data differently because they use different control settings.

Control levels:

All master data and documents are stored at the system level, organization level, or site level. Item data is stored at the system level, accounting information at the organization level, and work orders at the site level. An implied hierarchy exists among these levels.

An integration control can be configured to override values at any of the following levels:

Control value	Description
System-level	A system-level value applies to all system organizations and sites. If the control is not configured for organization-level or site-level values, system processing uses the system default. If the control is configured for organization-level values, or site-level values but none exists for a particular organization or site, system processing uses the system-level value.
Organization-level	An organization-level value applies to all system sites within an organization. If a control is configured for organization-level values but none exists for a particular organization, system processing uses the system-level value.
Site-level	A site-level value applies to a specific site within a system organization. If a control is configured for site-level values but none exists for a particular site, system processing uses the organization value (if one exists) or the system-level value.

Data that is processed by enterprise services or publish channels that use a control with an organization or site override must be at the organization or site level.

Control types:

You can create four types of integration controls to meet your business needs.

Boolean controls

A boolean integration control specifies a value of 0 (false) or 1 (true).

List controls

A list integration control contains a list of values. You can enter multiple values for the control and optionally assign a system domain to the control. Assigning a domain ensures the validation of any value that is entered for that control, at any level. If a domain is not assigned, there is no validation of the values that are entered.

For example, work orders are sent to an external system only if the status of the work order is APPR (approved) or COMPLETE. To determine whether to send the work order, the Java code or the processing rule can check the status of a work order against a list control that contains these two values.

Value controls

A value integration control contains a single value. You can enter a single value for the control and optionally assign a system domain to the control.

Cross-reference controls

A cross-reference control replaces one value with another. In a publish channel, a system value is converted to an external system value. In an enterprise service, an external system value is converted to a system value. You can optionally assign a system domain to a cross-reference control. If a domain is specified, any system value that is specified for the control is validated against that domain. If a domain is not assigned, there is no validation of the values that are entered.

Cross-reference controls must have a one-to-one mapping between the system value and the external system value. If two system values are associated with an external system value, or two external system values with a system value, a processing error occurs.

If you create the cross-reference control to function as a multiplication control on an enterprise service, one-to-many mappings can exist. A multiplication control is a cross-reference control that copies, or multiples, an inbound message for multiple organizations or sites. A multiplication control has one external value and multiple system values.

Multiplication controls are always specific to the external system. You identify the control as a multiplication control on the Enterprise Service tab in the Enterprise Services application.

For example, the system sites correspond to external system business units, but the two systems use different values for these entities. A cross-reference control can perform the translation between the two values. A cross-reference control in an enterprise service can translate business unit EX001 to system site MX001. In a publish channel, the same control can translate MX001 to EX001.

Multiplication controls

A multiplication control can update the company in every organization in the system database. For example, use a multiplication control to update the company in every organization within the system. Value updates occur when the system receives company data using an enterprise service.

New control creation:

Modifying control values at the external system level is generally sufficient to customize predefined enterprise service or publish channel processing. If new business rules are implemented or a new publish channel and enterprise service is implemented, a new control might be needed.

Use the following guidelines when you create new controls:

- Control names must unique.
- To use the controls as part of a processing rule, and to set a value at the external system level, associate controls with a publish channel or enterprise service.

- When you associate a publish channel or enterprise service to an external system, all associated controls are copied to the external system level. You can assign values at the external system level.

Configuring processing rules

You can configure processing rules for an object structure in the Publish Channels application or in the Enterprise Services application.

Defining integration control or system control evaluations:

You can use a control evaluation to compare a value in a specified field with the value in an integration control. You can also use a control evaluation to compare the value in a specified field with a system control value (a MAXVARS value).

About this task

If you evaluate an integration control, it must be a Boolean or Value type control.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the evaluation applies.
2. On the Object Structure Sub-Records table window, select the object structure to which the evaluation applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Add/Modify Conditions** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Add/Modify Conditions** on the Processing Rules for Sub-Record table window.
4. On the Conditions table window, click **New Row**.
5. In the **Conditions** field, enter a value. The condition value determines the order in which the integration framework evaluates the conditions.
6. To compare the control value with a field value, select a one of the following radio buttons and enter values in the appropriate fields.
   - **Integration Control**
   - **MAXVAR**
7. Enter values in the following fields:

Option	Description
Evaluation Type	Defines the type of evaluation that is carried out on the XML field.
Value	The value that is used in the control evaluation.

8. Click **OK**.

Defining object field evaluations:

You can use an object field evaluation to evaluate the value of a field in any object that is included in the definition of an object structure. You can also evaluate other business objects that can be accessed by using a WHERE clause.

About this task

Regardless of the locale setting of the application server or the database, all decimal fields must use a period (.) as the decimal placeholder. There is no formatting of numbers to the left of the placeholder. This format applies to inbound and outbound data. For example, $1,738,593.64 must be 1738593.64.

You can use an object field evaluation to perform the following evaluations:

- Check if the field is null or not null
- Compare the value in the object field with the value of an integration control or a system control
- Compare the value in the object field with predefined value

If the data does not satisfy the comparison, the evaluation returns a false result. If the data satisfies the comparison, the evaluation returns a true result.

Procedure

1. In the Publish Channels application, display the channel record to which the evaluation applies.
2. In the Object Structure Sub-Records table window, select the object structure to which the evaluation applies.
3. In the Processing Rules for Sub-Record table window, click **Add/Modify Conditions**.
4. On the Conditions table window, click **New Row**.
5. In the **Conditions** field, enter a value. The condition value determines the order in which the integration framework evaluates the conditions.
6. On the Object Field tab, click **New Row**.
7. Enter values in the following fields:

Option	Description
Object	The business object that contains the field that is evaluated.
Object Relationship	The relationship between the defined business object and the rule business object.
Field	The business object field that is evaluated.
Evaluation Type	The type of evaluation that is carried out on the business object field.
Evaluate When	How often the evaluation is performed.

   Enter a business object value only if it is not the in the business object on which the rule is created.
8. To compare the value of the business object field with another value, select a one of the following radio buttons and enter values in the appropriate field.
   - **Integration Control**
   - **Value**
   - **MAXVAR**
9. Click **OK**.
10. Click **Save Publish Channel**.

Defining object set evaluations:

You use an object set evaluation to determine whether records exist in a relationship between two business objects. If the relationship returns an business object set, the evaluation returns a true result.

Procedure

1. In the Publish Channels application, display the channel record to which the evaluation applies.
2. In the Object Structure Sub-Records table window, select the object structure to which the evaluation applies.
3. In the Processing Rules for Sub-Record table window, click **Add/Modify Conditions**.
4. On the Conditions table window, click **New Row**.
5. In the **Conditions** field, enter a value. The condition value determines the order in which the integration framework evaluates the conditions.
6. On the Object Set tab, click **New Row**.
7. Enter values in the following fields:

Option	Description
Object	The business object that contains the field that is evaluated.
Object Relationship	The relationship between the defined business object and the rule business object.
Evaluation Type	The type of evaluation that is carried out on the business object field.

8. Click **OK**.
9. Click **Save Publish Channel**.

Defining XML field evaluations:

You can use an XML field evaluation to evaluate a value in an object structure sub-record. When you use an XML field evaluation, you can check whether a field is null, and compare values in a business object field with an integration control, a system value, or a predefined value.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the evaluation applies.
2. On the Object Structure Sub-Records table window, select the object structure to which the evaluation applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Add/Modify Conditions** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Add/Modify Conditions** on the Processing Rules for Sub-Record table window.
4. On the Conditions table window, click **New Row**.
5. Optional: In the **Conditions** field, enter a value. The condition value determines the order in which the integration framework evaluates the conditions.

6. On the XML Field tab, click **New Row**.
7. Enter values in the following fields:

Option	Description
Field	The XML field that is evaluated.
Evaluation Type	The type of evaluation that is carried out on the XML field.
Evaluate When	How often the evaluation is performed.

8. To compare the value of the XML field with another value, select a one of the following radio buttons and enter values in the appropriate field.
   - **Integration Control**
   - **Value**
   - **MAXVAR**
   - **Comparison Field**
9. Click **OK**.

Defining processing rules:

You can define a processing rule to perform custom enterprise service and publish channel processing. When you use a processing rule, you can perform custom inbound and outbound processing without using a Java class.

Before you begin

Before you create a processing rule, you must consider defining:

- Whether an enterprise service or publish channel rule evaluates an XML field or a business object field.
- The specific sub-record or business object on which the rule is defined.
- The actions that trigger the rule.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel to which the rule applies.
2. In the Object Structure Sub-Records table window, select the object structure sub-record to which the rule applies.
3. For an enterprise service, perform one of the following actions:
   - On the Object Structure Processing Rules tab, click **New Row** to define an inbound processing rule on an object structure.
   - On the Object Processing Rules tab, click **New Row** to define an inbound processing rule on a business object.
4. For a publish channel, click **New Row** on Processing Rules for Sub-Record tab to define an outbound processing rule on a business object.
5. In the **Rule** field, enter a rule identifier.
6. In the **Action** field, enter a value.
7. To change the order in which the application applies the processing rules for an object, change the value in the **Sequence** field. The integration framework applies the rules sequentially starting with the primary-level object.
8. Perform one or more of the following actions.
   - Select or clear the **Apply on Primary Object Insert** check box.

- Select or clear the **Apply on Primary Object Update** check box.
- Select or clear the **Apply on Primary Object Delete** check box.

The noted settings determine whether the integration framework applies the processing rule when a row is inserted, updated, or deleted on the primary business object in the object structure.

9. Click **Save Enterprise Service** or **Save Publish Channel**.

Enabling processing rules:

You must enable a processing rule before the rule can be applied to enterprise service or publish channel objects. An enabled processing rule indicates that it is ready to perform custom inbound and outbound processing.

About this task

If you disable a predefined processing rule, it can cause integration framework processing errors.

Procedure

1. In the Enterprise Services or Publish Channel application, display the service or channel that contains the processing rule that you want to enable.
2. In the Object Structure Sub-Records table window, select the object to which the rule applies.
3. Specify whether you want the processing rule to be enabled or disabled.

Option	Enabled
Enabled	Selected
Disabled	Cleared

For the enterprise service, you can enable or disable the processing rule on the Object Structure Processing Rules or Object Processing Rules tab. For the publish channel,ing you can enable or disable the processing rule on the Processing Rules for Sub-Record tab.

4. Click **Save Enterprise Service** or **Save Publish Channel**.

Adding controls:

Integration controls are used to configure the behavior of any enterprise service or publish channel according to the requirements of individual organizations and sites. Control types include Boolean controls, cross-reference controls, list controls, and value controls.

Adding Boolean controls:

You can add a Boolean type control when you need an integration control that specifies a value of true or false. An enterprise service or a publish channel can use this Boolean control in its processing rule evaluations. The true or false value that you assign to the control determines whether an enterprise service or publish channel applies a processing rule on a transaction.

Procedure

1. In the Enterprise Services application, on the **Select Action** menu, select **Create Integration Controls**.
2. Click **Add New Control** > **Add New Boolean Control**.

3. In the **Integration Control** field, enter the identifier for the Boolean control.
4. Specify whether you want the Boolean control to have a default value of true or false.

Option	Default True
True value	Selected
False value	Cleared

5. Click **OK** to close the Boolean Control dialog box.
6. Click **OK** to close theCreate Integration Controls dialog box.

Example

You can use a Boolean control to indicate whether enterprise services or publish channels receive or send purchase order transactions. You can set a processing rule action on an enterprise service or publish channel to skip a transaction. If the default value that you assign to the Boolean control is true, and the processing rule evaluation is true, the enterprise service or publish channel receive and send purchase order transaction updates.

What to do next

You can associate an integration control with an enterprise service or publish channel in the Enterprise Services or Publish Channels application. These associations make the integration controls available for inbound and outbound message processing. You can also associate an integration control to an external system in the External Systems application. The value that you define on the control at the external level overwrites the control value that is defined at the enterprise service or publish channel level.

Adding cross-reference controls:

You add a cross-reference type control when you need an integration control to replace one value with another. You can replace a value in inbound or outbound messages and across multiple organizations or sites. A cross-reference control can perform the translation between a value in the asset management system and a value in the external system.

About this task

A publish channel uses a cross-reference control to convert an outbound asset management system value to an external system value. An enterprise service uses a cross-reference control to convert an inbound external system value to an asset management system value. If you use synonyms, enter the external value as the control value, not the internal application value. You must use a period (.) as the decimal placeholder when you enter decimals as a control value. The numbers to the left of the placeholder are not converted. For example, $1,738,593.64 must be 1738593.64.

Procedure

1. In the Enterprise Services or Publish Channels application, select the **Create Integration Controls** action.
2. Click **Add New Control** > **Add New XRef Control**.
3. Enter values in the following fields:

Option	Description
Integration Control	The identifier for the value control.
Domain	The domain that is used to check the values that are entered for the integration control.

4. In the Values table window, click **New Row**.
5. Enter values in the following fields:

Option	Description
Default Value	The value that is converted to or from an external system value.
Default External Value	The external system value that is converted to or from the default value.

6. Click **OK** to close the Cross-Reference Control dialog box.
7. Click **OK** to close the Create Integration Controls dialog box.

Example

The asset management system sites correspond to external system business units, but two external systems use different values for these entities. A cross-reference control can perform the translation between the two mismatched values and the asset management system value.

A cross-reference control in an enterprise service can translate the external system site value `EX001` to an asset management system site value `MX001`.

What to do next

You can associate an integration control with an enterprise service or publish channel in the Enterprise Services or Publish Channels application. These associations make the integration controls available for inbound and outbound message processing. You can also associate an integration control to an external system in the External Systems application. The value that you define on the control at the external level overwrites the control value that is defined at the enterprise service or publish channel level.

Adding list controls:

You create a list type integration control when you need a control that contains a list of values. An enterprise service or publish channel can use this list control in its processing rule evaluations. The rule could skip the processing of the transaction when the data field value does not match any of the defined list control values.

About this task

You must use a period (.) as the decimal placeholder when you enter decimals as a control value. The numbers to the left of the placeholder are not converted. For example, $1,738,593.64 must be 1738593.64.

Procedure

1. In the Enterprise Services application, on the **Select Action** menu, select **Create Integration Controls**.
2. Click **Add New Control** > **Add New List Control**.
3. Enter values in the following fields:

Option	Description
Integration Control	The identifier for the value control.
Domain	The domain that is used to check the values that are entered for the integration control.

4. In the Values table window, click **New Row**.
5. In the **Default Value** field, enter a value for evaluation.
6. Click **OK** to close the List Control dialog box.
7. Click **OK** to close the Create Integration Controls dialog box.

Example

Work orders are sent to an external system only if the status of the work order is `APPR` (approved) or `COMPLETE`. The processing rule can check the status of a work order against a list control that contains these two status values. If the status of a work order does not match the two list control values, the work order transaction is not sent to the external system.

What to do next

You can associate an integration control with an enterprise service or publish channel in the Enterprise Services or Publish Channels application. These associations make the integration controls available for inbound and outbound message processing. You can also associate an integration control to an external system in the External Systems application. The value that you define on the control at the external level overwrites the control value that is defined at the enterprise service or publish channel level.

Adding value controls:

You can create a value type integration control when you need a control that contains a single value. An enterprise service or publish channel can use this value control in its processing rule evaluations. The rule could skip the processing of the transaction when the data field value does not match the defined control value.

About this task

You can provide a default control value and assign a domain to the control to ensure the validation of any value that you enter for that control.

Procedure

1. From the **Select Action** menu in the Enterprise Services or Publish Channels application, select **Create Integration Controls**.
2. Click **Add New Control** > **Add New Value Control**.
3. Enter values in the following fields:

Option	Description
Integration Control	The identifier for the value control.
Domain	The domain that is used to check the values that are entered for the integration control.
Default Value	The default value for the integration control.

4. Click **OK** to close the Value Control dialog box.
5. Click **OK** to close the Create Integration Controls dialog box.

Example

Purchase orders are received by the asset management system only if the company type value is EX. To determine whether the purchase order is received, the processing rule can check the value of the company type against the value that is defined in the control. If the EX company value does not match the value contained in the control, the purchase order transaction is not sent to the asset management system.

What to do next

You can associate an integration control with an enterprise service or publish channel in the Enterprise Services or Publish Channels application. These associations make the integration controls available for inbound and outbound message processing. You can also set an integration control on an external system in the External Systems application. The value that you define on the control at the external system level overwrites the control value that is defined at the enterprise service or publish channel level.

Associating integration controls to enterprise services or publish channels:

You can associate integration controls with an enterprise service or a publish channel to make the control available for inbound and outbound message processing. An enterprise service or a publish channel can have an association with one or more integration controls.

About this task

When you associate an integration control to an enterprise service or a publish channel, you can override the predefined values of the controls that are set at the service or channel level. You can define integration controls globally and configure the controls for each external system, according to the requirements of individual organizations and sites.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel for which you want to associate an integration control.
2. Select the **Associate Integration Controls** action.
3. Perform one of the following actions:

Option	Description
Select integration controls individually	1. Click **New Row**. 2. In the **Integration Control** field, enter a control value.
Select multiple integration controls	1. Click **Select Controls**. 2. Select the appropriate controls. 3. Click **OK**.

4. Click **OK** to close the Associate Integration Controls dialog box.

Managing data in sub-record fields:

As part of an integration, you can work with the values in a single source data field to set the value, combine the value, split the value, or replace the value.

Setting sub-record field values:

You can assign a value to a specified data field to overwrite the existing value in the data field. You can indicate whether a value is always assigned or only assigned when the target data field is null.

Before you begin

You must create a processing rule that contains a set action before you can set sub-record field values.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the sub-record field set action applies.
2. On the Enterprise Service or Publish Channel tab, select the processing rule to which the sub-record field set action applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Sub-Record Fields** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Sub-Record Fields** on the Processing Rules for Sub-Record table window.
4. Click **New Row**.
5. In the **Field** field, enter the name of the target data field. This value defines the field that you want to replace.
6. Specify whether a value is always assigned or only assigned when the target data field is null.

Option	Replace When Null
Assign when data field is null	Selected
Always assign	Cleared

7. To specify the source field, select a one of the following radio buttons and enter values in the appropriate fields:
   - **Integration Control**
   - **MAXVAR**
   - **Field**

- **Object**

8. Click **OK**.

Combining sub-record field values:

You can combine the values in two or more source data fields or integration controls into a single target data field. You can combine values when mismatches occur between an asset management system value and the external system value. For example, a two-part external system key can map to a single part key in the asset management system.

Before you begin

You must create a processing rule that contains a combine action before you can combine sub-record field values.

About this task

The source and target fields must be in the same object. This action always overwrites the existing value in the target field. Ensure that the source and target fields are alphanumeric fields, or processing errors may occur.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the sub-record field combine action applies.
2. On the Enterprise Service or Publish Channel tab, select the processing rule to which the sub-record field combine action applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Sub-Record Fields** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Sub-Record Fields** on the Processing Rules for Sub-Record table window.
4. In the Target Sub-Record Fields table window, perform one of the following actions:

Option	Description
Select target fields individually	1. Click **New Row**. 2. In the **Field** field, enter a target data field. 3. In the **Separator Integration Control** field, enter a delimiter value that separates the segments in the target field.
Select multiple target fields at one time	1. Click **Select Field**. 2. Select the appropriate fields. 3. Click **OK**. 4. In the **Separator Integration Control** field, enter a delimiter value that separates the segments in the target field.

5. In the Source Sub-Record Fields for Target table window, perform one of the following actions:

Option	Description
Select source fields individually	1. Click **New Row**. 2. Select a one of the following radio buttons and enter values in the appropriate fields: • **Field** • **Integration Control** 3. Enter a value in the **Sequence** field to define the order in which the application move segments of the source field to the target fields.
Select multiple source fields at one time	1. Click **Select Field**. 2. Select the appropriate fields. 3. Click **OK**. 4. Enter a value in the **Sequence** field to define the order in which the application move segments of the source field to the target fields.

6. Click **OK**.

Splitting sub-record field values:

You can split the value in a single source data field into multiple target fields. You can split values when mismatches occur between an asset management system value and the external system value. For example, a single part asset management system key can map to a two-part key in the external system.

Before you begin

You must create a processing rule that contains a spit action before you can split sub-record field values.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the sub-record field split action applies.
2. On the Enterprise Service or Publish Channel tab, select the processing rule to which the sub-record field split action applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Sub-Record Fields** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Sub-Record Fields** on the Processing Rules for Sub-Record table window.
4. In the Target Sub-Record Fields table window, perform one of the following actions:

Option	Description
Select target fields individually	1. Click **New Row**. 2. In the **Field** field, enter a target data field. 3. In the **Separator Integration Control** field, enter a delimiter value that separates the segments in the target field.
Select multiple target fields at one time	1. Click **Select Field**. 2. Select the appropriate fields. 3. Click **OK**. 4. In the **Separator Integration Control** field, enter a delimiter value that separates the segments in the target field.

5. In the Target Sub-Record Fields for Source table window, perform one of the following actions:

Option	Description
Select source fields individually	1. Click **New Row**. 2. In the **Field** field, enter the name of the target field that receives the first or next segment of the source data. 3. If you did not specify a value in the **Separator Integration Control**field, enter a number in the **Field Length**field. 4. Enter a value in the **Sequence** field to define the order in which the application move segments of the source field to the target fields.
Select multiple source fields at one time	1. Click **Select Field**. 2. Select the appropriate fields. 3. Click **OK**. 4. If you did not specify a value in the **Separator Integration Control**field, enter a number in the **Field Length**field. 5. Enter a value in the **Sequence** field to define the order in which the application move segments of the source field to the target fields.

6. Click **OK**.

Replacing sub-record field values:

You can replace a value in a data field with another value. You can replace a value when the external system and the asset management system have different identifiers for the same entity. For example, a plant identifier of an external system can be translated to an asset management system site identifier.

Before you begin

You must create a processing rule that contains a replace action before you can replace sub-record field values. You must also create a cross-reference control that contains the original and new values for the data field.

Procedure

1. In the Enterprise Services or Publish Channels application, display the service or channel record to which the sub-record field replace action applies.
2. On the Enterprise Service or Publish Channel tab, select the processing rule to which the sub-record field replace action applies.
3. Perform one of the following actions:
   - For an enterprise service, click **Sub-Record Fields** on the Object Structure Processing Rules tab or the Object Processing Rules tab.
   - For a publish channel, click **Sub-Record Fields** on the Processing Rules for Sub-Record table window.
4. Click **New Row**.
5. In the **Field** field, enter the name of the target data field. This value defines the field that you want to replace. The target field must be in the sub-record object shown at the top of the dialog box.
6. In the **Integration Control** field, enter the name of the cross-reference type control that contains the original and new values for the target field.
7. Click **OK**.

Code-based customization

The integration framework provides placeholders in transaction flows where you can insert your own code to provide logic that customizes transaction processing. You can insert Java classes or automation scripts at selected points in the processing of object structures, publish channels, enterprise services, and invocation channels. You can also use XSL maps to transform messages.

When you create or update a Java class file, you must rebuild and redeploy the application EAR file before integration components can use it. You can write a script in one of the supported scripting languages in the Automation Scripts application, or you can import a script that you created externally, and you can activate it for immediate use. When you create or update an XSL map, you can either store it in the application EAR file or in a directory on your file system.

When the customization code is configured for use, the integration framework calls the code at the appropriate point during transaction processing.

You can download sample Java classes or XSL files from the IBM Tivoli Open Process Automation Library (OPAL).

Customization Java classes and methods

You can extend the standard processing of outbound and inbound messages by using Java classes. Multiple placeholders exist in message processing flows where you can insert custom Java code to affect message content or processing or both.

External exit classes:

External exit classes provide a base class, ExternalExit, that you can extend to customize publish channel, enterprise service, and invocation channel processing.

For outbound messages, the input to the external exit class is the irData element of the XML message. The output of the class processing is the erData element of the XML message that is delivered to the target destination. For inbound messages, the input to the external exit class is the erData element of the inbound message. The output is the irData element that is delivered to the related object structure for processing into Maximo Asset Management.

To use an external exit class for customization, specify your Java class in the **Processing Class** field in the Publish Channels application or in the Invocation Channels application.

An alternative approach to integration customization is to implement a user exit class. If an adapter for Oracle or SAP is installed, predefined processing classes are provided, and customizations must be done by using the user exit class. When no predefined processing classes exist, you can choose whether to implement custom code in the external exit class or the user exit class.

User exit classes:

For inbound and outbound transactions on a publish channel, an enterprise service, or an invocation channel, a user exit class can have a preprocessor method and a postprocessor method. For inbound transactions, the user exit class can have an extra method that processes business objects just before they are committed to the database.

Preprocessing method for outbound transactions:

The input to the first outbound user exit point is the XML that is generated by the object structure. If you have an ERP adapter processing class, you can use either the preprocessing method or postprocessing method to implement your custom logic, depending upon your requirement.

By using the preprocessing exit point, you can change data in the message which later alters the processing logic of a processing class when it is implemented.

Outbound message customization can be done in the preprocessor method, by using the following method:

```
public StructureData setUserValueOut(StructureData irData)
```

This method can perform the following processing:

- Validate data.
- Change system data by changing the IR record that is sent to the external system.
- Stop the message from being sent out of Maximo Asset Management by throwing a system exception. The entire message is rolled back. If the message is initiated by an event, the entire message is also rolled back, including any changes that you make in an application.
- Stop the message from being sent to the external system by throwing a SKIP_TRANSACTION exception.
- Log the transaction.

Postprocessing method for outbound transactions:

In the postprocessing method of the user exit, the IR element from the object structure and the ER element from the XML output of the processing class are both available for processing.

Use the following method to customize outbound messages.

```
public StructureData setUserValueOut(StructureData irData, StructureData erData)
```

The method can perform the following processes:

- Validate data.
- Change data by changing the ER record that is sent to the external system.
- Map additional data from the irData element to the erData element.
- Stop the transaction from being sent out of Maximo Asset Management by throwing a system exception. The entire message is rolled back. If the message is initiated by an event,the entire message is also rolled back, including any changes that you make in an application.
- Stop the message from being sent to the external system by throwing a SKIP_TRANSACTION exception.
- Log the transaction.

You identify the publish channel user exit class in the Publish Channels application.

Preprocessing method for inbound transactions:

In the preprocessing method for inbound transactions, the input is the XML message that is delivered from the external system. If necessary, you can change data in the message to affect the processing logic in the processing class.

Inbound transaction customization can be done in the first exit by using the following method:

```
public StructureData setUserValueIn(StructureData erData)
```

This method can perform the following processing:

- Validate data.
- Change external data by changing the ER record before it is mapped to the IR record and saved in the system.
- Stop further processing of the transaction by throwing an exception. For queue-based messages, the transaction remains in the queue to be reprocessed. For messages that are not queue-based, the messages are returned to the client that started the enterprise service.
- Stop the message from being processed into Maximo Asset Management by throwing a SKIP_TRANSACTION exception. In this case, the message is not saved in the system. For queue-based messages, the message is removed from the queue.
- Log the transaction.

Postprocessing method for inbound transactions:

In the postprocessing method for inbound transactions, both the IR records and ER records are available for processing.

Inbound transaction customization can be done in the postprocessing flow by using the following method:

```
public StructureData setUserValueIn(StructureData irData, StructureData erData)
```

The method can perform the following processes:

- Validate data.
- Change external data by changing IR record to be saved in the system.
- Map additional data from the ER record to the IR record.
- Stop further processing of the transaction by throwing an exception. For queue-based messages, the transaction remains in the queue for reprocessing. For messages that are not queue-based, the messages are returned to the client that started the enterprise service.
- Stop the message from being processed into Maximo Asset Management by throwing a skip_transaction exception. In this case, the message is not saved in the system. For queue-based messages, the message is removed from the queue.
- Log the transaction.

You identify the enterprise service user exit class in the Enterprise Services application.

Business object processing on inbound transactions:

The user exit class on an enterprise service has an additional method, setUserMboIn, that enables custom processing at the point when the business objects are created from object structure XML, but are not yet saved to the database.

This user exit is called after the system processing and can run on the objects that are created in the system by using the following method:

```
public void setUserMboIn(MboRemote mbo)
```

The object parameter is a reference to a primary object in the object structure.

This method is called once for the primary object. For an XML transaction with multiple nouns, the object exit is called once for each noun.

The method can perform the following processes:

- Validate data.
- Stop the transaction from being saved in the system by throwing a system exception. For queue-based messages, the transaction remains in the queue and is retried.
- Log the transaction.

You identify the user exit class in the Enterprise Services application.

Event filter classes:

For outbound asynchronous transactions that use publish channels, you can run a custom Java class or an automation script by using an event filter class. The processing for an event filter class occurs before the serialization of data in the object structure layer for an outbound message. The event filter class can insert logic that affects the creation of the integration message at this processing point.

A common use of the event filter class is to include logic that removes unwanted fields or objects from a message to reduce the amount of data in the serialization process. Another common use of the class is to override default recursion logic that prevents an event from an inbound integration from sending an outbound integration message. Overriding the default behavior in this scenario can allow an outbound message to be initiated from an event that is triggered by an inbound message.

Handler exit classes:

Multiple predefined methods, or handlers, are provided so that you can send data to an external system, including HTTP, an enterprise Java bean call, and interface tables. Some of these methods have user exit placeholders that are available for customization.

Enterprise bean processing user exit class:

You can use an exit placeholder for class customization when transactions are sent to an external system by an enterprise bean. This exit class is optional and is called before the enterprise bean is called.

The implementation of this Java class must resolve the method signature of the enterprise bean that is started by this handler and the parameters that the method requires. If no value is specified for this property, the system applies a default exit called `DefaultEJBExit`. This default exit attempts to resolve the enterprise bean method signature and parameters.

This class must implement the `psdi.iface.router.EJBExit` interface and the following methods:

- The `getClassParams()` method returns the method signature in the form of an array of Java classes:
  ```
  public Class[] getClassParams()
  ```
- The `getObjectParams` method returns an array of the parameters of the enterprise beans as an array of Java objects:
  ```
  public Object[] getObjectParams(byte[] data, String interfaceName, String
  destinationName)
  ```
- After the successful invocation of the enterprise bean, the `responseOk()` method is called with an object as the response of the invocation:
  ```
  public void responseOk(Object response)throws MXException
  ```
- If an error is encountered during the enterprise bean invocation, the `responseError()` method is called with the originating exception as a parameter:
  ```
  public void responseError(Exception e) throws MXException
  ```

You enter the fully qualified name of the Java class in the EJBEXIT property of the endpoint that implements the enterprise bean handler.

HTTP processing user exit class:

When you send outbound transactions to an external system by using HTTP, you can use an HTTP exit placeholder for customization. This exit class is optional and is called as part of the response from the HTTP call.

The HTTP processing exit class runs when a response is returned to an HTTP Post from an external system. In the default implementation of the

psdi.iface.router.DefaultHTTPExit class, the response code that is received from the external system is compared to a range of response codes. The range that is used by the default implementation is 200 - 299. If the code is outside that range, then the transaction was not delivered to the external system, and an exception is raised.

With some external systems, the response from an HTTP call is interpreted to see whether the external system accepted the message. The interpretation logic can be implemented in the HTTP exit class.

If the publish channel message is not accepted, the code must raise an exception. The message in the outbound queue is marked as an error and is not removed from the queue. If the message is accepted, the message is removed from the outbound queue. If the invocation channel message fails, the exception is returned to the invoker of the channel, and the invoker handles the exception according to the design requirements.

The HTTP processing exit class must implement the psdi.iface.router.HTTPExit interface and implement the following method:

```
public void processResponseData(int responseCode, String responseMsg,
byte[] msgBodyData)
```

If necessary, this class can interpret the response code and throw an exception. The class can perform the following actions:

- Check the response code from the HTTP post.
- If the response code is in the error range, the exception is logged on the ERROR level, and a system exception is thrown.
- If the response code is valid, the transaction is logged on the DEBUG level.

You must specify the fully qualified name of the Java class in the **HTTPEXIT** property of the endpoint that implements the HTTP handler.

JMS processing user exit class:

When you send transactions from the system to an external system by using JMS, you can use an exit placeholder for customization. This exit class is optional and is called before the JMS is called.

This class must implement the psdi.iface.router.JMSExit class and the following method:

```
public Map getMessageProperties(Map metaData, (byte[] data, Map
origProps)throws MXException
```

This method can perform the following processes:

- Change the properties of the JMS message
- Split the data to multiple properties, to match the JMS message

You enter the fully qualified name of the Java class in the JMSEXIT property of the endpoint that implements the JMS handler.

Web service processing user exit class:

You can use an exit placeholder for customization when you send transactions from the system to an external system by using a web service. This exit class is optional and is called before the web service is started.

This class must implement the `psdi.iface.router.WSExit interface` and the following methods:

`getServiceName()` method
The `getServiceName()` method returns the service name of the web service to start:

```
public String getServiceName(Map metaData, String endpointURL,
String serviceName, String interfaceName, String targetNameSpace)
throws MXException
```

`getEndpointURL()` method
The `getEndpointURL()` method returns the endpoint URL of the web service to start:

```
public String getEndpointURL(Map metaData, String endpointURL,
String serviceName, String interfaceName, String targetNameSpace)
throws MXException
```

`responseOk()` method
The `responseOk()` method is called after a successful invocation of the external web service.

```
public void responseOk(org.w3c.dom.Document response) throws
MXException
```

`responseError()` method
If an error is encountered during the web service invocation, the `responseError()` method is called with the originating exception as a parameter:

```
public void responseError(Exception e) throws MXException
```

`getOneWayWsInfo()` method
The `getOneWayWsInfo()` method returns a Boolean value that specifies whether the web service to start is one way:

```
public boolean getOneWayWsInfo(Map metaData, String endpointURL,
String serviceName, String interfaceName, String targetNameSpace,
boolean oneWayWs) throws MXException
```

`getSoapAction()` method
The `getSoapAction()` method returns the SOAPAction HTTP header to use to start the web service:

```
public String getSoapAction(Map metaData, String endpointURL, String
serviceName, String interfaceName, String targetNameSpace, String
soapAction) throws MXException
```

You must specify the fully qualified name of the Java class in the **WSEXIT** property of the endpoint that implements the web service handler.

The `psdi.iface.router.DefaultWSExit` class is a default implementation of the WSExit interface. This class overrides the `getEndpointURL()` method to concatenate the service name at the end of endpoint URL to form the new endpoint URL.

Customization with automation scripts

You can use automation scripts in place of Java classes to extend the main Java processing classes for predefined integration components. You can use automation scripts to apply custom logic at various points during the processing of object structures, publish channels, enterprise services, and invocation channels.

You define automation scripts for integration by using the Automation Scripts application. Defining a script for integration identifies which integration component the script is associated with and at what point in the processing flow that the script runs. To test scripts, you can set the log level to debug, activate the script, and use the data import or data export feature to initiate an integration transaction.

Creating automation scripts for integration:

An automation script can customize the processing of integration messages for inbound and outbound transactions. You associate an automation script with an integration component, and you configure when to insert it into integration processing.

Procedure

1. In the Automation Scripts application, select the **Create** > **Script for Integration** action.
2. Specify an integration component to associate with the automation script and then specify the insertion point for the script.
3. Optional: Select the **Activate** check box if you want the script to be immediately active.
4. Optional: Specify the logging level to apply when the script executes. For testing purposes, if you set the logging level to debug, the logs contain useful information that helps you troubleshoot any issues.
5. Specify the scripting language of the script.
6. Either enter the script directly into the **Source Code** field or browse to the location where the script is stored and click **Import**.
7. Click **Create**.

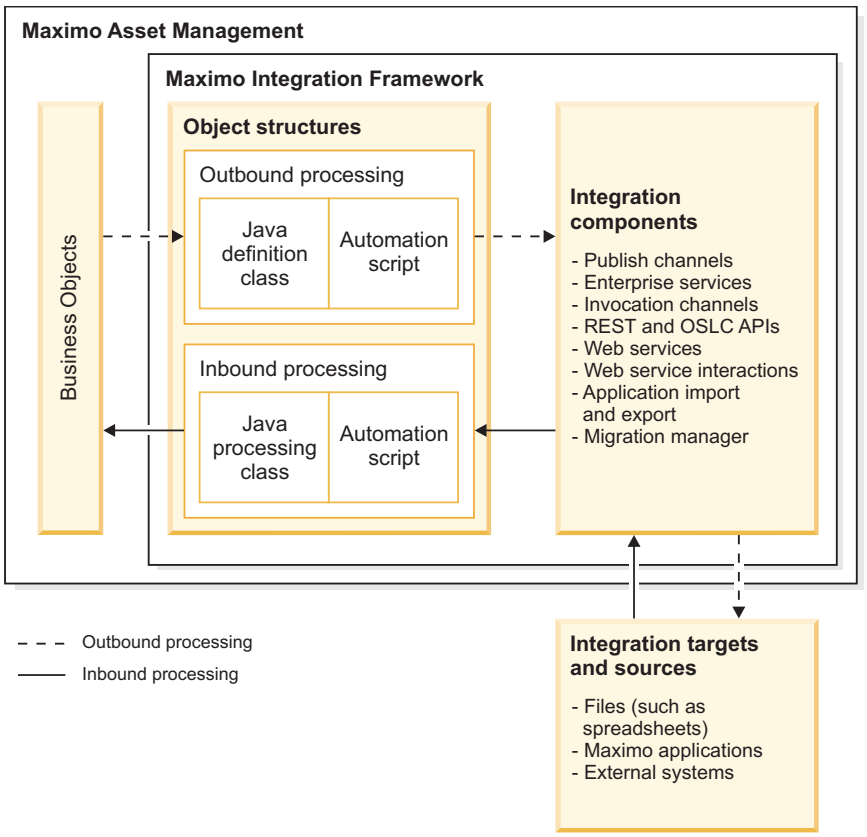
Customization of object structure processing with automation scripts:

Object structure processing supports the simultaneous use of automation scripts and Java classes for customization. Automation scripts that are defined for an object structure can affect the processing of data that is directly processed through the object structure, such as the REST API, or through other integration components such as publish channels and enterprise services.

Insertion points for customization of object structure processing with automation scripts:

You can insert script-based processing into the Java definition classes and Java processing classes that transform object structures during integration processing.

The illustration shows the insertion of automation scripts during the inbound and outbound processing of object structures.



Outbound object structure processing implements a Java definition class that allows for the insertion of custom processing during the serialization of business object data into an XML message. You can also implement an automation script to customize the processing of the object structure. The integration framework supports the customization of an object structure to be a Java class, an automation script, or both simultaneously.

Inbound object structure processing implements a Java processing class that allows for insertion of custom processing of data from the inbound XML message before it is mapped to the business object data. In addition to the Java processing class, you can implement an automation script to customize the processing in the object structure.

Outbound processing of object structures:

Outbound object structure processing performs the serialization of business object (mbo) data into an XML message. During this processing, an automation script can insert custom processing logic to change the default processing behavior of the object structure. The script can be used instead of the Java definition class or can be used with it.

Functions, such as skipMbo(ctx), are used in serialization. In the function name, *ctx* is the object in the function that is prepared by the integration framework and communicates data between the integration framework and the automation script. An automation script can affect the processing of the data and communicate those changes back to the integration framework.

Serialization includes the following functions that can be used by automation scripts to insert customized logic into the processing of the data:

skipMbo(ctx)
Filters object data from the XML message. APIs are available to filter data from the XML message, continue processing for data that you want in the XML message, and complete processing of the object structure after data is filtered out. For example, if the object structure is constructing an XML message for a purchase order, an automation script can insert logic that filters out purchase order line data where the line type is for service lines.

skipCols(ctx)
Filters a column or multiple columns from the object structure. For example, if the object structure is constructing an XML message for an asset, an automation script can insert logic that filters out a number of columns that are not needed by the system that will receive the XML message.

overrideValues(ctx)
Sets the value of an object structure field in the XML message. For example, if the object structure is constructing an XML message for an asset, an automation script CAN insert logic that might filter out a number of columns that are not needed by the system that will receive the XML message that is created.

Functions for outbound processing of object structures:

The skipMBO(c), skipCols(ctx), and overrideValues(ctx) functions are used in automation scripts for outbound processing of object structures. Each function has its own APIs that are available to process the data in the object structure. All example scripts are written in Jython.

Skip processing of business objects by using the skipMbo(ctx) function:

The skipMbo(ctx) function filters data from the XML message that is built during the serialization process. The skipMbo(ctx) function can filter a Maximo business object, continue processing the message after a business object is skipped, or terminate the processing of the message at the point of execution.

ctx.skipMbo() API

The ctx.skipMbo() API filters out the processing of any Maximo business objects in the XML message.

For example, the following script executes on the ctx.skipMbo() API, and filters out all purchase order lines and their related purchase order cost data for the 1234 purchase order line.

```
def skipMbo(ctx):
 if ctx.getMboName()=='POLINE' and ctx.getMbo().getString("itemnum")=="1234":
  ctx.skipMbo()
```

If you use the ctx.skipMbo() API on the root or primary object in the object structure, any business object that meets the criteria in the IF statement is filtered from the XML message.

ctx.process() API

The ctx.process() API continues processing lines that are not filtered out by the ctx.skipMbo() API, for example:

```
def skipMbo(ctx):
 if ctx.getMboName()=='POLINE' and ctx.getMbo().getString("itemnum")=="1234":
  ctx.skipMbo()
 else
  ctx.process()
```

Executing the ctx.process() API on a row of data enables that data to be serialized. If the Java definition class skips that row of data, the ctx.process() API overrides the Java definition class and allows the data to be serialized.

ctx.complete() API

When the ctx.complete() API is invoked, the outbound processing of the object structure is stopped for the current instance of the business object, for instances of the child business objects, and for instances of its peer-level business objects.

For example, the PO object consists of the child objects POLINE and POTERM. The POLINE object consists of a child object POCOST. When the ctx.complete() API is invoked on the PO object, the POLINE, POTERM, and POCOST objects that are related to that PO are not processed.

In the following example script, any purchase order that has a status of complete is processed without the serialization of its purchase order line data.

```
def skipMbo(ctx):
 if ctx.getMboName()=='PO' and ctx.getMbo().getString("status")=="COMPLETE":
  ctx.complete()
 else
  ctx.process()
```

Skip processing of columns in business objects by using the skipCols(ctx) function:

The skipCols(ctx) function filters the columns of a Maximo business object from the processing of the XML message. The ctx.skipCol(String) API can accept one string argument to filter a single column or multiple comma-separated string arguments to filter multiple columns.

The ctx.skipCol(String) API can filter a single column of a Maximo business object. The following example script filters out the description column of the ASSET object:

```
def skipCols(ctx):
   if ctx.getMboName()=='ASSET':
      ctx.skipCol("description")
```

You can also use the ctx.skipCol(String) API to filter multiple columns. The following example script filters out the description and the asset number columns of the ASSET object:

```
def skipCols(ctx):
   if ctx.getMboName()=='ASSET':
      ctx.skipCol("description","assetnum")
```

The ctx.skipCol(String) API can filter the column of a child object in an object structure. For example, the following script filters out the METERNAME column of the ASSETMETER object:

```
def skipCols(ctx):
   if ctx.getMboName()=='ASSETMETER':
      ctx.skipCol("metername")
```

You can also use the ctx.skipCol(String) API to filter columns based on a specified value. For example, the following script filters out the description column when the assettag attribute has a value of 12593:

```
def skipCols(ctx):
   if ctx.getMboName()=='ASSET' and ctx.getMbo().getString("assettag")=="12593":
      ctx.skipCol("description")
```

Set values in fields by using the overrideValues(ctx) function:

The overrideValues(ctx) function sets the value of a field in the XML message.

The ctx.overrideCol() API sets the value of the field in the XML message. For example, the following script sets the value of the site ID to ABC:

```
def overrideValues(ctx):

   if ctx.getMboName()=='ASSET':
      ctx.overrideCol("SITEID","ABC")
```

In the following example script, if the description field of a purchase order is empty, the description field is set to the purchase order number:

```
def overrideValues(ctx):
 if ctx.getMboName()=='PO':
  mbo = ctx.getMbo()
  if mbo.isNull("description"):
 ctx.overrideCol("DESCRIPTION",mbo.getString("ponum"))
```

Java definition classes and automation scripts:

You can use a Java definition class and an automation script at the same time on an object structure. The Java definition class is executed before the automation script for each function.

Some of the object structures that are included in the integration framework and Migration Manager provide Java definition classes that filter data. For example, the item object structure filters items that are of the type TOOL. When an automation script is implemented with a Java definition class, the automation script can override the filtering of the Java definition class and remove the filtering of the Java class or change the filtering criteria to exclude more data from or include more data in the XML message.

For example, the Java definition class of the MXOPERLOC object structure filters out the locations of the type LABOR while allowing locations of the type OPERATING to be included in the message. You can change the filtering, add more filtering, or replace the default filtering. The following example script changes the processing to filter the locations of the type OPERATING and continues to process locations of the type LABOR:

```
def skipMbo(ctx):
   if ctx.getMboName()=='LOCATIONS':
      if ctx.getMbo().getString("type")=="LABOR":
         ctx.process()
      if ctx.getMbo().getString("type")=="OPERATING":
        ctx.skipMbo()
```

The following example script adds more filtering for the MXOPERLOC object structure by filtering out type COURIER:

```
if ctx.getMboName()=='LOCATIONS':
   if ctx.getMbo().getString("type")=="COURIER":
      ctx.skipMbo()
```

The following example script replaces the definition class filtering so that only locations of type COURIER are filtered out:

```
def skipMbo(ctx):
   if ctx.getMboName()=='LOCATIONS':
      if ctx.getMbo().getString("type")=="COURIER":
         ctx.skipMbo()
     else:
         ctx.process()
```

Inbound processing of object structures:

The processing of inbound messages by the integration framework supports the use of automation scripts on the object structure to support custom logic. This use of automation scripts allows customization for any inbound integration message that uses an object structure and for customization of processing Migration Manager data.

For integration processing, a script that is implemented on the object structure applies to messages from the following sources:

- Data import
- REST
- Application import
- Web service
- OSLC
- Integration servlet
- Integration tables

The object structure supports a Java processing class or an automation script. You can use a Java processing class, an automation script, or both on the same object structure. The processing supports functions that are in either the class file or the automation script.

A *context* (ctx) is passed between the processing class of the inbound object structure and the automation script, which implements the custom code. The context is supported bidirectionally. Predefined APIs are available for the context. The script framework provides a built-in Java class that prepares a context.

All of the following code examples and fragments use JavaScript. You can implement the logic at the following points in the processing:

- Before the processing of a Maximo business object. For example, the beforeProcess(ctx) function is processed once for each noun in the inbound message before business objects that are related to the message are created.
- During the processing of a Maximo business object. For example, the beforeCreateMboSet(ctx) function is processed sequentially for each object within every noun in the inbound message. Other functions that are processed sequentially include afterCreateMboSet(ctx), mboRules(ctx), beforeMboData(ctx), and afterMboData(ctx).

- After Maximo business object processing completes. For example, the preSaveRules(ctx) and changeStatus(ctx) functions are processed once for each noun in the inbound message after the business objects are created.

In the Automation Scripts application, you can create a script for integration to include in the object structure processing.

Functions for inbound processing of object structures:

Each function provides APIs that you can use to customize the processing of the data in the object structure.

Skip or change message processing by using the beforeProcess(ctx) function:

The beforeProcess(ctx) function provides an injection point to run the logic in the automation script before business objects are created.

For example, you can skip the processing of a message that is based on the evaluation of the data in the XML message. Another example is to change the action attribute for the message from Sync to Create.

The following JavaScript example sets the message action to Sync:

```
function beforeProcess(ctx)
{
     ctx.setMsgType("Sync");
}
```

Create business object sets by using the beforeCreateMboSet(ctx) function:

The beforeCreateMBOSet(ctx) function processes inbound data before the framework creates the Maximo business objects or business object sets and can operate on all business objects in the object structure.

If an automation script that is processing an object structure creates a business object set, then the createMboSet() function in the base class is skipped. Processing of a child business object supports access to the parent object by using the getParentMbo() function.

You can also use the beforeCreateMboSet(ctx) function to conditionally create the business object that is based on the data that is included in the XML message.

You cannot create a child business object that is independent of its parent business object.

The following example shows how to create a child business object from a parent business object.

```
importPackage(Packages.psdi.server);
function beforeCreateMboSet(ctx)
{
 var struc=ctx.getData();
 var ponum=struc.getCurrentData("PO",ctx.getUserInfo();
 var siteid=struc.getCurrentData("SITEID");
 var poSet = MXServer.getMXServer().getMboSet("PO",ctx.getUserInfo());
 poSet.setQbeExactMatch(true);
 poSet.setQbe("ponum",ponum);
 poSet.setQbe("siteid",siteid);
```

```
 var poMbo = poSet.moveFirst();
 var polineSet = poMbo.getMboSet("POLINE");
 ctx.setMboSet(polineSet);
}
```

Change business objects or business object sets by using the afterCreateMboSet(ctx) function:

The afterCreateMboSet(ctx) function processes the inbound object structure after the framework creates the Maximo business object or business object set.

The afterCreateMboSet(ctx) function is available to operate on all business objects in the object structure. This function can change the business object or business object set that the framework created or can inject more business objects into the business object set.

The following example sets an **MboSet** property to note that the created location has a type of Storeroom. This logic replaces the default logic that is in the MXSTORELOC processing class.

```
importPackage(Packages.psdi.server);
function afterCreateMboSet(ctx)
{
    var loc type = MXServer.getMXServer().getMaximoDD().getTranslator()
        .toInternalString("LOCTYPE",ctx.getData().getCurrentData("TYPE"));

    if (loc Type=="STOREROOM")
        {
            ctx.getPrimaryMboSet().setStoreroom();
        }
    else
        {
            ctx.getPrimaryMboSet().setNonStoreroom();
        }
}
```

Change processing of rules by using the mboRules(ctx) function:

The mboRules(ctx) function can skip the processing of a business object, skip a transaction, continue processing, or create a business object. This function operates before the creation of each Maximo business object.

The following example shows how to skip a transaction:

```
function mboRules(ctx)
{
     ctx.skipTxn();
}
```

Set values in fields by using the beforeMboData(ctx) function:

The beforeMboData(ctx) function is available when the Maximo business object is created but before the values in the business object are set by the integration framework. Processing of the object structure can change the setting of business object data. Processing can also set flags on the fields, such as the samevaluevalidation flag, which triggers field validations even when the field value is set with the current value of the field.

Add business objects and change values in fields by using the afterMboData(ctx) function:

The afterMboData(ctx) function implements custom logic to create an extra related Maximo business object for inclusion in the transaction or change data in a

Maximo business object. The function is available after the business object is created and the values are set in the Maximo business object by the integration framework.

When columns in the object structure are restricted, you can use the afterMboData(ctx) function to provide the logic to set those columns instead of having the integration framework set the values.

In the following example, an asset is created with the description field set to a concatenation of the EXTERNALREFID field and the description field that is passed from the object structure:

```
importPackage(Packages.psdi.server);
importPackage(Packages.psdi.mbo);
function afterMboData(ctx)
{
     var mbo = ctx.getMbo();
     var struc = ctx.getData();
     mbo.setValue("description", "FROM: "+struc.getCurrentData("EXTERNALREFID")+ "   DESC: "+struc.getCurrentData("DESCRIPTION"));
}
```

Change the transaction before saving to the database by using the preSaveRules(ctx) function:

The preSaveRules(ctx) function allows for extra processing that is related to the transaction as a whole. You can use the preSaveRules(ctx) function to create a related object and add it to the transaction. The function is called before the Save action and is called for each noun in the message.

Change status or status date by using the changeStatus(ctx) function:

The changeStatus(ctx) function implements a status change outside the statefulMicSetIn class that is available in the integration framework. You can also use this function to set the status date with a value instead of using the system date.

For the automation script to use the changeStatus(ctx) function, the statefulMicSetin class or the class that extends the statefulMicSetin class must be registered as the processing class in the object structure. You can use this function to support a status change for five parameters. The base class supports only three parameters.

The following example sets the memo field that is related to the change status action to a string value if the **NP_STATUSMEMO** field does not contain a value:

```
importPackage(Packages.psdi.server);
importPackage(Packages.psdi.mbo);

function changeStatus(ctx)
{
     var mbo = ctx.getMbo();
     var struc = ctx.getData();
     var stat = struc.getCurrentData("STATUS");
     var memo = struc.getCurrentData("NP_STATUSMEMO");
      if(struc.isCurrentDataNull("NP_STATUSMEMO"))
     {
          memo = "Status change via Integration";
     }
      mbo.changeStatus(stat, MXServer.getMXServer().getDate(), memo, MboCon-stants.NOACCESSCHECK);
}
```

Context APIs for the inbound processing of object structures:

A *context* (ctx) is an object that provides convenience methods for automation scripts to pass data or trigger actions. The context is supported bidirectionally. Predefined APIs are available for the context, and each function can use different contexts.

An automation script requires context methods to implement processing logic. A context is passed between the processing of the object structure and the automation script, which implements the custom code. The following table shows the context APIs and the common usage of the APIs for each of the functions:

Table 28. Context APIs as used by the functions

Context APIs	beforeProcess (ctx)	beforeCreateMBOSet (ctx)	afterCreateMboSet (ctx)	mboRules (ctx)	beforeMboData (ctx)	afterMboData (ctx)	preSaveRules (ctx)	changeStatus (ctx)
ctx.setMsgType()	✓							
ctx.getProcessTable()		✓	✓	✓	✓	✓		
ctx.setProcessTable(String ProcessTable)	✓							
ctx.setMboSet(MboSet mboSet)		✓						
ctx.setMbo(Mbo mbo)		✓						
ctx.getData()	✓	✓	✓	✓	✓	✓	✓	✓
ctx.getMosDetailInfo()		✓	✓	✓	✓	✓	✓	✓
ctx.setSkipBaseAdditionalRules()							✓	
ctx.skipMbo()			✓					
ctx.skipTxn()	✓	✓	✓	✓	✓	✓	✓	✓
ctx.complete()			✓					
ctx.process()			✓					
ctx.getUserInfo()	✓	✓	✓	✓	✓	✓	✓	✓
ctx.getParentMbo()		✓	✓	✓	✓	✓		
ctx.isPrimary()		✓	✓	✓	✓	✓	✓	✓
ctx.bypassMbo()					✓			
ctx.getMsgType()	✓	✓	✓	✓	✓	✓	✓	✓
ctx.getPrimaryMboSet()			✓	✓	✓	✓	✓	✓
ctx.getPrimaryMbo()					✓	✓	✓	✓
ctx.processAsUpdate()				✓				
ctx.processAsAdd()				✓				
ctx.processAsAddAtEnd()				✓				
ctx.log ()	✓	✓	✓	✓	✓	✓	✓	✓

The following APIs are available for inbound data processing:

ctx.getMosDetailInfo()
Provides information about the data dictionary cache for the integration framework for the object structure that is being processed.

ctx.skipMbo()
Skips the processing of a Maximo business object.

ctx.skipTxn()
Skips the processing of an entire transaction, such as a message.

ctx.complete()
Ends the processing of the object structure at the point of execution, which means that child data is not processed.

ctx.process()
Continues the processing at the point of execution. This API is typically used as part of conditional logic.

ctx.getParentMbo()
Retrieves the parent business object. This API can be used from a child object whose business object is not yet created.

ctx.isPrimary()
Identifies whether the current business object is the root-level business object of the object structure.

ctx.getUserInfo()
Retrieves user information, which is needed if a script is creating a new business object.

ctx.bypassMbo()
Bypasses the creation of the business object and continues to the next business object that is being processed.

ctx.getMsgType()
Provides access to the message type, such as Sync or Create.

ctx.setMsgType()
Sets the message type, such as Sync or Create. This API can be used before processing begins.

ctx.getData()
Provides access to the StructureData, which is the XML message.

ctx.setMboSet(MboSet mboSet)
Sets values in a business object set.

ctx.setMbo(Mbo mbo)
Sets values in a business object.

ctx. getPrimaryMboSet()
Retrieves the root business object of an object structure during the processing of a child business object.

ctx.setProcessTable(String processTable)
Sets the table for a process. The method is used if you have a nonpersistent business object, such as MXRECEIPT, that has processing logic to determine whether the MATRECTRANS table or the SERVRECTRANS table is updated. The ctx.setProcessTable(String processTable) API is used in the beforeProcess(ctx) function.

ctx.getProcessTable()
Retrieves the table name that is set by the ctx.setProcessTable(String processTable) API.

ctx.setSkipBaseAdditionalRules()
Adds custom logic to in the preSaveRules(ctx) function to operate on a completed object structure that is now prepared in the processing.

ctx.processAsUpdate()
Sets the processing action of a business object to the Update action instead of the Add or Delete action.

ctx.processAsAdd()
Sets the processing action of a business object to the Add action instead of the Update or Delete action.

ctx.processAsAddAtEnd()
Sets the processing action of a business object to the Add action instead of the Update or Delete action and creates the business object at the end of the collection instead of at the top of the collection.

ctx.log ()
Runs a log statement from an object structure processing context.

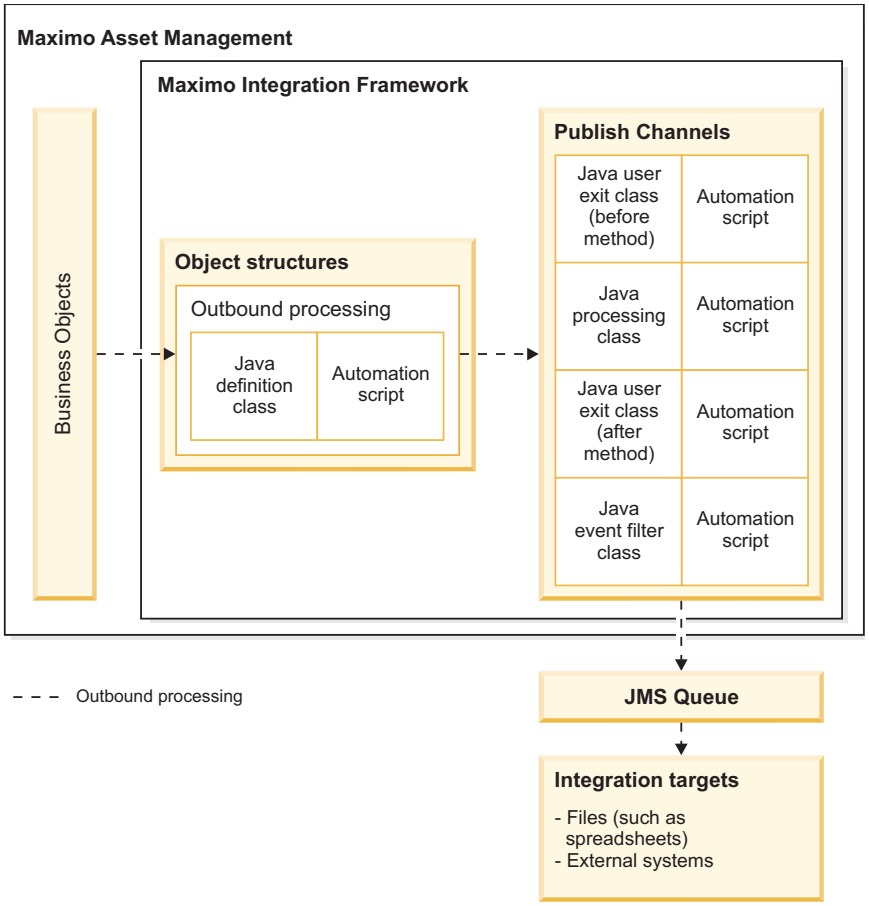
Customization of channel and service processing by using automation scripts:

When you create an automation script, you identify the channel or service that the script runs on. You also specify where to insert the script during the processing flow.

Customization points during publish channel processing:

Publish channels process outbound integration messages that do not require a response. The Java processing class that is associated with a publish channel includes several hooks where you can insert code to provide custom logic.

The illustration shows the customization points during outbound processing of integration messages by using publish channels.



After object structure processing constructs the integration message, it is forwarded to the associated publish channel. You can add custom logic at the following points during publish channel processing:

- External exit class
- User exit class that runs before the external exit class
- User exit class that runs after the external exit class
- Event filter class

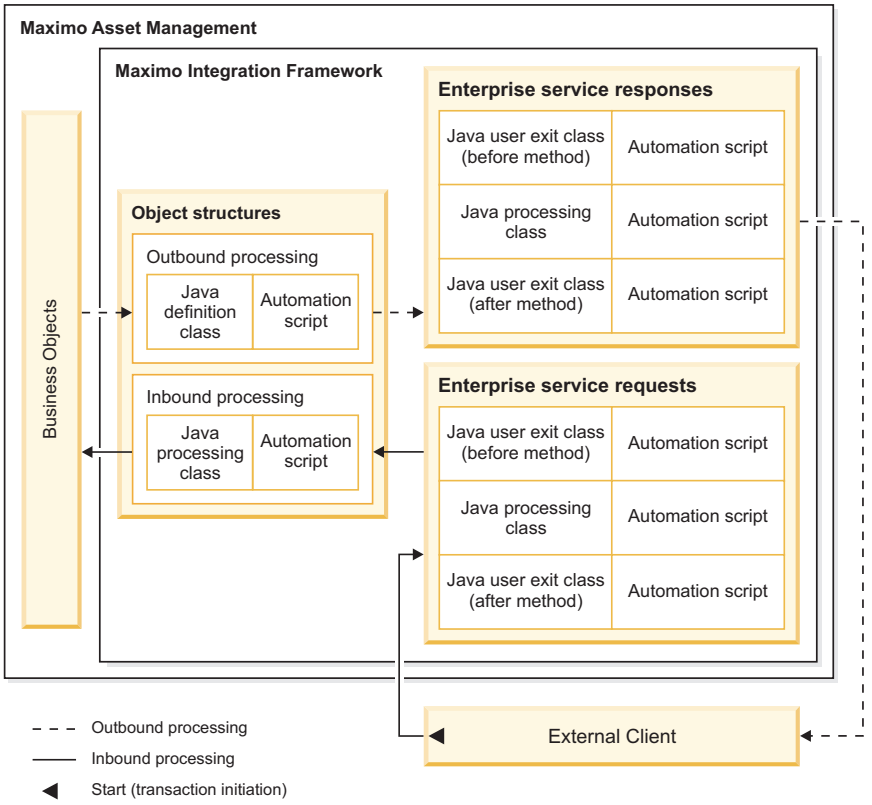
The message is then sent to the JMS queue for delivery to the target destination.

When you configure an external exit or user exit script for a publish channel, a predefined Java class is inserted on the channel that is used to run the script. The predefined classes are the com.ibm.tivoli.maximo.script.ScriptExternalExit class and the com.ibm.tivoli.maximo.script.ScriptUserExit class. You cannot implement a script and a Java class at the same processing point. If you attempt to create a script on a processing point where a Java class is configured, you cannot save the script.

Customization points during enterprise service processing:

Enterprise services provide asynchronous and synchronous processing of inbound integration messages. You can insert code to provide custom logic to the Java request processing class for inbound processing and to the Java response processing class for outbound processing.

The illustration shows the customization points during the inbound and outbound processing of integration messages by using enterprise services.



For asynchronous messages that do not require a response, an external service opens a connection to send an enterprise service request. When the request is validated, the message is dropped into a JMS queue for enterprise services processing. For synchronous messages that require a response, the external service maintains a continuous connection during the transaction until the response is received.

In the Automation Scripts application, when you configure an automation script for an enterprise service, you specify one of the following insertion points for the script:

- Insertion points for inbound asynchronous processing:

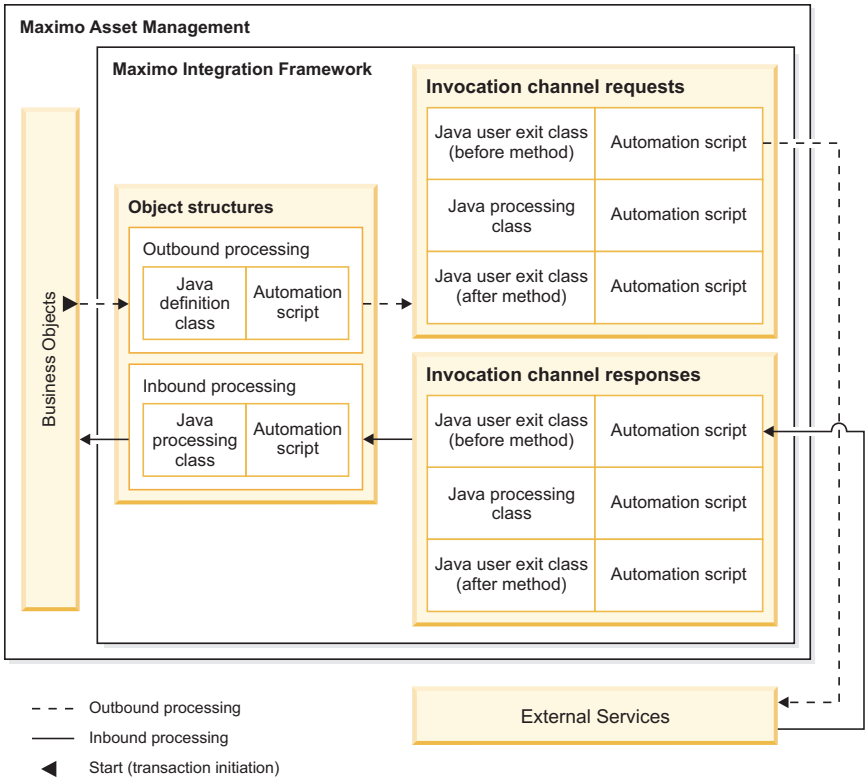
  - Request, inbound: User exit class that runs before the external exit class
  - Request, inbound: External exit class
  - Request, inbound: User exit class that runs after the external class
- Insertion points for inbound synchronous processing, where a response is required:
  - Request, inbound: User exit class that runs before the external exit class
  - Request, inbound: External exit class
  - Request, inbound: User exit class that runs after the external class
  - Response, outbound: User exit class that runs before the external exit class
  - Response, outbound: External exit class
  - Response, outbound: User exit class that runs after the external class

When you configure an external exit or user exit script for the request or the response of an external service, a predefined Java class is inserted that runs the script. The predefined classes are the com.ibm.tivoli.maximo.script.ScriptExternalExit class and the com.ibm.tivoli.maximo.script.ScriptUserExit class. You cannot implement a script and a Java class at the same processing point. If you attempt to create a script on a processing point where a Java class is configured, you cannot save the script.

Customization points during invocation channel processing:

Invocation channels process outbound integration messages that require a response. The invocation channel request class and the invocation channel response class include several hooks where you can insert code to provide custom logic.

The illustration shows the customization points during inbound and outbound processing of integration messages by using invocation channels.



When a user action in Maximo Asset Management starts an outbound integration transaction that requires a response, an invocation channel provides integration processing of requests and responses.

In the Automation Scripts application, when you configure an automation script for an invocation channel, you specify one of the following insertion points for the script:

- Request, outbound: User exit class that runs before the external exit class
- Request, outbound: External exit class
- Request, outbound: User exit class that runs after the external exit class
- Response, inbound: User exit class that runs before the external exit class
- Response, inbound: External exit class
- Response, inbound: User exit class that runs after the external exit class

When you configure an external exit or user exit script for the request or the response of an invocation channel, a predefined Java class is inserted that runs the script. The predefined classes are the com.ibm.tivoli.maximo.script.ScriptExternalExit class and the com.ibm.tivoli.maximo.script.ScriptUserExit class. You cannot implement a script and a Java class at the same processing point. If you attempt to create a script on a processing point where a Java class is configured, you cannot save the script.

Examples of using automation scripts during processing by channels and services:

Channel and service processing takes a data record from the source system as an input value, manipulates the data as required, and constructs a data record for the target system. The example python scripts query input data and provide logic to manipulate the data before the output data is constructed.

When Maximo Asset Management starts an integration transaction, the object structure provides an internal record data (irData) element to a publish channel or an invocation channel. Processing manipulates the irData element and constructs an external record data (erData) element before the message is forwarded to its destination. Invocation channel transactions can require a response from the external system. Response processing passes the erData element to the invocation channel, which manipulates the response data and constructs the irData element.

When an external system starts an integration transaction, the message provides an erData element to an enterprise service. Processing manipulates the erData element and constructs an irData element before the message is forwarded to the associated object structure. For messages that require a response, the object structure provides the irData element to the enterprise service. Processing manipulates the data and constructs the erData element that is forwarded to its destination.

The examples consist of simple scripts that can be used for test purposes. You can use the data import and data export features in the External Systems application to start transactions to test scripts.

Example: Script that changes the description of operating assets on outbound transactions

In this scenario, the MXASSET object structure provides irData to the MYASSET publish channel for processing. An automation script is configured to run on the external exit class of the publish channel. The script checks the status of the asset in the irData element. If the asset is in operating status, the script inserts a value in the description field and prints a message to the log file. The erData element is then constructed and is forwarded to the external system.

```
if irData.getCurrentData("STATUS") == 'OPERATING' :
   irData.setCurrentData("DESCRIPTION","hello")
   print "MYASSET description change"
```

Example: Script that changes the description of operating assets on inbound transactions

In this example, the MYASSET enterprise service processes an inbound message for the MXASSET object structure. An automation script inserts a script on the external exit class of the enterprise service. The script checks the status of the asset in the erData element. If the asset is in operating status, the script inserts a value in the description field and prints a message to the log file. The irData element is then constructed and is forwarded to the associated object structure for processing.

```
if erData.getCurrentData("STATUS") == 'OPERATING':
   erData.setCurrentData("DESCRIPTION","hello inbound")
   print "MYASSET inbound description has changed"
```

Example: Script that changes the description of lines on a purchase order by using an automation script variable

In this example, the MXPO object structure provides the irData element to the MYPO publish channel for processing. An automation script is configured to run on the external exit class of the publish channel. A literal value, world, is defined as an input variable on the automation script. The script queries the irData element for purchase orders that contain purchase order lines. The script numbers each purchase order line in a sequence that starts at 1, sets hello as the value in the

description field, and adds the input variable from the automation script. The script updates the parent purchase order before the erData element is constructed and forwarded to the external system.

```
lines = irData.getChildrenData("POLINE")
i = 0
if lines is not None:
   for value in lines:
      ++i
      irData.setAsCurrent(lines,i);
      irData.setCurrentData("DESCRIPTION","hello"+world)
irData.setParentAsCurrent()
```

Example: Script that skips transactions based on the status of records

In this example, the MXPO object structure is sent to the MYPO2 publish channel for processing. An automation script is configured to run on the user exit class before the external exit class runs. The script queries the irData element for purchase orders with a status of WAPPR and skips the processing of these purchase orders.

```
if irData.getCurrentData("STATUS") == 'WAPPR' :
   errorgroup = "iface"
   errorkey ="SKIP_TRANSACTION"
```

Example: Script that prints transaction information to a log file to assist with troubleshooting

In this example, the MXASSET object structure provides irData to the MYASSET2 publish channel for processing. An automation script is configured to run on the external exit class. The script queries the irData element for assets with a status of operating, prints transaction information to a log file, and constructs the erData element without making any changes. To print information about the transaction, you must set logging to debug. In the Logging application, the logger is set to automation scripts and the logging level is set to DEBUG. In the Automation Scripts application, the logging level for the script is set to DEBUG.

```
if irData.getCurrentData("STATUS") == 'OPERATING' :
   print "Test script
      variable VAR_EXIT_IFACETYPE  " + ifaceTypeprint "Test script
      variable VAR_EXIT_IFACENAME  " + ifaceNameprint "Test script
      variable VAR_EXIT_EXTSYSTEM  " + extSystemprint "Test script
      variable VAR_EXIT_MESSAGETYPE  " + messageTypeprint "Test script
      variable VAR_EXIT_EXTSYSTEM  " + extSystemprint "Test script
      variable VAR_EXIT_OSNAME  " + osNameprint "Test script
```

When an asset is exported, the following debugging information is printed to a log file:

```
18 Mar 2014 11:35:06:877 [DEBUG] [MXServer] [CID-MXSCRIPT-2022] execution completed
for cached compiled script PUBLISH.MYASSET.EXTEXIT.OUT for launch point null
Test script variable VAR_EXIT_IFACETYPE MAXIMO
Test script variable VAR_EXIT_IFACENAME MYASSET2
Test script variable VAR_EXIT_EXTSYSTEM MYEXTSYS
Test script variable VAR_EXIT_MESSAGETYPE Publish
Test script variable VAR_EXIT_OSNAME MXASSET
```

XSL mapping

For outbound transactions, you can implement an XSL file to manipulate the data that is sent to the external system after Java exit processing is completed. For inbound transactions, you can implement an XSL file to manipulate the data to be set to the object structure after Java exit processing is completed.

The XSL file is always called with the XML message that is output from the Java exit processing. You can register the XSL file in the application EAR file under the `businessobjects/classes/` directory or you can reference it using a directory filepath that is not part of the application EAR file.

Omit the `.xsl` file extension to register the file in the `businessobjects/classes/psd/iface/xsl` directory, for example:

```
psdi.iface.xsl.mapping
```

Include the `.xsl` file extension to register the file in a file directory that is accessible by the application server, for example:

```
c:/psdi/iface/xsl/mapping.xsl
```

Interface table user exit class

When you use interface tables to receive messages from an external system, you can perform customization in the polling program that retrieves the data from the interface table and sends data to the system.

The cron task manager runs the interface table polling program. The IFACETABLECONSUMER cron task, has an optional EXITCLASS property where you can specify the fully qualified name of a Java exit class.

The Java data structure list represents the record from the interface tables, where the first element is always the action of the message. The remaining elements of the list are the mapped data structures and each map represents a row in the interface table for each message. The keys in the map are the column names and the values are the corresponding column values. All the column values are converted to their translated string format before they are set in the map.

The EXITCLASS class must implement the `psdi.iface.intertables.IfaceTbExit` interface and the following methods:

- `public void beforeQueue(long transid, String extSys, String ifaceName, List data, Connection conn)`

  This method is called after the data is received from the interface table and before the data is inserted into one of the inbound queues.
- `public void afterCommit(long transid, String extSys, String ifaceName, Connection conn)`

  This method is called after the data is inserted to an inbound queue and deleted from the interface queue table, and the database commit is done.
- `public void afterRollback(long transid, String extSys, String ifaceName, Connection conn)`

  This method is similar to the `afterCommit` method but is called if the transaction is rolled back.

This class can perform the following processes:

- Validate data.
- Change external data by changing the IR record to be saved in the system.
- Stop the transaction from being saved in the queue by throwing an exception. In this case, the transaction remains in the MXIN_INTER_TRANS table with the error message and is reprocessed.
- Stop the message from being sent to external system by throwing a skip_transaction exception. In this case, message the system does not save the message; the message is removed from the queue.

- Log the transaction

The user exit that uses the `afterCommit` or `afterRollback` method can perform the following processes:

- Perform custom processing and cleanup.
- Log the transaction.

You identify this class in the Cron Task Setup application.

Configuring the integration framework

Setting up the integration framework includes configuring related system properties, JMS queues, and security. Implementing integration scenarios requires a knowledge of configuring JMS queues on the application server and knowledge of J2EE and product security support.

Integration system properties

System properties define the behavior and characteristics of the integration framework. To review or change integration framework properties, filter for the properties in the System Properties application.

General integration properties

To see a list of general integration properties, specify `mxe.int` as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 29. General integration properties

Property	Description	Default value
mxe.int.containerdeploy	Deploy web services to the application server container. When set to 0 (false), web services are deployed to the product container.	0
mxe.int.credentialmapperclassname	Credential mapper classname is a class file that can be used for mapping credential information when an integration module is implemented.	No default value
mxe.int.genboolasbool	Generate Boolean as schema Boolean.	1
mxe.int.globaldir	Specifies the location of a global directory which stores files that are related to integration	No default value
mxe.int.queueusercachesize	Number of users that are cached for inbound queue messages.	10
mxe.int.resolveschema	Resolves all schema includes to contain inline schema definition.	1
mxe.int.servicedeployer	Web services deployer class is a custom Java class for web service deployment when the default deployer class is not used.	No default value
mxe.int.uddiinqurl	Represents the integration UDDI registry inquiry URL.	No default value
mxe.int.uddiname	Represents the integration UDDI registry user ID.	No default value
mxe.int.uddipassword	Integration UDDI registry password.	No default value
mxe.int.uddipuburl	Integration UDDI registry publish URL.	No default value

Table 29. General integration properties (continued)

Property	Description	Default value
mxe.int.validatedbupdates	Validates the database updates completed by integration. When set to 1 (true), the deletion of business objects, attributes, indexes, and relationships by a user through the Database configuration application are validated against integration content. The validation ensures that the data that is deleted is not referenced by an integration component. If a reference exists, the user is not able to complete the delete action.	1
mxe.int.validatemmpackage	Validates the Migration Manager database updates by integration.	0
mxe.int.verifywebappurl	Verifies web application URL when schema files are generating.	1
mxe.int.webappurl	Represents the integration web application URL. Configure this property to contain the correct host name and port number.	http://localhost/meaweb
mxe.int.wsdlcurrentschema	Shows the current schema definition in WSDL.	1
mxe.int.wsdlincludesschema	Includes the schema directly in the WSDL.	1
mxe.int.wsdlnamespace	Represents the integration WSDL namespace.	http://www.ibm.com/maximo/wsdl
mxe.int.xmlnamespace	Represents the integration XML namespace.	http://www.ibm.com/maximo
mxe.int.binarytext	Converts a text value to base 64 encoded value.	0
mxe.int.defaultaction	The default action for flat file import.	AddChange
mxe.int.defaultoperation	The default operation for the application export.	Sync
mxe.int.dfltuser	Represents the Integration default login user.	mxintadm
mxe.int.doclink.maxfilesize	Represents the maximum file size (MB) for attachments that are included as part of an integration message.	10
mxe.int.enabledatemillis	Enables the dates with milliseconds part.	0
mxe.int.expupdatesender	Updates the **SENDERSYSID** field on the primary object during data export.	0
mxe.int.extracttrycount	The File Extract Retry Count is the number of times an error message is retried during data import when using file-based error management.	0
mxe.int.flatfiledelimiter	Integration flat file text delimiter is the default delimiter value that is used for application import enablement and for data import.	,
mxe.int.flatfilenewline	Retains new line character in flat files. For fields, such as descriptions, that can contain new line characters, the characters are retained in the integration messages when the property value is 1 (true).	0
mxe.int.interactiveimport	Performs the application import as interactive.	0
mxe.int.keyresponse	Provides response content for inbound integration messages for all operations. When set to 1 (true), response content, that includes the primary object key values, is provided for all service operations. When set to 0 (false), response content is provided for Query and Create operations only.	1
mxe.int.maxextractdocs	Represents the number of error documents that are written to each temporary file when an extract file is building .	1000
mxe.int.mdbdelay	Represents the wait time in milliseconds before a message from the error queue is processed.	-1

Table 29. General integration properties (continued)

Property	Description	Default value
mxe.int.propagateuser	Propagate authenticated user through the inbound queue. When set to 1 (true), the user of the integration message is saved with the queue message and used during the processing of the message because it is processed from the queue to the business objects.	0
mxe.int.savemessage	Indicates the save JMS message.	0
mxe.int.setclobasaln	Controls the truncation of characters that are sent to interface tables.	0
mxe.int.textqualifier	The flat file text qualifier is the default text qualifier value in application import enablement and in data import.	"
mxe.int.updatecoafromglcomp	Updates the Chart of Accounts that contain an identified component. When set to 1 (true), processing of inbound GL component data initiates related updates to any chart of account data that references the GL component.	1
mxe.int.usescientific	Uses scientific notation for double values.	1
mxe.int.validatexmltext	Validates XML element value for invalid XML characters. When set to 1 (true), an outbound message is validated to ensure that all data in the message uses valid XML characters. If messages contains invalid characters, the operation stops and no outbound message is delivered.	0
mxe.int.whereclausepolicy	Sets the where-clause policy for an integration query.	parse
mxe.int.adminfromemail	The email address FROM integration administration, which is used as the From email address when integration initiates an email. Must be a valid email address format, such as from@example.com.	No default value
mxe.int.admintoemail	The email address TO integration administration, which is used as the To email address when integration initiates an email. Must be a valid email address format, such as to@example.com. You can provide more than one email address in a comma-separated list.	No default value
mxe.int.usedbinfoforifacetb	Sets the processing for interface tables. When set to 1, the interface process refers to the existing table to determine what to insert into the table rather than the object structure definition.	0
mxe.int.enableosauth	Enables the authorization configuration of object structures.	1

REST integration properties

To see a list of REST API integration properties, specify `mxe.rest` as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 30. REST API integration properties

Property	Description	Default value
mxe.rest.format.json.mimetypes	The REST supported mime types for JSON.	application/json
mxe.rest.format.xml.mimetypes	The REST supported mime types for JSON.	application/xml,text/xml
mxe.rest.handler.mbo	The REST MBO resource handler.	com.ibm.tivoli.maximo.rest. MboResourceRequestHandler

Table 30. REST API integration properties (continued)

Property	Description	Default value
mxe.rest.handler.os	The REST object structure resource handler.	com.ibm.tivoli.maximo.rest.OSResourceRequestHandler
mxe.rest.handler.ss	The REST standard service resource handler.	com.ibm.tivoli.maximo.rest.MaxServiceResourceRequestHandler
mxe.rest.serializer.mbo.imglib.image	The REST serializer for the imagelib MBO for image format.	com.ibm.tivoli.maximo.rest.ImageLibSerializer
mxe.rest.serializer.mbo.json	The REST serializer for MBO for JSON format.	com.ibm.tivoli.maximo.rest.MboJSONSerializer
mxe.rest.serializer.mbo.xml	The REST serializer for MBO for xml format.	com.ibm.tivoli.maximo.rest.MboXMLSerializer
mxe.rest.serializer.os.json	The REST serializer for object structures for JSON format.	com.ibm.tivoli.maximo.rest.OSJSONSerializer
mxe.rest.serializer.os.xml	The REST serializer for object structures for xml formats.	com.ibm.tivoli.maximo.rest.OSXMLSerializer
mxe.rest.serializer.ss.json	The REST serializer for standard services for JSON format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseJSONSerializer
mxe.rest.serializer.ss.xml	The REST serializer for standard services for xml format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseXMLSerializer
mxe.rest.webappurl	Token Authentication on Web Application URL.	No default value
mxe.rest.mbo.blockaccess	Blocks access to the comma-separated list of MBOs.	No default value
mxe.rest.mbo.defaultformat	The REST default format for all MBOs.	xml
mxe.rest.mbo.imglib.defaultformat	The REST default format for the MBO imglib.	image
mxe.rest.os.blockaccess	Blocks access to the separated list of object structures.	10
mxe.rest.os.defaultformat	The REST default format for all object structures.	xml
mxe.rest.ss.defaultformat	The REST default format for all standard service response	xml
mxe.rest.supportedformats	The REST supported formats for a response.	xmljsonimage
mxe.rest.whereclausepolicy	Sets the where clause policy for REST query.	parse

OSLC integration properties

To see a list of OSLC integration properties, specify mxe.oslc as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 31. OSLC integration properties

Property	Description	Default value
mxe.oslc.dfltconsumerversion	The default OSLC version that the consumer uses.	2
mxe.oslc.dfltversion	The default OSLC version for an OSLC provider.	2
mxe.oslc.enableprovider	Enables the OSLC provider.	1
mxe.oslc.idleexpiry	Indicates the idle expiry time.	300
mxe.oslc.webappurl	The provider's public URL.	http://localhost/maximo/oslc/

Table 31. OSLC integration properties (continued)

Property	Description	Default value
mxe.oslc.collectioncount	Adds the total count in the OSLC collection.	0
mxe.oslc.defaultep	The default OSLC Endpoint.	OSLCDEFAULT
mxe.oslc.defaultformat	The default format for OSLC.	oslcjson
mxe.oslc.errorresponse	The OSLC Error Response Format.	1
mxe.oslc.preferproviderdesc	Prefers OSLC provider description for resource registry reconciled URLs	false
mxe.oslc.prefersmallpreview	Prefers small preview for OSLC consumer.	false
mxe.oslc.prettyjson	Pretty printed JSON.	0
mxe.oslc.prettyrdf	Pretty printed RDF.	0
mxe.oslc.prqueryep	The Provider Registry Query Endpoint.	PROVIDERREGISTRY
mxe.oslc.prcreateep	Represents the Provider Registry Create Endpoint.	No default value

JMS queue configuration

Asynchronous transactions that are exchanged using either publish channels or enterprise services, use Java Message Service (JMS) queues to exchange data with an external system.

For inbound processing, when an enterprise service message is received, the message is immediately written to a JMS queue and the caller of the service is released from the transaction. The message is processed from the inbound JMS queue, through the application business objects, and saved to the database. Messages remain in an inbound queue until they are successfully processed or until they are deleted from the queue. A common strategy for inbound queue implementation is to isolate the queues and the queue consumers to a separate server, or server cluster. This strategy ensures that inbound message-processing does not have a performance impact on application users.

For outbound processing, messages sent out using a publish channel are written to a JMS queue and the user who initiated the message is released from the transaction. The message is processed from the outbound JMS queue using the configured endpoint, and is delivered to the external application. Messages remain in the outbound queue until they are successfully delivered to the external application or are deleted from the queue.

There are three default message queues:

- One outbound sequential queue
- One inbound sequential queue
- One inbound continuous queue

A JMS queue implementation can operate on a single application server or across a cluster of application servers.

Creating and configuring a queue

You can choose to use a single JMS queue for multiple external systems or create a separate queue for each external system that you use. Queue creation and configuration involves multiple steps. You can use the default queues or create additional queues, depending on your integration requirements.

About this task

Separate queues are used to support outbound transactions and inbound transactions. Configure a queue to support either inbound or outbound transactions, not both. Configure outbound queues for use with publish channels and configure inbound queues for use with enterprise services.

Procedure

1. Create and configure the message queue on the application server. JMS queues can be configured automatically or manually on the WebSphere Application Server. JMS queues must be manually configured on the WebLogic Server.
2. In the External Systems application, add properties to the queue. You can create additional queues to meet system needs. If you do not use the default queues, use an application server provider for your queue configuration.
3. In the External Systems application, configure the external system and enterprise services to use the queues.

Queue properties:

In the External Systems application, you can configure several properties for each JMS queue.

Property	Description
Queue JNDI Name	References the Java Naming and Directory Interface (JNDI) name that is configured on the application server. A default value is provided.
Queue Connection Factory Name	References the Connection Factory name that is configured on the application server. A default value is provided.
Initial Context Factory	A value that you must configure when you do not to use the default queues and do not use a provided application server.
Provider URL	A value that you must configure when you do not to use the default queues and do not use a provided application server.
User ID	The user ID that you configure when the queue is secured on the application server.
Password	The password that is configured when the queue is secured in the application server for the user ID.
Inbound	Identifies whether the queue is used for inbound processing. If the inbound value is null, the system uses the queue for outbound processing.
Sequential	Identifies whether the queue is a sequential queue. If the sequential value is null, the system uses the queue for continuous queue processing.

Property	Description
Compress	Identifies whether the messages are compressed when they are written to the queue and decompressed when they are pulled from the queue. Compression provides significantly reduced message sizes. The standard Java Inflater and Deflater APIs (java.util.zip) are used for compression.
Maximum Try Count	Identifies how many times the integration framework attempts to reprocess a message after it encounters an error. The system continues to retry the message until the count value is met. The value of this property must be set to zero when an error queue is implemented.

Sequential queues

The sequential queue is a JMS queue that uses a predefined system cron task to consume messages. Messages in sequential queues are processed on a strict first-in-first-out basis, ensuring that messages are processed in the order that they are generated and received.

When a message results in an error, the system generates an error that can be managed in the Message Reprocessing application, and does not process subsequent messages in the queue until the error is cleared.

You can configure two system sequential queues for inbound and outbound message processing. A predefined cron task, JMSQSEQCONSUMER, polls the queues. There are two instances of the task, one that polls the inbound queue and one that polls the outbound queue. If you create additional sequential queues, you can configure additional instances of the cron task to point to the additional queues.

The following table lists the cron task parameters that you can configure.

Parameter	Description
MESSAGEPROCESSOR	Java class that processes the messages from the queue. The system provides this class.
QUEUENAME	Queue JNDI name, when the queue is created on the application server.
SELECTOR	The WHERE clause for configuring an instance of the cron task to process a subset of messages in the queue. This parameter is optional.
TARGETENABLED	Ensure that the value is at the default of 0 (false). The functionality of this flag is superseded by the donotrun functionality. Use the donotrun parameter in the cron task framework to control which servers the cron task runs on.

Continuous queues

A continuous queue is a JMS queue with a message-driven bean (MDB) as a consumer. A continuous queue is predefined for enterprise services only and uses

multi-threaded processing to provide better system performance. A continuous queue does not guarantee the processing order of the messages as is the case with a sequential queue.

When message processing results in an error that can be managed in the message reprocessing application, the system generates an error message and then continues processing subsequent messages in the queue. There is one default continuous queue to process inbound messages. You can choose to implement additional continuous queues, depending on your integration requirements.

Enabling message beans:

Message beans on the application server act as the consumer of messages from a continuous queue. To enable message beans to support the continuous queue, you must uncomment lines in the XML deployment files on the application server.

Procedure

1. For both WebSphere Application Server and WebLogic Server environments, uncomment the following lines of code in the `ejb-jar.xml` file located in the `...\applications\maximo\mboejb\ejbmodule\META-INF\` folder:

```
<!-- MEA ejb for MDB
<message-driven id="MessageDriven_JMSContQueueProcessor_1">
 <ejb-name>JMSContQueueProcessor-1</ejb-name>
 <ejb-class>psdi.iface.jms.JMSContQueueProcessor</ejb-class>
 <transaction-type>Container</transaction-type>
 <message-destination-type>javax.jms.Queue</message-destination-type>
 <env-entry>
  <env-entry-name>MESSAGEPROCESSOR</env-entry-name>
  <env-entry-type>java.lang.String </env-entry-type>
  <env-entry-value>psdi.iface.jms.QueueToMaximoProcessor</env-entry-value>
 </env-entry>
</message-driven>

-->
<!-- MEA ejb for MDB
 <container-transaction>
  <method>
   <ejb-name>JMSContQueueProcessor-1</ejb-name>
   <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
 </container-transaction>
-->
<!-- Notf MDB
     <message-driven id="MessageDriven_JMSNotificationProcessor_1">
       <ejb-name>JMSNotificationProcessor-1</ejb-name>
       <ejb-class>psdi.iface.jms.JMSContQueueProcessor</ejb-class>
       <transaction-type>Container</transaction-type>
       <message-destination-type>javax.jms.Queue</message-destination-type>
       <env-entry>
         <env-entry-name>MESSAGEPROCESSOR</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>com.ibm.tivoli.maximo.notification.
              NotificationMessageProcessor</env-entry-value>
       </env-entry>
     </message-driven>
-->
<!-- Notf MDB for error queue

    <message-driven id="MessageDriven_JMSNotificationProcessor_2">
      <ejb-name>JMSNotificationProcessor-2</ejb-name>
      <ejb-class>psdi.iface.jms.JMSContQueueProcessor</ejb-class>
      <transaction-type>Container</transaction-type>
```

```
      <message-destination-type>javax.jms.Queue</message-destination-type>
      <env-entry>
        <env-entry-name>MESSAGEPROCESSOR</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>com.ibm.tivoli.maximo.notification.
        NotificationMessageProcessor</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>MDBDELAY</env-entry-name>
 <env-entry-type>java.lang.Long</env-entry-type>
 <env-entry-value>30000</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>ERRORQUEUE</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>1</env-entry-value>
      </env-entry>

    </message-driven>
-->
```

2. In a WebSphere Application Server environment, uncomment the following lines in the ...\applications\maximo\mboejb\ejbmodule\META-INF\ibm-ejb-jar-bnd.xmi file:

```
<!-- MEA ejb for MDB
<ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
  xmi:id="MessageDrivenBeanBinding_1" activationSpecJndiName="intjmsact">
 <enterpriseBean xmi:type="ejb:MessageDriven"
    href="META-INF/ejb-jar.xml#MessageDriven_JMSContQueueProcessor_1"/>
</ejbBindings>-->

<!-- NOTF MDB
  <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
   xmi:id="MessageDrivenBeanBinding_3" activationSpecJndiName="notfact">
    <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-jar.xml
     #MessageDriven_JMSNotificationProcessor_1"/>
  </ejbBindings>
-->
<!-- NOTF MDB for error queue
  <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
   xmi:id="MessageDrivenBeanBinding_1" activationSpecJndiName="notfacterr">
    <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-jar.xml
     #MessageDriven_JMSNotificationProcessor_2"/>
  </ejbBindings>
-->
```

3. In a WebLogic Server environment, uncomment the following lines in the ...\applications\maximo\mboejb\ejbmodule\META-INF\weblogic-ejb-jar.xml file:

```
<!-- MEA MDB
 <weblogic-enterprise-bean>
  <ejb-name>JMSContQueueProcessor-1</ejb-name>
  <message-driven-descriptor>
   <pool>
    <max-beans-in-free-pool>3</max-beans-in-free-pool>
   </pool>
   <destination-jndi-name>jms/maximo/int/queues/cqin</destination-jndi-name>
   <connection-factory-jndi-name>jms/maximo/int/cf/intcf
      </ connection-factory-jndi-name>
  </message-driven-descriptor>
  <transaction-descriptor>
   <trans-timeout-seconds>600</trans-timeout-seconds>
  </transaction-descriptor>
 <jndi-name>JMSContQueueProcessor-1</jndi-name>
</weblogic-enterprise-bean>
-->
```

4. After you change the XML files, rebuild the EAR file and redeploy it to the application server for the changes to take effect.

Continuous queue performance:

To improve queue performance, you can increase the number of message-driven beans for a queue and introduce additional application servers in a cluster. Because message processing is multi-threaded, errors can occur because of the random order of processing.

The following examples describe errors that can occur. In both scenarios, the integration error management processing can successfully reprocess the error before the system administrator can review it.

For example, you are batch loading a large volume of item and inventory messages in the continuous queue, and multiple inventory records exist for the same item number. If an inventory message for Item A is processed before the item message that adds Item A to the system is processed, the inventory message produces an error because Item A does not exist. Processing continues with the next message.

Eventually, the item message for Item A is processed and Item A is added to the system. The failed message can then be successfully processed. In this case, the error is corrected without manual intervention.

This type of situation can occur when you load related messages in the continuous queue at the same time. Such a situation is more likely to occur when the volume of transactions is high but also can occur whenever two messages process related data concurrently.

For example, two messages try to update the same system record at the same time. One message succeeds and the other fails. However, the error management processing of the system must process the second message after the first update is completed.

Configuring message beans:

Server-specific extensions control the maximum number of beans that you can create.

Configuring message beans on WebSphere Application Server:

By default, the server is configured to have five message-driven beans.

Procedure

1. In the administrative console, select JMS activation specification.
2. Select **intjmsact**.
3. Specify a value in the **Maximum Concurrent End Points** field.

Configuring message beans on WebLogic Server:

By default, the WebLogic Server is configured to have three message-driven beans.

Procedure

1. Open the `weblogic-ejb-jar.xml` file in a text editor.
2. Search for the following lines of code:

```
<pool>
 <max-beans-in-free-pool>3</max-beans-in-free-pool>
</pool>
```

3. Replace the value 3 with a different value if required. Start with a relatively low number of message-driven beans and monitor performance.
4. Modify the file to increase the number of message-driven beans incrementally until you are satisfied with the processing performance of the messages in the continuous queue.

What to do next

If system performance is poor, you can resolve some system performance issues by clustering servers and isolating the inbound message processing to a specific server cluster.

Message caching:

Continuous queue processing uses the Maximum Batch Size property, under the Activation Specification definition, to control the number of messages received from the messaging engine in a single batch.

If three message-driven beans are enabled, and the batch size is 10, up to 30 messages can be cached. Test with different values to ensure that the value that you set does not impact application users or server processes.

An unlimited number of messages can be processed when you set the Maximum Batch Size value to -1 on a WebLogic Server environment.

If you plan to use a WebSphere Application Server error queue, use the default value for the batch size.

Configuring an error queue for the continuous queue:

You can implement an error queue on the application server that moves a message out of the main queue to a secondary queue when a message goes in error. Unless your integration scenario supports a very low volume of messages, configure an error queue to support message processing through the continuous queue.

A continuous queue uses MDBs to consume messages. When a message goes in error, the MDBs continue to consume the messages in error, even after the messages reach the maximum try count that is configured for processing. The continuous selection of these messages consumes system resources and can slow down, or even prevent, other messages from being processed. Configuring an error queue for the continuous queue helps to avoid performance delays or transaction bottlenecks by passing messages in error to a secondary queue.

Configuring an error queue on WebSphere Application Server:

You can configure a continuous queue to have a corresponding error queue. If a message encounters an error, it is moved out of the continuous queue to the error queue when the number of retries set in the maximum failed deliveries parameter is met. The continuous queue then processes new queue messages.

About this task

The system provider must perform this task on behalf of the tenant.

Procedure

1. Configure an error queue destination within the same bus member where the continuous queue resides.
2. Configure the continuous queue destination definition to have an exception destination. The exception destination must point to the error queue destination that you defined.
3. In the error queue, add an exception destination that points to itself. Errors in the error queue move from the top of the error queue to the bottom of the error queue. Messages in error are continuously retried.
4. Open the `ejb-jar.xml` file in a text editor to enable the MDBDELAY property. To avoid excessive use of system resources during message reprocessing, the MDBDELAY property delays the processing of messages.
5. Uncomment the following lines of code and set an appropriate value:

   ```
   <env-entry>
   <env-entry-name>MDBDELAY</env-entry-name>
   <env-entry-type>java.lang.Long </env-entry-type>
   <env-entry-value>30000</env-entry-value>
   </env-entry>
   ```

   The default value is 30 seconds (30000 milliseconds).
6. Open the `ibm-ejb-jar-bnd.xmi` file and uncomment the following lines of code to enable the message-driven beans on the error queue.

   ```
   <!-- MEA MDB for error queue
   <ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
     xmi:id="MessageDrivenBeanBinding_1" activationSpecJndiName="intjmsacterr">
   <enterpriseBean xmi:type="ejb:MessageDriven"
     href="META-INF/ejb-jar.xml#MessageDriven_JMSContQueueProcessor_2"/>
   </ejbBindings>
   -->
   ```

Continuous queue errors on WebLogic Server:

The WebLogic Server queue has a redelivery delay property that can control how messages in error are reprocessed. The redelivery delay property represents the time between when the message reports an error and when the message is reprocessed.

You cannot view the message in the queue for the amount of time you specified in the redelivery delay property. The redelivery delay improves system performance. Messages other than messages in error can be processed for the amount of time that is defined in the redelivery delay property. The processing delay applies to the message and not to the thread that processes the message.

If you set the batch size property to -1 (unlimited), and the redelivery delay property to 30 seconds (30,000 milliseconds), new messages can be processed in the queue. Processing continues even when a large number of errors are being reprocessed.

The same connection factory is used for both the sequential and continuous queues. To avoid sequential consumer processing issues, set the redelivery delay value in the destination queue configuration. Do not set the connection factory level configuration.

If the number of times that a message in error is processed exceeds the configured try count, the message stops processing and is redirected for error management.

Alternatively you can implement an error queue. To implement an error queue, you must uncomment entries for the error queue in the `ejb-jar.xml` and `weblogic-ejb-jar.xml` files.

Queue message format

Messages that are loaded into the JMS queues by the integration framework have defined components and formats The message body contains the XML message that is processed into the system or sent to the external system.

Message header

The message header can contain the JMS message ID and standard JMS header values.

Header	Description
JMSMessageID	A message ID that is generated by the system.
JMSRedelivered	Identifies whether the message was reprocessed.

Message properties

The properties contain the following properties from the JMS provider and the integration framework. The integration framework properties are of the string data type.

Property	Description
MEAMessageID	The message ID that is generated by the integration framework.
destjndiname	The name of the queue or topic that the message is written to.
INTERFACE	The name of the publish channel (outbound queue) and the enterprise service (inbound queue).
destination	The external system name for outbound messages.
SENDER	The external system name for the inbound messages.
USER	The name of the user that is associated with the inbound integration message. This value can be used for authorization security if required.
compressed	Indicates whether the message is compressed. Values can be true or false; the default value is false. The standard Java Inflater and Deflater APIs (`java.util.zip`) are used for compression.
uncompressed_length	Stores the original message payload size before compression. This value must comply with the int [xsd:int] schema type and is present only when the compressed property is set to true.

Property	Description
MSG_TRK_ENABLED	Internal value.
MSG_OP_MODE	The endpoint name for the outbound messages. Fixed string MXJMS for inbound messages.
MSG_TRK_STORE_MSG	Internal value.
MSG_TRK_EXTSYS	The external system name for outbound and inbound messages.
Msgkeyval	A field name message key.
searchfieldval	A comma-separated search field value.
msgoperation	Indicates whether the publish channel or enterprise service contains a sync, create, update, or delete operation.
msgstatus	Indicates whether the message has a RECEIVED, ERROR, DELETED, or a PROCESSED status value.
msgerrmsg	Contains the exception message text.

Messages in text format:

Messages that are written to a queue by the integration framework are in byte format by default. You can use the External Systems application to ensure that messages in text format are also supported.

To support messages in text format, select the **Text** check box in the Add/Modify queue window in the External Systems application. When a queue is configured to support messages in text format, all subsequent messages that are written to the queue by an integration component are in the *javax.jms.TextMessage* format instead of the default *javax.jms.BytesMessage* format.

When the queue is configured for messages in text format, the **Text Message Encoding** field in the External Systems application identifies the encoding of text-formatted messages in any of the inbound queues configured for the external system. When no value is provided, messages are assumed to be encoded in UTF-8. Messages that are written to the outbound queue are always encoded with UTF-8.

Queue selectors

Selectors act as WHERE clauses in the JMS queue consumer. Selectors can be applied to message headers and properties in either a continuous or a sequential queue.

The following table lists how you can use continuous selectors in the JMS queue consumer.

Type of queue	Where to identify selector
Sequential queue	Specified as a property of the cron task.
Continuous queue	Specified n the `ejb-jar.xml` code of the message-driven bean.

Applying selectors splits a queue into smaller queues, each of which contains a subset of data that each cron task or message-driven bean uses. An error in one subset of the data does not stop processing in the others in a sequential queue.

While selectors provide flexibility in separating the processing of transactions, they impair the performance of poll processing. Depending on the volume of transactions, you can prefer to implement multiple queues instead of one queue with multiple selectors. Multiple queues typically provide better performance.

You can add the following statement to the SELECTOR property of the SEQQIN instance of the JMSQSEQCONSUMER to instruct the cron task to process the MXPOInterface and MXPRInterface transactions from the corresponding external system:

```
SENDER='EXTSYS1' and INTERFACE in ('MXPOInterface', 'MXPRInterface')
```

Add the following content to the message bean configuration in the `ejb-jar.xml` file to instruct the message-driven bean to process the MXPOInterface and MXPRInterface transactions from the corresponding external system:

```
<message-selector>
  SENDER='EXTSYS1' AND INTERFACE IN ('MXPOInterface', 'MXPRInterface')
</message-selector>
```

If two external systems send data to an inbound sequential queue, an error in any record stops the processing of all transactions in that queue to maintain a first-in-first-out processing order. Create multiple instances of a cron task, each with a selector that processes a different external system, to prevent an error in one system from stopping transactions from the second system.

Ensure that the where clauses in the selectors identify the mutually exclusive sets of transactions in a sequential queue. Include all the transactions that are inserted into the queues to ensure that all messages are processed in a first-in-first-out order.

Viewing and deleting messages in a JMS queue

You can view a list of the messages in a JMS queues, download these messages to view their content, and you can delete messages from configured queues. When viewing or deleting messages, you can apply a selector to limit the messages returned for processing.

About this task

Messages that are currently being processed to a continuous queue by a message-driven bean or to a sequential queue by a JMS cron task are not available to view or delete. You can deactivate the cron task to stop the processing of messages to a sequential queue. Message-driven beans continually process messages and the number of messages being processed can vary. Unless you disable message-driven beans, it is likely that not all messages can be viewed or deleted. If you view or delete messages while message-driven beans are enabled, it is possible that not all messages are available for processing.

Procedure

1. In the External Systems application, select the **Add/Modify Queues** action .
2. Select a queue, and click either the **View Queue Data** button or the **Delete Queue Data** button.

3. Optional: If you select the **View Queue Data** option, you can specify a number in the **Count** field to limit the number of records to view and you can check the **Count Only** field if you only want to see how many messages are currently in the queue.
4. Optional: Specify one of the following selectors if you want to filter the records returned in either the View Queue Data window or in the Delete Queue Data window:

Option	Description
MEAMessageID	ID of the message (applies to inbound and outbound messages).
INTERFACE	The name of the enterprise service for inbound messages or the name of the publish channel for outbound messages.
destination	The name of the external system for outbound queues only.
SENDER	The name of the external system for inbound messages only.
USER	The user provided with the message. This value is optional, and applies to inbound messages only.

Selector values are case sensitive and you must enclose the value of the selector in single quotes, for example INTERFACE='MXPERSONInterface'.

Configuring queues with WebSphere MQ

You can use IBM WebSphere MQ to configure and manage queuing activities. The integration framework supports WebSphere MQ Version 6.

Related concepts:

"Messages in text format" on page 183
Messages that are written to a queue by the integration framework are in byte format by default. You can use the External Systems application to ensure that messages in text format are also supported.

Configuring JMS endpoints and handlers:

Outbound messages in a publish channel are placed into the default queue and you must configure the JMS endpoint and handler to send the message to WebSphere MQ.

Procedure

1. Create an WebSphere MQ provider on the WebSphere Application Server and configure an integration endpoint to point to the message queue (MQ) provider by using:
   a. The destination JNDI name (DESTJNDINAME)
   b. The connection factory JNDI name (CONFACTORYJNDINAME)
2. Configure the endpoint to point to the WebSphere MQ by using:
   a. The destination JNDI name (DESTJNDINAME)
   b. The connection factory JNDI name (CONFACTORYJNDINAME)
   c. The provider URL (PROVIDERURL)
   d. The initial context factory (CONTEXTFACTORY)

Configuring integration queues and WebSphere MQ provider:

To replace integration framework queues with WebSphere MQ queues, configure the queues on WebSphere Application Server and add these queue definitions in the External Systems application.

Procedure

1. Create the JMS queue by defining an alternate provider to replace the default provider.
2. Create a proxy queue on the WebSphere Application Server by using the WebSphere MQ provider that points to your message queue (MQ) server queue.
3. In the Add/Modify Queues dialog window in the External Systems application, add values to the **Queue JNDI Name** and **Connection Factory** fields to point to the proxy queue and connection factory.

Results

Outbound messages that are destined for the default queue are delivered to the message queue (MQ), and inbound messages are retrieved from the message queue (MQ).

Error management

The integration framework supports a variety of message formats, protocols for exchanging messages, and both synchronous and asynchronous message processing. The management of errors requires multiple options to meet the varied implementation configurations that you can choose.

JMS queues are used by the integration framework as a staging mechanism for inbound and outbound messages. Queue error management is initiated when an error condition is identified and you can view, correct, cancel, and reprocess problematic messages.

Non-queue error management

When you start the synchronous processing of inbound or outbound integration messages, you are notified of any errors at the time of execution.

Non-queue error management is necessary if an error occurs when a synchronous message is being processed. For inbound messages, instead of relying on an error queue, the integration framework responds synchronously to the caller of the process with an error message. The calling application must receive the response, correct the error, and retry the transaction.

Use the system log to troubleshoot synchronous transaction errors. The system log contains the processing exception that the integration framework issues to the caller of the process.

Queue-based error management

You can use the Message Reprocessing application to manage erroneous inbound and outbound asynchronous integration messages that use JMS queues.

Errors that occur when a message is sent from a queue to an external system are typically caused by a communication failure or a problem with database configuration when writing to interface tables or files. Errors that occur during

inbound processing are typically a result of business rule validations or the inbound processing logic of the integration framework.

The sequential queue processes messages one at a time, in a first-in-first-out sequence. When the integration framework encounters an error in processing a message in a sequential queue, inbound or outbound, the error management mechanism is initiated and the message is flagged as having an error. Subsequent messages in the queue are not processed until the message in error is resolved or deleted. As a result, only a single error can exist in a sequential queue.

The continuous queue handles inbound processing only and processes messages in a multi-threaded mode. When an error occurs in the continuous queue, error management is initiated and the message is flagged as having an error. The integration framework continues to process subsequent messages in the queue. As a result, multiple errors can exist in a continuous queue.

Depending on your system configuration, the integration framework makes several attempts to reprocess the message, for either type of queue, before determining that an error requires intervention. The system also performs the following activities when encountering an error:

- Sends a notification to a specified email account, informing the recipient that an error occurred. On an IBM WebSphere Application Server environment, the integration framework sends an additional email message to the specified email account each time that you successfully restart your application server.
- Creates a record in the Message Reprocessing application. Creates a record that can be viewed in the Message Reprocessing application. This record includes the message that was placed in the queue.

Configuring error management

To configure error management, you must configure system properties and configure the external system.

Configure error management properties:

Before you use the integration framework, configure properties in the System Properties application.

Procedure

1. Filter for the `mxe.int.adminfromemail` property and specify an email address, such as mxintadm@example.com. This address appears in error notifications that are submitted by the integration framework. Some SMTP servers require this address to be a valid email address format; some servers accept any value.
2. Filter for the `mxe.int.admintoemail` property and specify one or more email addresses to receive notification of message processing errors. Use commas to delimit several email addresses. You can optionally use the email address property at the queue level to override the administrator address. Use this option if you want to specify different email addresses for each queue. If you do not configure an email address, no email notification is sent when queue processing errors occur.
3. Filter for the `mail.smtp.host` property and specify an SMTP server if none has been configured. This property is not unique to the integration framework, and can be configured for other applications.

Configuring error management on the external system:

Configure the external system to handle error management.

Procedure

1. In the External Systems application, specify an appropriate value in the **Maximum Try Count** field. There is no limit to the number of times that the system retries the transaction. After the first unsuccessful attempt to process the transaction, the system administrator receives a notification, and a message is written to an error file. This value is typically set to 0 for outbound queues.
2. Specify a value in the **E-mail Address** field if you want error notification messages to be sent to different addresses for each queue. You can enter multiple addresses, delimited with a comma (,). The value in this property overrides the value in the administrator email address property. If no value is specified, email notifications are sent to the email addresses that are specified for the administrator email address property.
3. Click **Save**.

Persisting message delivery on WebSphere Application Server:

WebSphere Application Server includes a flag to persist the redelivery condition for error messages. Restarting the application server does not reset this flag and error messages are not reprocessed upon a server restart. WebSphere Application Server includes a property that you can use to configure message persistence.

Procedure

1. Log in to the WebSphere Application Server Admin Console, and navigate to **Service Integration** > **Buses** > **intjmsbus** > **Destinations** > **Bus Destination**.
2. Enable the **Keep count of failed deliveries per message** option.
3. Save the change.

Error notification

When an inbound or outbound transaction results in an error in a queue, an email notification is sent to the system administrator only if no other unresolved errors are waiting in the same queue. If multiple errors exist in the queue, the system administrator must resolve all of them before notification of new errors is sent.

An email error message includes a Java error stack trace.

The same notification process is used for all errors, for continuous and sequential queues, for inbound and outbound messages, and regardless of whether the system is running in a clustered or non-clustered environment.

The following example describes error notification for a continuous inbound queue contains ten messages. The first four messages are processed successfully and an error occurs on the fifth message. Depending on the value you set for the Maximum Try Count property for the external system, the message can be tried one or more times. If the message continues to cause an error, an email notification is sent to the system administrator and subsequent messages in the queue are processed. If another error occurs in the seventh message, another email notification is not sent if the system administrator has not resolved the original error. If the system administrator resolved the original error and no errors are pending, a new email notification is sent.

If the error is encountered in a sequential queue, processing is the same as in a continuous queue except that the system does not process subsequent messages until the message with the original error is resolved.

Multiple errors can exist only in the continuous inbound queue. In a clustered environment, the system administrator can receive one email error notification per application server, depending upon the timing of the transactions in error.

An uncommon exception condition can occur for outbound messages when a message is saved to the queue but the commit of the transaction to the queue fails. This exception can occur because a database connection to the JMS data store is not available. If this exception occurs, a notification is sent and the message in error is visible in the Message Reprocessing application with a status of either JMSERROR or SAVED. Because the message was not saved successfully in the queue, you cannot set the message status to RETRY, but you can process the message from the application.

Message reprocessing

In the Message Reprocessing application, you can manage messages that are flagged with an error, including changing the message status, correcting the message, or deleting it from the database.

If message tracking is enabled, use the Message Tracking application to determine which tracked messages are flagged with an error. If message tracking is not enabled, you can check for transaction errors in the Message Reprocessing application.

Message status values:

To change the status of a message, select the **Change Status** action in the Message Reprocessing application. The system designates a status to each message to indicate whether it is ready for processing.

A message can have a status of RETRY or HOLD:

Status	Description
RETRY	The message is ready to be reprocessed by the system.
HOLD	The message is not ready to be reprocessed by the system.

The RETRY status is the default status for messages that are flagged with an error. Until you correct the processing problem, the system continues to reprocess the message according to the configured queue retry count. When the retry count condition is met, the system changes the messages status to HOLD.

You can halt message reprocessing by changing the message status to HOLD. A hold status prevents the system from reprocessing the flagged message and from updating the system database tables.

Error XML messages:

You can review the XML message that is generated when an error occurs and you can modify the content of the message.

When you manage an error in the Message Reprocessing application, you can the see the following information:

- The Error Data field contains the error message.
- The message content contains the original message that was placed in the queue and is causing an error. You can edit this element.
- If available, the internal record contains the XML representation of the message at the time that the error occurred. You cannot edit the internal record but you can use it to view if integration processing changed the original message before the error occurred.

The internal record represents the object structure that was created during enterprise service and user exit processing. An internal record is available only for inbound transactions, and only when enterprise service and user exit processing complete successfully. If an internal record is available, it is provided for information only and you cannot change it.

Following is an example of an error XML message:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SyncMXPERSON xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      creationDateTime="2014-04-22T14:04:03-04:00"
      transLanguage="EN" baseLanguage="EN" messageID="11798570432187483"
      maximoVersion="7 6" event="1" messageid="11798570432652428">
   <MXPERSONSet>
    <PERSON action="Update">
    .
    .
    .
    </PERSON>
   </MXPERSONSet>
  </SyncMXPERSON>
```

Following is an example of an internal record XML message:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SyncMXPERSON xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      creationDateTime="2014-04-22T14:04:03-04:00"
      transLanguage="EN" baseLanguage="EN" messageID="11798570432187483"
      maximoVersion="7 6" event="1" messageid="11798570432652428">
   <MXPERSONSet>
    <PERSON action="Update">
    .
    .
    .
    </PERSON>
   </MXPERSONSet>
  </SyncMXPERSON>
 </IR>
</ERROR>
```

Critical errors:

Critical errors are processing exceptions that the integration framework error correction process cannot retry.

Transaction processing exceptions can occur when incorrect data, such as a special character, is present in the XML file. To correct a critical error, you remove the incorrect data from the error XML message. You can see incorrect data that is associated with a critical error in the main tab of the Message Reprocessing application.

Correcting errors:

In the Message Reprocessing application, you can either reprocess a message that is in error, or you can delete the message from the database.

Reprocessing an edited message:

You can view, modify, and reprocess messages in the Message Reprocessing application. After you edit the error XML message, you can save or cancel the changes without reprocessing the message.

About this task

You can edit only those messages that have a HOLD status. If the message has a RETRY status, the content of the message is read-only.

Procedure

1. In the Message Reprocessing application, select the message that you want to modify and click the **Message Details** icon.
2. In the Error Data window, make any necessary changes to the message.
3. Click **Process** if you want to reprocess the message. You can click **Save** to save the changes without reprocessing the message or you can click **Cancel** to discard any changes you made.

Results

If the processing is successful, the Message Reprocessing application performs the following tasks:

- Deletes the record in the error message table.
- Updates the DELETEFLAG, CHANGE BY, and CHANGE DATE attributes in the error status table

What to do next

In the Message Details window, refresh the list of messages that are flagged with an error. If the message was successfully reprocessed, it is dropped from the list.

Deleting messages:

You can delete messages from the error message queue. After you delete a message, it cannot be reprocessed.

Procedure

1. In the Message Reprocessing application, select the message records that you want to delete.
2. Select the **Delete Message** action.
3. Click **OK**.

Results

When a message is deleted, the record is deleted from the MAXINTERRORMSG and MAXINTERROR tables. The application refreshes the result set and omits the deleted message listing on the main tab of the Message Reprocessing application.

Refresh messages:

You can use the **Refresh** icon in the Message Reprocessing application to update the message list. When you refresh the message list, you can check on the status of a specific message listings.

If the message reprocessing was successful, the application omits the applicable message listings. If you delete a message, the application omits the deleted message listing.

Error management with file-based data import

The integration framework supports error management with the Message Reprocessing application which allows you to review, correct, reprocess, and delete messages that go in error when processed from an inbound queue. When data is imported from an XML file or a flat file, a second option is available that manages errors using a downloaded file instead of the Message Reprocessing application.

File-based error management is beneficial when you import a large number of messages that can cause a large number of errors. Managing a large number of errors in a file format can be easier and quicker than managing them individually in the Message Reprocessing application.

Related concepts:

"Exporting and importing file-based data" on page 218
The integration administrator can initiate export and import data from within the External Systems application to support, for example, integrating data using files. The import process includes the ability to preview a data load from a file to validate the data prior to saving it to the database. The import process also includes an option to manage errors that result from file loading in the same file format as the imported file

File-based error management:

You can use file-based error management with the data import feature. File-based error management provides the ability to identify all integration messages flagged with an error and download a complete file that contains all errors. The downloaded file is in a format that is consistent with the file that was used for the import.

To configure file-based error management, in the Data Import window, select the **File-based Error Management** check box.

When you select the **File-based Error Management** check box and leave the **Import Preview** check box cleared, you receive a system message after the inbound messages are processed successfully to the queue.

Inbound processing logic identifies processing errors on any of the messages in the source file and makes these failed messages available for download in a reprocessable file. The Message Reprocessing application provides a facility to download the reprocessable file.

Selecting the **File-based Error Management** check box means that any integration messages flagged with an error are available only in the reprocessable file and are not displayed in the messages section of the Message Reprocessing application.

Configuring error management in data import cron tasks:

You can set the **ISFILEEXTRACT** parameter in the XMLFILECONSUMER and FLATFILECONSUMER cron tasks to identify the error management mechanism that you want to use.

The following table contains the possible values for the **ISFILEEXTRACT** parameter:

Value	Error management mechanism
0	Error management is handled in the Message Reprocessing application.
1	File-based error management is handled in the Data Import window and in the reprocessable file.

File-based error management means that any integration messages flagged with an error are available only in the reprocessable file and are not displayed in the messages section of the Message Reprocessing application.

Information extracted by file-based error management:

File-based error management provides the ability to download a reprocessable file that contains all of the error messages that originated from a single input file.

The following table contains the information available in the Error Extract section of the Message Reprocessing application.

Field	Description
Import file	Name of the source file that generated the reprocessable file.
Enterprise service	Name of the enterprise service that was used to import the source file.
External system	Name of the external system that was used to import the source file.
Import date	Date and time when the data import process initiated processing of the source file.
Imported count	Total number of messages imported from the original source file.
Processed count	Number of messages processed successfully.
Error count	Number of messages that have errors.
File format	Format of the source file.
Available to extract	Identifies whether the inbound processing of the source file has completed.
Extract icon	Action button that initiates the file download process. To assure that the source file has been completely processed by the import mechanism, files are only available for download when the sum of error count and processed count is equal to the imported count.
Delete icon	Action button that deletes the selected record from the Error Extract table. You can only delete table records if the sum of the error count and processed count is equal to the imported count.

Downloading reprocessable files:

You can download a reprocessable file and fix the processing errors identified in the file. You can then attempt to reload the reprocessable files without removing any of the error message description information that is contained in the file.

Procedure

1. In the Message Reprocessing application, identify the source file from which you will be downloading its corresponding errors.
2. Verify that the source file has been fully processed by the data import process. The sum of the error count field and the processed count field must equal the imported count field.
3. Click the **Download** icon.
4. Save the reprocessable file to your client or to an accessible file server location.

Reprocessable file format:

When you download the reprocessable file, it is provided in the same format as the original input file. If you use the flat file format, the file uses the same delimiter and text qualifier. The default file name for the reprocessable file is *<UniqueFileIdentifier>_<OriginalFileName>.<OriginalFileExtension>*

Example of an XML reprocessable file

A reprocessable file based on an XML file, includes an additional element in the main MBO of the original object structure. For example, a reprocessable file generated based on MXASSET information includes the MAXINTERRORMSG element as part of the asset elements.

```
<?xml version="1.0" encoding="UTF-8"?>
  <SyncMXASSET xmlns="http://www.ibm.com/maximo" transLanguage="EN">
    <MXASSETSet>
      <ASSET>
        <ANCESTOR />
        <ASSETID>94</ASSETID>
        <ASSETNUM>THREE_T2002</ASSETNUM>
        <ASSETTAG />
        <ASSETTYPE />
        ...
        ...
        ...
        <WARRANTYEXPDATE>2020-12-24T00:00:00-05:00</WARRANTYEXPDATE>
        <YTDCOST>0.0</YTDCOST>
        <MAXINTERRORMSG>
          The following error occurred while processing ASSET.BMXAA4147E -
          Item set error1 does not exist.
        </MAXINTERRORMSG>
    </ASSET>
    ...
    ...
    ...
  </MXASSETSet>
</SyncMXASSET>
```

Example of flat format reprocessable file

Reprocessable files that follow a flat file structure include an additional column. For example, a reprocessable file generated based on MXASSET information includes the MAXINTERROR column at the end of the original record structure:

```
EXTSYS1,MXASSETInterface,,EN
ASSETNUM,AS_DESCRIPTION,AS_DESCRIPTION_LD,HIERARCHYPATH,AS_SITEID,MAXINTERRORMSG
T-SP500_error,autospray,,,TEXAS,
The following error occurred while processing ASSET.BMXAA4049E -
The value specified T-SP500_error exceeds the maximum field length.
```

Deleting reprocessable files:

You can permanently delete data that is available to reprocess as a file.

Procedure

1. In the Message Reprocessing application, identify the row of reprocessable data that you want to delete.
2. Click the **Delete** icon.

Interface table error management

Interface tables can be used for inbound and outbound data exchange. Errors can occur when writing inbound data from an interface table to a queue or outbound data from a queue to an interface table.

If an error is encountered by the cron task that pulls an outbound message from a queue, the message remains in the queue until the error condition is addressed. Errors may occur for the following reasons:

- The interface table does not exist.
- There is database error due to lack of space.
- The message content, as defined by the object structure, was altered but the interface table was not recreated to reflect the new message format.

When an outbound message reaches an interface table, it is the responsibility of the external application to retrieve that data and manage errors based on their integration implementation.

The cron task that writes inbound messages to a queue can encounter errors for the following reasons:

- The JMS queue is deactivated or free space is not available.
- The enterprise service or external system name is not valid.
- The enterprise service is not enabled for the external system.
- The external system is not enabled.

When an error occurs during inbound interface table processing, the polling program writes the exception trace in the IMPORTMESSAGE column of the MXIN_INTER_TRANS queue table. For the first error in the MXIN_INTER_TRANS queue table, the system sends an email notification to the administrator. You can resolve the error condition by updating the row of data in MXIN_INTER_TRANS, for example, correcting the value of the external system name, or by updating the configuration data in the application, for example, marking the enterprise service as enabled.

After the cron task processes subsequent records in the MXIN_INTER_TRANS queue table, it switches to an idle state that is based on the defined cron task processing intervals. When processing resumes, the cron task tries to process the records in error, as well as new records added to the MAX_INTER_TRANS queue table.

After sending an error notification, the cron task does not send notification of additional errors if the queue table contains one transaction that is marked in error. It is assumed that the person who was notified of the initial error sees and corrects additional errors when the queue table is examined. After all current errors are corrected, the cron task sends a notification when it encounters a new error.

Any errors that occur after the cron task successfully writes an interface table message to an inbound queue are managed by the error handling process for the queues.

Common causes of errors

Errors when processing messages in the outbound queue occur because there is a problem delivering a message to the endpoint that is specified for the external system. Errors when processing messages in an inbound queue are typically related to a business rule validation or in the inbound processing of the enterprise service.

For outbound processing, typical problems are disruptions of the communication link to the external application, issues with the database table space, or file space issues in the external application. To resolve an outbound error, you typically do not need to modify the XML message.

The following table describes the most common message errors that you can encounter and provides suggestions for correcting them. Correcting an error in the XML message can create a mismatch in data between the sending and receiving systems.

Error type	Description	Actions
Sequence error	Caused by sequencing problems between messages. The system stops message processing when a record references another record that is in a pending state.	This error is applied to ensure that records process in the correct order. Depending on the direction of the message transaction and the processing logic that applies, the processing can self-correct the error when the record that was in the pending state has processed.
Data error	Occurs because the data or record does not exist in the system database, and is not part of the inbound messages in the queue.	Add the missing data to the system database.
Communication error	Caused by communication problems with the external system from system failures or network issues.	Restore communication with the external system.
Message error	Caused by erroneous message data values.	• Change the transaction status to HOLD. • Correct the error XML message. • Change the transaction status to RETRY to reprocess the transaction.

Error research

When you receive an error notification, look at the XML message in the Message Reprocessing application. Depending on the type of queue (sequential or continuous), the number of messages in the queue can be zero, one, or more in the Message Reprocessing application for an individual queue.

No messages in error exist

If no message in error exists in the Message Reprocessing application, the message was retried and the error was not encountered again. The error message was deleted when the message was successfully processed.

For example, an error occurs in an inbound receipt message due to an incorrect general ledger (GL) account. After the error occurs, an online user enters that GL account in the system. The message is reprocessed successfully and the data is saved.

In another example, an outbound transaction encounters a communication error. When the communication problem is resolved, the message is sent to the external system and the message in error is deleted.

One or more messages in error exist

When an error occurs in a sequential queue (inbound or outbound), processing of the queue stops until the error is resolved.

When an error occurs in a continuous queue, processing of the queue continues and additional errors can occur before the initial error is resolved. Multiple messages in error can exist in the Message Reprocessing application.

Message tracking

The Message Tracking application tracks and displays the processing history of publish channel messages and queue-based enterprise service messages.

The Message Tracking application works with the Message Reprocessing application. When you use the Message Tracking application, you can determine which messages are flagged with an error. You then can select a failed message and go to the Message Reprocessing application to take appropriate action to correct erroneous data.

Message details:

When you enable message tracking, the integration framework writes all processed messages to the MAXINTMSGTRK table. A status is assigned to each message which represents its current position in the queue-based processing cycle. Individual message events are displayed in the Message Details window.

When you enable message tracking, queue messages that existed before the function was enabled are not identified by the message tracking logic. When you disable message tracking, pre-existing queue messages that existed before the function was disabled are identified but new messages are not identified.

Messages have the following attributes, and values are assigned based on the originating enterprise service or publish channel data:

- Integration mode: The name of the integration mode used in processing the message. For inbound messages, the system assigns an MXJMS default value. For outbound messages, the system assigns the name of the endpoint that is used in message processing.
- Operation: The processing operation the system applies to the tracked message, which can be any of SYNC, UPDATE, QUERY, DELETE, CREATE, and PUBLISH.
- System: The name of the external system that is associated with either the enterprise service or publish channel.
- Integration component: The name of the enterprise service or publish channel.
- Adapter: The name of the adapter that is associated with either the enterprise service or publish channel.
- Queue name: The name of the queue used by the integration framework to process the message.

The following attributes are assigned values at the time the transaction record is created.

Attribute	Value
Received Datetime	The date and time the message was received in the queue.
Message ID	Unique message identifier that is assigned by the integration framework.
Search ID	Message identifier that is assigned by an external application and that is used in the message searches.
External Message ID	Unique message identifier that is assigned by an external application.

The following attributes have dynamic values that change based on the transaction events.

Attribute	Value
Current Status	The most current processing status for the tracked message.
Status	The status that is associated with the individual message event in the transaction history.
Status Date	The status date for the individual message event in the transaction history.
Error	The error message for the individual error message event in the transaction history.

Message status values:

Every inbound and outbound queue-based message that is registered in the Message Tracking application has a status value that indicates its position in the transaction processing cycle.

The message tracking status indicates whether the message was successfully received or processed. The message tracking status also indicates whether the message was deleted, or flagged with errors.

Inbound message status

Inbound messages can have the following status values:

Status	Description
ERROR	Message processing failed due to validation problems.
DELETED	Message was deleted from the queue.
PROCESSED	Message was successfully processed.
RECEIVED	Message was successfully written to the inbound queue.

Outbound message status

Outbound messages can have the following status values:

Status	Description
ERROR	Message processing failed due to validation problems.
DELETED	Message was deleted from the queue.
PROCESSED	Message was successfully processed.
RECEIVED	Message was successfully written to the outbound queue.

Message events:

The Message Tracking application tracks and displays inbound and outbound queue-based transaction processing events. Transaction processing events trigger the system to update the MAXINTMSGTRK table.

The following message table attributes are updated according to event type:

- STATUS
- STATUSDATETIME
- ERRORMSGR

The following inbound and outbound events update the MAXINTMSGTRK table:

Table 32. Tracked inbound and outbound events

Event	Details
Message is written to queue	A record is created in the message tracking table when the integration framework first writes the message to the queue. When the message is successfully written to the queue, the message record status is set to RECEIVED. If the integration framework encounters an error when it is writing an inbound message to the queue, it sends a message to the process caller detailing the cause of failure.

Table 32. Tracked inbound and outbound events (continued)

Event	Details
Error in message processing	The existing record in the message tracking table. When the asset management system encounters a processing error, it updates the message record status to ERROR. If you resend your message and a processing error occurs again, the asset management system maintains the ERROR message status.
End-of-queue processing	The following transaction processing events update the existing record: • The asset management system successfully completes the message processing and updates the message record status to PROCESSED. Because the processing cycle is complete, no further updates are made to the message tracking table. • If you delete the message from the queue, the asset management system sets the message record status to DELETED. The message tracking table is no longer updated.

Message tracking configuration:

You can track messages that are sent through publish channels or that are received from enterprise services.

In the Publish Channels and Enterprise Services applications, you can configure the following message tracking functions:

- Enable or disable message tracking.
- Store transaction messages in the database along with the tracking details.
- Specify the message data that the Message Tracking application search function uses by using an XPATH expression.
- Uniquely identify messages with a single ID value by using an XPATH expression.
- Identify messages with a search ID value by using an XPATH expression.

The XPATH expressions that are associated with the external message ID values and the search ID values can identify multiple nodes in an XML file. In this case, the message ID and search ID values are registered as a comma-separated list of values. Database field lengths are applicable to external ID and search ID fields. If necessary, you can adjust the length of these fields in the Database Configuration application.

Stored messages

When you configure message tracking, messages and message tracking details are saved in the database and can be viewed in the Message Reprocessing application.

External message tracking ID

Each inbound message has an external message identifier that is stored in the MAXINTMSGTRK table. With the Message Tracking application, you can use this external message ID to locate specific messages. The syntax that you use to identify a message node must be a fully qualified XPATH expression.

To find all messages for the MXPERSONInterface enterprise service, specify the following fully qualified XPATH expression in the **External Message ID** field:

```
/{http://www.ibm.com/maximo}SyncMXPERSON/@messageID
```

When a multi-noun inbound message is received, the integration framework uses the XPATH expression for the external message ID to identify the message. If the XPATH expression points to an element included in each one of the nouns in the inbound message, the integration framework creates a multi-noun, comma-separated list of external identifiers.

Search ID

By specifying an XPATH expression to identify nodes, the integration framework can perform efficient multi-noun searches. The system stores the search identifier in the MAXINTMSGTRK table.

To find all messages for the MXPERSONInterface enterprise service, create the following fully qualified XPATH expression as the search ID:

```
/{http://www.ibm.com/maximo}SyncMXPERSON/{http://www.ibm.com/
maximo}MXPERSONSet/{http://www.ibm.com/maximo}PERSON/{http://www.ibm.com/
maximo}PERSONID
```

When a multi-noun inbound message is received, the integration framework uses the XPATH expression for the search ID to identify the message. If the XPATH expression points to an element included in each one of the nouns in the inbound message, the integration framework creates a multi-noun, comma-separated list of search identifiers.

Enabling message tracking:

Enable message tracking for outbound transactions in the Publish Channels application and for inbound transactions in the Enterprise Services application.

Procedure

1. In the Publish Channels application or the Enterprise Services application, select the channel or service that processes integration messages.
2. Select the **Message Tracking** action.
3. In the Message Tracking Setting window, select the **Enable Message Tracking** check box.
4. Optional: Select the **Store Message** check box to store transaction messages.
5. Optional: In the **External Message ID** field, specify one or more XPATH expressions to locate specific messages.
6. Optional: In the **Search ID** field, specify one or more XPATH expressions to search for messages.

Cluster configuration

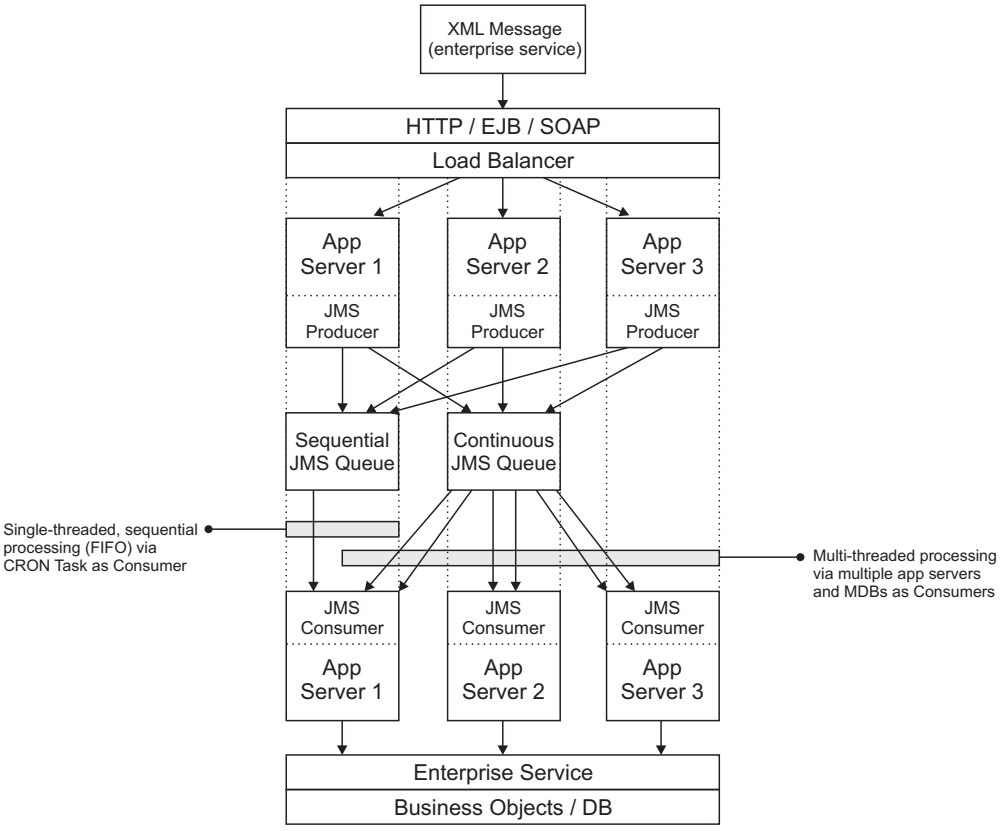
The integration framework can provide integration services across a cluster of application servers. If the continuous queue on a single server is inadequate for your message volume, you can add central processing units, hardware, and a cluster configuration to improve message processing performance.

When you implement a server cluster, you can use multiple servers simultaneously to process inbound messages. These messages are processed in the continuous queues by using message-driven beans (MDBs). Cluster configurations facilitate the processing of messages in large volumes.

JMS queues in a server cluster

In a cluster configuration, JMS queues are associated with one member of the server cluster and access to the queues is provided by the Java Naming and Directory Interface (JNDI) service.

The JDNI service is available across all the members on the cluster.



All cluster members can produce messages into the sequential queue. A single-threaded cron task reads the messages in the sequenetial queue to support first-in-first-out processing.

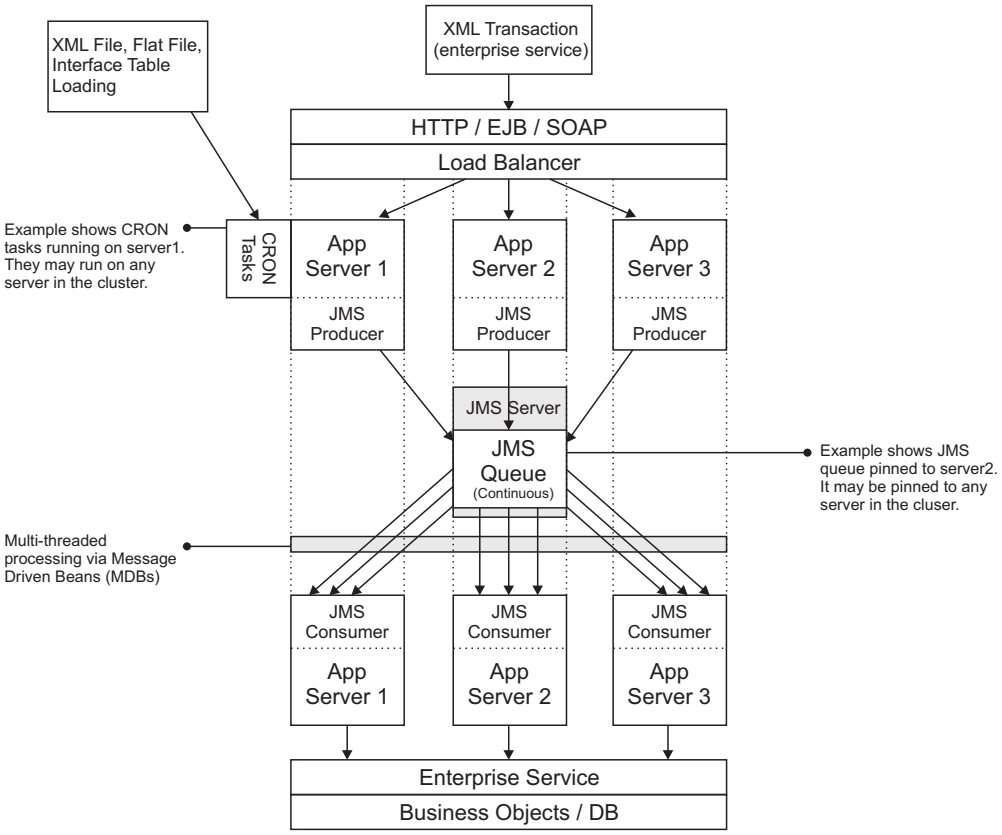
The continuous queue is multi-threaded on the consumer side to support message processing in high volumes. In this queue, the message processing order is not considered.

Enterprise service messages that use the sequential queue are processed in a strict sequential order. Message processing is single-threaded. Clustering does not significantly impact the processing performance of messages through the sequential queue.

Continuous queue on an application server cluster:

In a clustered environment, you can pin the continuous JMS queue to one member of the cluster.

The following diagram shows an example of a cluster configuration on an application server. The continuous queue receives JMS producer messages and processes the JMS consumer messages.



In the example, three application servers exist in the clustered environment. The continuous JMS queue is pinned to one member of the cluster.

The processing occurs when the integration framework receives enterprise service messages by using HTTP, enterprise beans, and SOAP actions. The load balancer directs the application server to drop enterprise service messages into the continuous queue. Each member of the cluster places messages into the queue, which exists on one member of the cluster.

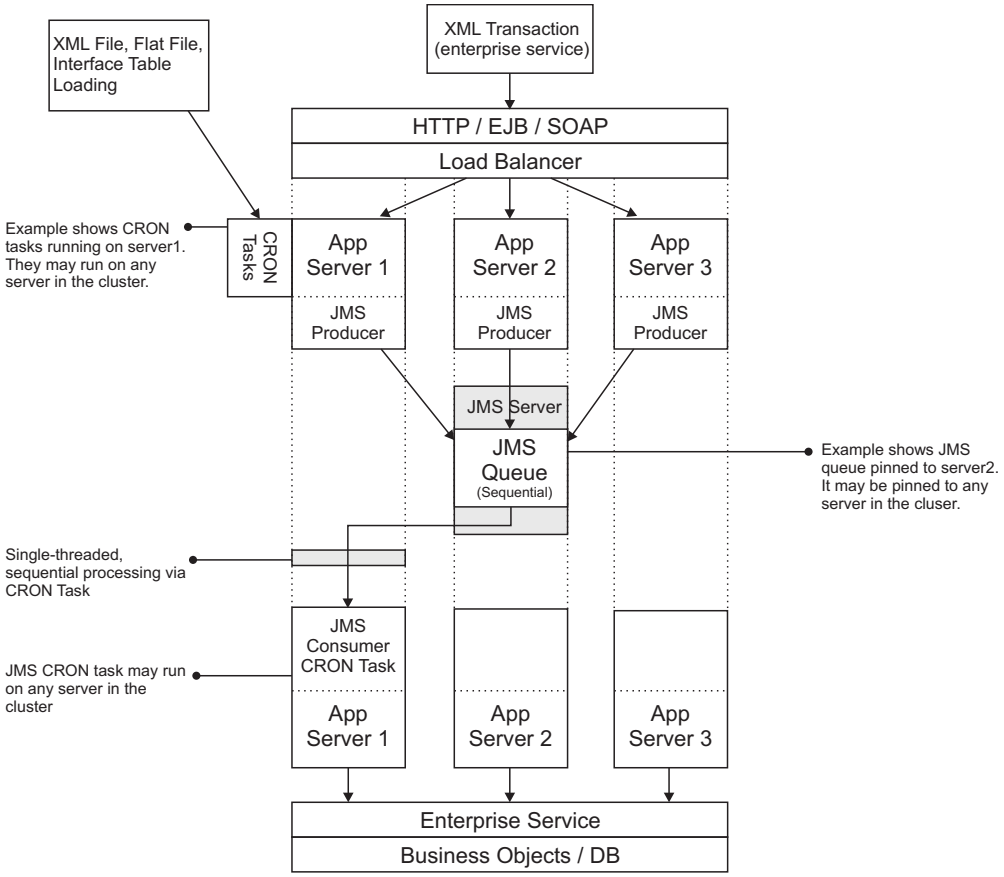
An integration cron task delivers enterprise service messages from flat or XML files and interface tables. The cron tasks that place messages into the queue can run on any server in the cluster.

After messages are in the queue, all application servers can simultaneously pull messages from the queue and process them into the system. Message-driven beans must be enabled on each application server in the cluster.

Sequential queue on an application server cluster:

The sequential queue receives JMS producer messages and processes the JMS consumer messages.

The following diagram shows one example of a cluster configuration on application servers.



In the example, three application servers exist in the clustered environment. The sequential JMS queue is pinned to one member of the cluster.

The processing occurs when the integration framework receives enterprise service messages by using HTTP, enterprise beans, and SOAP actions. The load balancer directs the application server to drop enterprise service messages into the sequential queue. Each member of the cluster places messages into the queue, which exists on one member of the cluster.

An integration cron task delivers enterprise service messages from flat or XML files and interface tables. The cron tasks that place messages into the queue can run on any server in the cluster. Once messages are in the queue, the application server that is running the JMS consumer cron task processes messages in a sequential order.

Unlike the continuous queue, there is no multi-threading of messages by design. A cluster implementation does not significantly impact the processing performance of messages that process through the sequential queue.

Configuring the cron task

The interface table cron task, the data import cron task, and the JMS queue cron task are cluster-aware functions. By default, the cron task framework runs a task on a randomly chosen server. You can configure a cron task to run on a specific application server within a server cluster. Use the donotrun parameter in the cron task framework to control which servers the cron task runs on.

Configuring a message processing server

For high volumes of inbound messages, you can improve server efficiency by moving inbound message processing to a separate application server or server cluster.

Procedure

1. Deploy a separate application EAR file on the message processing server or server cluster.
2. On the message processing application server or server cluster, use the donotrun parameter in the cron task framework to control which servers the cron task runs on.
3. Optional: Configure message-driven beans to pull data from the inbound continuous queue. Do not enable message-driven beans for inbound processing within the server or server cluster that is dedicated for online users. You can run other background processing cron tasks on this server or server cluster.
4. Optional: Grant user interface access to users who use of the data import feature.

Global directory configuration

In a cluster environment, if you configure a global directory for management of integration files, the directory must be accessible to all members of the cluster.

Depending upon your use of the integration framework components, files may or may not be written to the global directory. In a multitenancy environment, a global directory is required. One of the main reasons for configuring a global directory is to support the use of file-based endpoints. If you intend to use file-based endpoints for invocation channels and messages are initiated by application users that are configured in a UI cluster, all members of the cluster must have visibility to the global directory.

To define the global directory, update the `mxe.int.globaldir` system property in the System Properties application.

Access to services by inbound messages

Enterprise services, object structure services, and standard services can all function in a server cluster.

An inbound transaction can access the relevant service with any of the following communication methods:

- Java remote method invocation (RMI), Internet Inter-ORB Protocol (IIOP), and enterprise beans
- HTTP and HTTPS servlet
- SOAP, HTTP, and HTTPS servlet

Enterprise beans

With a single server, the provider URL for accessing the JNDI tree is the single server URL. With a server cluster, the provider URL can be the URL of any of the servers that has the enterprise beans deployed. All members of the cluster share the JNDI tree and any member of the cluster can look for and retrieve a cluster.

Following a cluster look up, the client retrieves a cluster-aware proxy of the enterprise beans which load balances all the subsequent calls that use that proxy. Load balancing happens transparently to the client code. There is no difference between the code for a single server and for a cluster setup. A separate enterprise bean is deployed for each type of service: object structure service, enterprise service, and standard service.

HTTP servlet

The integration servlet is deployed across all members of the cluster. With a single server configuration, the URL is the HTTP and HTTPS URL of that server. With a cluster server configuration, the URL is the HTTP and HTTPS URL of the load balancer for the cluster. A separate servlet is deployed for each type of service: object structure, enterprise, and standard service.

The URL formats for each service are shown in the following table. The *meaweb* variable in the URL represents the value you specify for the `mxe.int.webappurl` in the System Properties application.

Service	URL
Object structure service	`http://`*`hostname`*`:`*`port`*`/`*`meaweb`*`/os/object structure name`
Enterprise service (bypassing queue)	`http://`*`hostname`*`:`*`port`*`/`*`meaweb`*`/es/extsysname/enterprise service name`
Enterprise service (through the queue)	`http://`*`hostname`*`:`*`port`*`/`*`meaweb`*`/esqueue/extsysname/enterprise service name`
Standard service	`http://`*`hostname`*`:`*`port`*`/`*`meaweb`*`/ss/standard service name`

Web services

Integration web services are homogeneously deployed across all the server members in the cluster. Web service access for a cluster is the same as for a single server, except that the web service URL and Web Service Definition Language (WSDL) URL point to the cluster instead of to a specific server in the cluster.

The following properties must point to the cluster URL:

- Web application URL `mxe.int.webappurl`
- UDDI registry inquiry URL `mxe.int.uddiinurl`
- UDDI registry publish URL `mxe.int.uddipuburl`

The URL to access a web service is `http://`*`hostname:port`*`/`*`meaweb`*`/services/`*`web service name`*. The *meaweb* variable in the URL represents the value you specify for the `mxe.int.webappurl` in the System Properties application.

Integration security

The integration framework includes support for J2EE authentication and for component-level authorization.

Authentication security

You can configure J2EE authentication security for JMS queues, EJB, HTTP, and web services. You can also configure authentication security for remote integration APIs and for the Java handler classes for outbound routing.

Configuring Maximo authentication:

Maximo authentication requires some configuration to enable inbound transactions.

Maximo authentication is enabled by setting the **mxe.useAppServerSecurity** system property to 0 (false). When Maximo authentication is used, all HTTP-based inbound transactions must specify an HTTP header with the following attributes:

Field	Value
Name	*MAXAUTH*
Value	*<User credentials encoded to base64 format>*

The user credentials must be encoded to base64 in the following format: *<username>:<password>*.

Services that are called through the integration servlet and SOAP-based web services do not require an HTTP MAXAUTH header if the default integration user is enabled. You can enable the default integration user by setting the ALLOWDFLTLOGIN value to 1 (true) in the `ejb-jar.xml` deployment file as shown in the following code example:

```
<env-entry>
  <env-entry-name>ALLOWDFLTLOGIN</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>1</env-entry-value>
</env-entry>
```

The `ejb-jar.xml` deployment file can be found on your Maximo administrative workstation in the following directory: `c:\maximo\applications\maximo\mboejb\ejbmodule\META-INF`. The ALLOWDFLTLOGIN value must be set to 1 (true) for each of the following services in the XML file:

- Enterprise Services
- Object Structure Service
- Standard Services
- Workflow service

The default integration user is not supported if you use the REST API or OSLC API. After you update the `ejb-jar.xml` deployment file, you must rebuild and redeploy the Maximo EAR file.

Configuring J2EE security:

The integration framework supports basic J2EE security to restrict access based on authentication and authorization.

Configuring J2EE restrictions for JMS queues:

The JMS queues that are used by integration processing support J2EE security, based on user ID and password-based authentication and authorization. You can prevent unauthorized access to the queue by assigning a user ID and password to the Java Naming and Directory Interface (JNDI) name, even if the JNDI name of the queue is known.

About this task

Multiple queues can use the same or different user identifications.

Procedure

1. On the administrator console for the server, specify user ID and password values for the following properties, to enable J2EE restrictions:
   - java.naming.security.principal (user ID)
   - java.naming.security.credentials (password)
2. In the External Systems application, select the **Add/Modify Queues** action and specify the same user ID and password that you provided in Step 1. This step provides access to the queue to the integration producer and consumer programs.
3. To provide access to the continuous queue, under the <enterprise-beans> section in the `ejb-jar.xml` file, add the <security identify> elements shown in bold text:

```
<enterprise-beans>
  <message-driven id="MessageDriven_JMSContQueueProcessor_1">
    <ejb-name>JMSContQueueProcessor-1</ejb-name>
    <ejb-class>psdi.iface.jms.JMSContQueueProcessor</ejb-class>
    <transaction-type>Container</transaction-type>
    <message-driven-destination>
      <destination-type>javax.jms.Queue</destination-type>
    </message-driven-destination>
    <env-entry>
      <env-entry-name>MESSAGEPROCESSOR</env-entry-name>
      <env-entry-type>java.lang.String </env-entry-type>
      <env-entry-value>
        psdi.iface.jms.QueueToMaximoProcessor
      </env-entry-value>
    </env-entry>
      <security-identity>
        <run-as>
          <role-name>maximouser</role-name>
        </run-as>
      </security-identity>
    </message-driven>
```

4. Under the <assembly-descriptor> section in the `ejb-jar.xml` file, add the <security-role> elements shown in bold text:

```
<assembly-descriptor>
  <security-role>
    <role-name>maximouser</role-name>
  </security-role>
  <container-transaction>
    <method>
      <ejb-name>JMSContQueueProcessor-1</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

Securing enterprise bean access:

If J2EE Authentication on the system is enabled, you must enable the security for each enterprise bean in the deployment descriptors.

About this task

Under the <enterprise-beans> section of the `ejb-jar.xml` file, three integration EJBs are deployed with a default value of 1, which indicates that no authentication is required.

The <ejg-name> to service mapping is:

<ejb-name>	**Service**
enterpriseservice	Enterprise Service
mosservice	Object Structure Service
actionservice	Standard Service

Procedure

1. To force authentication, change the ALLOWDFLTLOGIN value to 0 (false), for each of three services, indicated in bold in the following code example:

```
<enterprise-beans>
<session id="Session_enterpriseservice">
  <ejb-name>enterpriseservice</ejb-name>
  <home>psdi.iface.gateway.MEAGatewayHome</home>
  <remote>psdi.iface.gateway.MEAGateway</remote>
  <local-home>psdi.iface.gateway.MEAGatewayHomeLocal</local-home>
  <local>psdi.iface.gateway.MEAGatewayLocal</local>
  <ejb-class>psdi.iface.gateway.MEAGatewayBean</ejb-class>
  <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <env-entry>
      <env-entry-name>ALLOWDFLTLOGIN</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>0</env-entry-value>
    </env-entry>
    <security-role-ref>
      <description>
        Application Users
      </description>
      <role-name>maximouser</role-name>
      <role-link>maximouser</role-link>
    </security-role-ref>
  </session>
  <session id="Session_mosservice">
    <ejb-name>mosservice</ejb-name>
    <home>psdi.iface.mos.MOSServiceHome</home>
    <remote>psdi.iface.mos.MOSServiceRemote</remote>
    <local-home>psdi.iface.mos.MOSServiceHomeLocal</local-home>
    <local>psdi.iface.mos.MOSServiceLocal</local>
    <ejb-class>psdi.iface.mos.MOSServiceBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <env-entry>
      <env-entry-name>ALLOWDFLTLOGIN</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>0</env-entry-value>
    </env-entry>
    <security-role-ref>
      <description>
```

```
      Application Users
    </description>
    <role-name>maximouser</role-name>
    <role-link>maximouser</role-link>
   </security-role-ref>
  </session>
<session id="Session_actionservice">
  <ejb-name>actionservice</ejb-name>
  <home>psdi.iface.action.MAXActionServiceHome</home>
  <remote>psdi.iface.action.MAXActionServiceRemote</remote>
  <local-home>psdi.iface.action.MAXActionServiceHomeLocal</local-home>
  <local>psdi.iface.action.MAXActionServiceLocal</local>
  <ejb-class>psdi.iface.action.MAXActionServiceBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
  <env-entry>
    <env-entry-name>ALLOWDFLTLOGIN</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>0</env-entry-value>
  </env-entry>
  <security-role-ref>
    <description>
      Application Users
    </description>
    <role-name>maximouser</role-name>
    <role-link>maximouser</role-link>
  </security-role-ref>
</session>
```

Client programs call the secure version of the enterprise bean methods for each service type:

- Enterprise service: secureProcessExtnernalDataAsync(..) , secureProcessExtnernalDataSync(..)
- Object structure service: secureProcessMOS(..)
- Standard service: secureAction(..)

2. To create a secure context for calling the enterprise bean, perform either one of the following tasks:
   - Add the following code to to the client code:

     ```
     Properties env = new Properties();
     .
     .
     .
     if(userid != null && password != null)
     {
     env.put(Context.SECURITY_CREDENTIALS, password);
     env.put(Context.SECURITY_PRINCIPAL, userid);
     }

     Context ctx = new IntialContext(env);
     //instead of using the default IntialContext() constructor
     ```

   - Use the default InitalContext constructor to pass the security information through –D parameters in the .bat/.sh script that launches the client:

     ```
     –Djava.naming.security.principal=<username>
     –Djava.naming.security.credentials=<password>
     ```

     The SSL version of the Internet Inter-ORB protocol performs data encryption in the provider URL, while the system communicates with the enterprise bean.

Securing the HTTP servlet:

The HTTP servlet is a J2EE component that handles inbound HTTP posts. To secure the HTTP servlet, you must first secure the enterprise bean. You can use HTTP basic authentication to secure the HTTP servlet. Authorized users, with a valid user name and password can post an XML transaction to the system.

About this task

To enable HTTP basic authentication, modify the web.xml file of the Web application:

- Remove the comments from the <security-constraint> section of the integration servlets. There are three <security-constraint> sections, one for each type of service: enterprise service, object structure service, and standard service.

The <web-resource-name> to service mapping is:

<web-resource-name>	**Service**
Enterprise Service Servlet	Enterprise Service
App Service Servlet	Standard Service
Object Structure Service Servlet	Object Structure Service

Procedure

1. In the web.xml file, uncomment the security constraint sections for each service type, as in the following code example:

```
<!--
<security-constraint>
<web-resource-collection>
<web-resource-name>Enterprise Service Servlet</web-resource-name>
<description>
  Enterprise Service Servlet (HTTP POST) accessible by authorized users
</description>
<url-pattern>/es/*</url-pattern>
<url-pattern>/esqueue/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<description>
  Roles that have access to Enterprise Service Servlet (HTTP POST)
</description>
<role-name>maximouser</role-name>
</auth-constraint>
<user-data-constraint>
<description>data transmission gaurantee</description>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>App Service Servlet</web-resource-name>
<description>
  App Service Servlet (HTTP POST) accessible by authorized users
</description>
<url-pattern>/ss/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
```

```
<auth-constraint>
<description>
  Roles that have access to App Service Servlet (HTTP POST)
</description>
<role-name>maximouser</role-name>
</auth-constraint>
<user-data-constraint>
<description>data transmission gaurantee</description>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>Object Structure Service Servlet</web-resource-name>
<description>
  Object Structure Service Servlet (HTTP POST) accessible by authorized users
</description>
<url-pattern>/os/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<description>
  Roles that have access to Object Structure Service Servlet (HTTP POST)
</description>
<role-name>maximouser</role-name>
</auth-constraint>
<user-data-constraint>
<description>data transmission gaurantee</description>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
    </security-constraint>

    -->
```

2. Verify that the <security-role> section in the web.xml file is not commented out, as in the following example code:

```
<security-role>
  <description>An Integration User</description>
  <role-name>maximouser</role-name>
</security-role>
```

3. Change the value from 0 to 1 in the useAppServerSecurity <env-entry-name> section, as in the following example:

```
<description>
  Indicates whether to use Application Server security or not
</description>
<env-entry-name>useAppServerSecurity</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>1</env-entry-value>
</env-entry>
```

What to do next

You can securely deploy a web service by using a Secure Socket Layer (SSL) for HTTPS posts. Configure the SSL on the application server with the appropriate digital certificates.

Securing web services:

You can secure integration web services by using HTTP basic authentication in standard J2EE security. These security settings provide access to web services to authorized users with a valid user name and password.

Procedure

1. Similar to the procedure for securing the HTTP servlet, in the web.xml file, uncomment the <security-constraint> section for the web service invocation, as in the following example

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Integration Web Services</web-resource-name>
    <description>
      Integration Web Services accessible by authorized users
    </description>
    <url-pattern>/services/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description>
      Roles that have access to Integration Web Services
    </description>
    <role-name>maximouser</role-name>
  </auth-constraint>
  <user-data-constraint>
    <description>data transmission gaurantee</description>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

2. Verify that the <security-role> section in the web.xml file is not commented out, as in the following example code:

```
<security-role>
  <description>An Integration User</description>
  <role-name>maximouser</role-name>
</security-role>
```

3. Change the value from 0 to I in the useAppServerSecurity <env-entry-name> section, as in the following example:

```
<description>
  Indicates whether to use Application Server security or not
</description>
<env-entry-name>useAppServerSecurity</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>1</env-entry-value>
</env-entry>
```

4. For web service invocation, ensure that the client program uses the following user name and password calls in the JAX-RPC Call object:

```
call.setProperty(Call.USERNAME_PROPERTY, username);
call.setProperty(Call.PASSWORD_PROPERTY, password);
```

What to do next

You can securely deploy a web service by using a Secure Socket Layer (SSL) for HTTPS posts. Configure the SSL on the application server with the appropriate digital certificates.

Interface table security:

Interface tables use the default database authentication and authorization. If authentication and authorization are in effect, external programs that read or write to the interface tables must provide proper authorization. To read from and write to the interface tables, the USERNAME and PASSWORD values are configured for the endpoint that implements the interface table handler.

Securing remote integration APIs:

Some MicService remote APIs secure access by forcing the user of those methods to provide the UserInfo object. If a valid UserInfo object is not provided, an error occurs and the call is not completed.

The following remote methods are protected because they either provide sensitive information or perform sensitive data transaction processing:

- exportData(..)
- deleteQueueData(..)
- processExternalData(..) (both versions)
- query(..)
- viewQueueData(..)
- loadData(..)
- loadSystemData(..)
- processObjectStructure(..)
- routeData(..)

To run these methods, the caller must retrieve a valid UserInfo object and pass it to the method to gain access to the secure layer.

A UserInfo object is a serialized object that contains user details including user, password, locale, language, and time zone information, that is used for security purposes.

The system uses Java Remote Method Invocation (RMI) and Java Remote Method Protocol (JRMP). You can communicate to the system services by using a secure version of JRMP protocol using SSL.

Outbound router handler security:

The outbound router handlers have support for authorization and confidentiality. The enterprise bean, HTTP, JMS, web service, and interface table handlers have support for security.

Authorization security

You can configure authorization security at application-level, object-level, object structure-level, or at the level of a method defined in a standard service. After user authentication completes, the integration framework checks that the user has been granted the authority to send messages to the target application, object, object structure, or method.

Object-level authorization

Object-level authorization is based on the security configuration set in the Data Restrictions tab of the Security Groups application. If an object or attribute is marked as read-only or hidden, inbound message data processing is limited to queries. You cannot insert, update, or delete data in the relevant object or attribute.

Object structure-level authorization

You configure object structure-level authorization by using the **Configure Object Structure Security** action in the Object Structures application. You can define

whether your object structure inherits the authorization from a related application or whether your object structure defines its own authorization by using signature options.

To reuse the signature option of a related application, in the **Authorization Name** field, specify the application name. The signature options of the application are used by default for the object structure. You can choose which signature options to apply to the object structure.

To define the authorization for the option for an object structure, select the **Use Object Structure for Authorization Name** check box. The authorization name is set to the object structure name, and the default insert, delete, save, and read are assigned. For some object structures, you can then create signature options for any supported services that are available by using the object structures. Click **Select Service Option** to view and create signature options for these services.

After authorization for the object structures is defined, a security administrator can configure the authorization for a security group by using the Security Groups application.

If the `mxe.int.enablement` system property is set to `1`, the authorization of the object structure must be configured before users can access and update data. When the system property is set to `0`, authorization is not required.

Application-level authorization

You configure application-level authorization in the Object Structures application. In the **Authorized Application** field, specify the application to authorize. The specified application and the user group of the integration message user together provide authorization for inbound integration messages for both object structures and enterprise services. The combination of application and user group also provide authorization for the export of data related to this object structure.

If you use the REST API, you might need to configure application-level authorization to access business object resources.

Standard service authorization

Standard service authorization does not support the use of a condition that you associate with the signature option. Any condition that you assign is ignored.

You can configure a standard service transaction to take on the same security profile for the integration user as if that user entered the transaction through an application. This level of authorization requires manual configuration. The application service must have a properly annotated method and the service must be registered in the Database Configuration application. Assign a signature option to a standard service to limit access to the users or groups that are authorized for the selected option.

Run a SQL script that updates the MAXSERVSECURITY table with the details of the standard service to authorize. The insert statement for the MAXSERVSECURITY must include the fields listed in the following table.

Field	Description
MAXSERVSECURITYID	A unique ID that is numeric. You can, for example, query the max(MAXSERVSECURITYID) from MAXSERVSECURITY and use the next sequential value.
ROWSTAMP	A unique ID that is numeric. You can, for example, query the max(ROWSTAMP) from MAXSERVSECURITY and use the next sequential value.
SERVICENAME	The service name registered in the Database Configuration application (MAXSERVICE.SERVICENAME).
APP	The application name where the signature option is configured to (MAXAPPS.APP).
METHODNAME	The name of the annotated method in the application service.
OPTIONNAME	This value is a combination of the signature option table for the application and the signature option (SIGOPTION.OPTIONNAME).

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Language support

Your database can contain some elements, such as descriptions and long descriptions, in multiple languages. Publish channels and enterprise services can include these translated columns. The integration framework also supports the use of bidirectional language formats.

A single database can contain data in multiple languages. You specify the base language when you install the application. If your system uses a language that differs from the base language, you can enable the integration framework to send and receive data that is not in the base language data.

Default processing of multiple languages

When you log in, you can choose a language code other than the base language for the system. In any application, you can then enter language-specific values for columns that are designated as translatable.

By default, outbound transactions contain the applicable column values in the language that is associated with the login session. The language values display whether the transaction is initiated by an application or the data export feature. For example, if the base language of your system is English, you can log in as a French user and update an item record with a French description. The outbound message contains the item description in French, even when the description also exists in English, or a third language.

If a database table contains translatable columns, the database contains a corresponding table called L_tablename, for example, ITEM and L_ITEM. The

L_tablename table stores the non-base language values for every translated column except the long description. Long descriptions in all languages are in the LONGDESCRIPTION table.

To include translated values in the output XML, include the L_tablename and LONGDESCRIPTION objects in the applicable object structures. Provide the base language values as a service input to object structures that have the L_tablename as part of their object definition. Your service input must be in the core object, and all other languages must be in the additional language enabled object.

For example, when English is the base language, the ITEM table contains the English description of an item and the L_ITEM table contains the French and German descriptions of that item. The LONGDESCRIPTION table contains the English, French, and German long descriptions.

When you add the L_tablename object to an object structure, assign the same value to the transLanguage and baseLanguage attributes. Otherwise, the base language values are not available and are processed for the multilanguage-enabled fields.

Multilanguage attributes

The root element of the XML structure for services and channels includes language attributes that specify the language attributes for the record.

The following language attributes are used:

- The `baseLanguage` attribute identifies the base language of the system or application that generates outbound XML. For inbound transactions (input XML), this attribute is not validated.
- Output XML includes the `langenabled` attribute on every translatable column, as shown in the following example:

  ```
  <DESCRIPTION langenabled="1">Item 1 description</DESCRIPTION>
  ```
- The `transLanguage` attribute identifies the language in which the values for applicable multilanguage fields are specified. If this attribute is missing or does not contain a value, all data is assumed to be in the base language. If the `transLanguage` value cannot be interpreted, or if the value does not identify a valid language, an error is returned to the service requester.

Bidirectional language support

The integration framework supports bidirectional languages such as Arabic and Hebrew and enables data exchange with external systems that use different bidirectional language formats. Bidirectional languages combine characters that you read from right to left with some characters, such as numbers or dates, that read from left to right.

The integration framework can transform the bidirectional formats of all inbound and outbound data to and from the bidirectional format specified for the external system. The transformations apply to synchronous and asynchronous communications for enterprise services, object structure services, and standard services. You can also transform the bidirectional formats of files you import with a cron task or when you use the data import feature.

Bidirectional language formats

The default bidirectional formats used by the integration framework are based on Unicode standards. These formats can differ from the formats used by external systems.

Bidirectional formats have five parameters.

Parameter	Value	Flags	Default	Details
Text Type	• Implicit • Visual	• I • V	I: Implicit (logical)	
Text Orientation	• LeftToRight • RightToRight • ContextualLeftToRight • ContextualRightToLeft	• L • R • C • D	L: LeftToRight	
Text Symmetric Swapping	• On • Off	• Y • N	Y: On	
Text Shaping	• Not Shaped • Shaped • Isolated	• N • S • B	N: Not Shaped	Arabic only
Numerals Shaping	• National • Nominal • ContextualNational • ContextualNominal	• H • N • C • T	N: Nominal	Arabic only

Configuring bidirectional language support for external systems

Specify the bidirectional language format used by an external system to enable the correct transformation of data to (and from) the default format.

Procedure

1. Optional: Select the **Bidirectional Format** action in the External Systems application for enterprise services or in the Invocation Channels application for object structure services.
2. Select the bidirectional language format used by the external system from the list of options and click **OK**.

Results

After setting the bidirectional language format used by an external system, transformation to the appropriate format occurs automatically during integration processing.

Exporting and importing file-based data

The integration administrator can initiate export and import data from within the External Systems application to support, for example, integrating data using files. The import process includes the ability to preview a data load from a file to validate the data prior to saving it to the database. The import process also includes an option to manage errors that result from file loading in the same file format as the imported file

You can use file-based error management as an alternative to managing errors through the Message Reprocessing application. Importing data from files, in flat file or XML format, can be done on a scheduled basis using a predefined cron task.

Additionally, administrators can enable an application for export and import, and users can then export and import data directly from within the enabled application.

Related concepts:

"Error management with file-based data import" on page 192
The integration framework supports error management with the Message Reprocessing application which allows you to review, correct, reprocess, and delete messages that go in error when processed from an inbound queue. When data is imported from an XML file or a flat file, a second option is available that manages errors using a downloaded file instead of the Message Reprocessing application.

Exporting and importing data in the External Systems application

You can initiate a data export in the **Publish Channels** tab of the External Systems application. You can initiate a data import in the **Enterprise Services** tab of the External Systems application.

Exporting file-based data

With the data export feature, you can perform a bulk export of message data from a file to an external system. You can initiate the export process for each publish channel that is associated to an external system.

Before you begin

In a multitenancy environment, you can use the data export feature only if the system provider provides you with access to a file server that is accessible by the application server. You must then configure a file-based endpoint to point to the location of this file server.

You must enable both the external system and the publish channel before you can export data. The data for export must either be in an XML file format that adheres to the object structure schema or in a delimited flat file, such as comma separated, that is a flattened version of the object structure schema format.

About this task

The optional SQL query that you enter in the **Export Condition** field, can affect the size of the exported XML message. You can filter the content to limit the amount of data that is being exported. The export process performs the standard outbound processing on the result set of the query for the selected publish channel.

Procedure

1. In the External Systems application, click the **Publish Channels** tab and select the publish channel that you want to export.
2. In the **End Point** field, specify a file-based endpoint handler, for either XML file or flat file format.
3. Click **Data Export**.
4. Optional: Enter a SQL query in the **Export Condition** field. The query must be against the primary or top-level object in the publish channel object structure.
5. Optional: Specify an integer value in the **Export Count** field to limit the number of records that are contained in the exported file. If the result of the query contains more records than the number you specify, those records are not included in the exported file.

6. Click **OK** to begin the data export process.

What to do next

When the data export is executed, the selected data is formed into a message and dropped into the outbound queue that is configured for the publish channel and its corresponding external system. The message is then processed from the outbound queue to the configured endpoint. If an error occurs when delivering a message to the endpoint, you can manage and view the data export messages that are flagged with an error in the Message Reprocessing application.

Importing file-based data

You can use the data import feature to load data from either XML or flat, delimited files, to update the Maximo database. You can preview and validate the data prior to loading it and committing it to the database. You can choose to manage errors with the Message Reprocessing application or by extracting errors to a file format that is the same as the imported file format.

Before you begin

Before data can be imported, if you plan to import data from a flat file, such as a .csv file, the enterprise service object structure must support flat file structures. Ensure that the **Support Flat File Structure** check box is selected on the associated object structure record in the Object Structures application. You also must enable both the external system and the enterprise service before you can import data.

About this task

The data that you import must be in a delimited flat file, such as comma separated, or an XML file format. The data import process can use a predefined or user-defined enterprise service.

Procedure

1. In the External Systems application, display the external system that contains the enterprise service from which you want to import data.
2. On the **Enterprise Services** tab, select the enterprise service from which you want to import data.
3. Click **Data Import**.
4. Optional: Select the **Import Preview** check box to examine the data before importing and committing the data to the database. Use the preview option to sample data records. This feature is not intended to support a large file containing hundreds of records. Processing synchronously processes the file to the business objects and returns any error messages encountered, without committing any updates to the database.
5. Specify the type of file that you want to use for the file import.

Option	Description
XML File	Imported data is in XML format.
Flat File	Imported data is in a delimited flat file. If necessary, modify the **Delimiter** and **Text Qualifier** values.

6. In the **Specify Import File** field, enter the file name path the imported file uses for identification and storage.

7. Select the **File-based Error Management** check box if you want to manage any errors you encounter through a file in the same format as the file being imported. This option is an alternative to managing errors with the Message Reprocessing application
8. Click **OK** to begin the import data process.

What to do next

When the data import is executed, the file that is selected for import is formed into multiple messages and dropped into the inbound queue that is configured for the enterprise service and its corresponding external system. The messages are then processed from the inbound queue to the application objects for updating. The processing of messages from an inbound queue requires the enablement of the JMS cron task when the sequential queue is used or the enablement of Message Driven Beans for the continuous queue. If errors occur when processing a file, you can manage and view the data import messages that are flagged with an error in the Message Reprocessing application.

Cron tasks for processing inbound data

If you want to schedule file loads rather than using the data import window, you must create, configure, and enable the XMLFILECONSUMER cron task and the FLATFILECONSUMER cron task before cron task message processing can occur.

XMLFILECONSUMER cron task

The XMLFILECONSUMER cron task is the mechanism that you use to load XML files without any application user intervention.

The XMLFILECONSUMER cron task has the following predefined parameters:

Parameter	Description
EXTERNALSYSTEM	A required parameter value that identifies the external system value that the cron task data load process uses.
SOURCEDIRECTORY	A required parameter value that defines the directory where source files are loaded. This directory must exist on the application server.
ENTERPRISESERVICE	A required parameter value that identifies the enterprise service name that the cron task uses to process all files that are located in the source directory.
FILENAME	An optional parameter value that determines whether the cron task data load process selects source files based on its file name.
USEREXITCLASS	An optional custom processing class value that defines additional cron task capability (for example, file load order).
TARGETENABLED	Ensure that the value is at the default of 0 (false). The functionality of this flag is superseded by the donotrun functionality. Use the donotrun parameter in the cron task framework to control which servers the cron task runs on.

Parameter	Description
ISFILEEXTRACT	When set to 1 (true), this parameter indicates that file-based error management is used for inbound message processing.

All XML files that are available in the source directory are associated with a specific external system – enterprise service when you add a value to the EXTERNALSYSTEM, SOURCEDIRECTORY, and the ENTERPRISESERVICE parameters. These XML files also are loaded into the integration framework.

The files that match the cron task file name property are associated with a specific external system – enterprise service and are loaded into the system when you add a value to the EXTERNALSYSTEM, SOURCEDIRECTORY, ENTERPRISESERVICE, and FILENAME parameters

FLATFILECONSUMER cron task

The FLATFILECONSUMER cron task is the mechanism that you use to load flat files without any application user intervention.

Cron task parameters

The FLATFILECONSUMER cron task has the following predefined parameters:

Table 33. FLATFILECONSUMER cron task parameters

Parameter	Description
SOURCEDIRECTORY	A required parameter value that defines the directory where source files are loaded. This directory must exist on the application server.
DELIMITER	A required parameter that indicates the character that is used as field delimiter in the flat file. The default value is , (comma).
USEREXITCLASS	A custom processing class value to enable additional cron task capability (for example file load order).
TEXTQUALIFIER	A required parameter value that defines the character that is used as text qualifier in the flat file. The default value is " (double quotation marks).
TARGETENENABLED	Ensure that the value is at the default of 0 (false). The functionality of this flag is superseded by the donotrun functionality. Use the donotrun parameter in the cron task framework to control which servers the cron task runs on.
ISFILEEXTRACT	When set to 1 (true), this parameter indicates that file-based error management is used for inbound message processing.

All files available at the source directory are processed by the FLATFILECONSUMER cron task when you add a value to the SOURCEDIRECTORY parameter. The external system and the enterprise service

are identified from the first record in the flat file that is imported, which is part of flat file definition.

Cron task processing properties

The FLATFILECONSUMER cron task uses the following processing properties for inbound messages:

- File processing order - The order in which files are loaded on to the server that is determined by the XML file time stamp. The cron task user exit class can be used to overwrite the inbound message processing logic.
- File split - Multi-noun files that are processed by the FLATFILECONSUMER cron task are split before they are written to the queue. The cron task identifies if the file that is loaded is a multi-noun XML file. If the XML file is a multi-noun file, the integration framework uses the enterprise service key columns to identify where the file split occurs. For example, PONUM and SITE in the MXPOInterface enterprise service.
- Queue processing - The FLATFILECONSUMER cron task identifies the queue in which the flat file is loaded. The location is based on the queue (continuous or sequential) specified at the external system and enterprise service level.

The cron task creates an index file (`recovery_filename.txt`) that contains a reference to the last successfully processed noun when you process a multi-noun file. The entry in the index file is updated when the noun is successfully committed to the queue. Index files are available in the RECOVERY folder which is created in the cron task source directory.

The FLATFILECONSUMER cron task uses the index file name to identify the file that was processed before the server or queue problem was encountered. The cron task continues to process the XML file starting at the last successfully committed noun in the index file. Errors that are identified after a message is successfully written to an inbound queue must be resolved in the Message Reprocessing application.

Configuring an application for data export and import

To enable application-based data import and export you must define the content of an object structure and enable this object structure for import and export.

Defining the object structure content

Application users can be granted access to export or import data directly from an application. An administrator can identify an appropriate object structure for an application and grant either export or import (or both) capabilities for users to manage application data in a file, such as a .csv file, that can be maintained in spreadsheet.

About this task

Determine the object structure to use for the integration transactions and then modify the content of the object structure to limit data exchange to the content that is required.

Procedure

1. In the Application Designer, open the application that you want to enable and note the value in the **Main Object** field in the header area. This value must be the main object that you use for the object structure.

2. In the Object Structures application, specify the name of the main object in the **Object Structure** field to filter for object structures based on this object. If the search does not return any object structures based on the main object, you must create an object structure and the following steps do not apply.
3. If the search returns multiple object structures, select an object structure where the value in the **Consumed By** field is set to Integration.
4. Review the content of the object structure and ask the following questions:
   a. Is the main object of the object structure the same as the main object of the application that you want to enable for data export and import?
   b. Does the object structure include child objects that contain data that users do not need to import or export? For example, the MXPERSON object includes the PHONE, EMAIL, and SMS child objects. Your users may want to import and export data for the PERSON and PHONE objects only.
5. After you review the content of the object structure, you can choose to use the predefined object structure with all of its existing content or to duplicate this object structure and modify its content. If you make changes to a predefined object structure, it can affect any other integration scenarios that use this object structure. Duplicating the object structure and using this as a template does not cause unexpected behavior for other integration users.
6. Assuming that you duplicate the object structure to provide a new object structure for this scenario, delete any child objects that are not necessary for your integration users.
7. For each object that remains in the object structure, select the **Include/Exclude Fields** option in the **Select Action** menu, and deselect all fields that users do not require to import or export. This step limits the data fields to only those needed by the application users.
8. If delimited flat files are required (for use in a spreadsheet), select the **Support Flat Structure** check box. This step activates a validation to ensure that every column for every object in the object structure has a unique name. If the validation fails, an **Alias Conflict** indicator is set.
9. If the object structure contains an alias conflict, select the **Add/Modify Alias** option in the **Select Action** menu to assign a unique alias for any field name where a duplicate exists. Alias conflicts do not exist in any of the predefined object structures.
10. Save the object structure.

What to do next

The content of the object structure is now configured and you can grant application access to it in the Object Structures application.

Enabling data import and export in an application

After you define the content of the object structure, you must enable an application for data import and export. Enabling data import is a separate procedure to enabling data export. Because the procedures are separate, you can enable some users to export data, enable other users to import data, or enable a separate set of users to import and export data.

Procedure

1. In the Object Structures application, select the object structure that you want to enable.
2. Select the **Add/Modify Application Export Support** action.

3. In the Add/Modify Application Export Support window, click **New Row**. In the Details section, the value in the **Application** field defaults to the application that has the same main object as the main object of the object structure.
4. Specify either **XML File** or **Flat File** as the format for data export files.
5. In the **Maximum Count** field, you can specify a value that limits the number of rows that can be exported during a single execution. Even if the user selects 1000 records, if the limit is set to 100, then only 100 rows are exported. A value of 0 or null allows the processing of an unlimited number of rows.
6. Select the **Is Default** check box if you enable multiple object structures for application export and you want this object structure to be the default.
7. Click **OK** to save the export configuration.
8. Select the **Add/Modify Application Import Support** action and repeat the same procedures to configure data import for the application.
9. Select the user group for the application users in the Security Groups application and grant access to the Application Export option, the Application Import option, or to both options.

Initiating data export and import in an application

After enabling data export and import in an application, icons are added to the toolbar of the application for starting the transactions.

Starting data export in an application:

After you enable data export for an application, users can initiate export transactions from within the application user interface.

Procedure

1. In the List tab of the application, select the data for export in one of the following ways:
   - Use the filter options to filter for a set of records to export, and click the **Application Export** icon in the toolbar.
   - Select a specific record to export, and click the **Application Export** icon. This option exports only the data for a single record.
2. In the Data Export window, review the export settings and make any necessary adjustments:
   a. Optional: Specify a different value in the **Object Structure** field if you do not want to use the default object structure for the export. You can specify an alternate object structure only if another object structure has been enabled for this application. If you specify an alternate object structure, the window refreshes to include the Export Configuration section where you can configure additional settings.
   b. Optional: If the value in the **Selected to Export** field exceeds the value in the **Export limit** field, you can cancel the export and use a different filter to reduce the number of records to export. The value in the **Export limit** field is set by the administrator, and cannot be exceeded, regardless of the number of records selected.
3. Optional: In the Export Configuration section, (expanded only if you specified an object structure that is not the default), specify the data format:
   a. If you select **XML** format, specify the operation for the export, for example Sync.
   b. If you select **Flat File** format, specify values in the **Delimiter** and **Text Qualifier** fields.

4. Click **OK** to start the export. A **Save** dialog box opens in which you can specify where to save the file. For a flat file to open in a spreadsheet application, you can change the filename to use the .csv suffix. If the Save dialog box does not open, change the security settings of your browser to enable automatic prompting for downloads.

Starting a data import in an application:

After you enable data import for an application, users can initiate import transactions from within the application user interface.

Procedure

1. From anywhere in the application, click the **Data Import** icon in the toolbar.
2. Optional: In the Data Import window, specify a different value in the **Object Structure** field if you do not want to use the default object structure for the import. If you specify an alternate object structure, the Import Configuration section expands, where you must perform additional configurations.
3. Optional: If you specified an alternate object structure, in the expanded Import Configuration section, specify the following values:
   a. If you specify **XML** format, no additional settings are required because the incoming XML file specifies the operation, language code and action code.
   b. If you specify the **Flat File** format, you can specify values in the **Delimiter**, **Text Qualifier**, **Action**, and **Language Code** fields, or you can use the default values.
4. Optional: Review the value in the **Import limit** field. You cannot change this value. If the limit is set, for example, to 100 records, and the import file contains 200 records, only the first 100 records are imported.
5. Click **Browse** to navigate to the import file, and select it.
6. Optional: If you select **Import Preview**, when you run the import, the file is processed by the business objects but the data is not saved to the database. You can use this option to test the data load and validate that there are no errors prior to loading. Any errors are displayed and you can apply corrections to the input file and attempt the import again.
7. Click **OK** to start the import. Because data import is a synchronous process, you must wait in the user interface until the load is complete and a confirmation message appears. If the load encounters errors, no data is loaded, as the file is processed as a single transaction with a single commit.

Federated MBO integration

Maximo Integration Framework supports the sharing of data between Maximo Asset Management and external systems in several ways.

The most common way to share information using Maximo Integration Framework is to replicate data between the systems. A federated resource can share system data without replicating external system data in Maximo Asset Management.

MBOs are a set of Java classes that implement the data persistence layer and business rules in Maximo Asset Management. With federated MBO integration, you can associate Maximo Asset Management objects with data from an external system, and persist the data outside of the Maximo Asset Management database. The external data must be exposed through a JSON REST API in order for the data to be available as an MBO.

External data can be registered using the Maximo Asset Management JSON Resources application. External systems can share data with Maximo Asset Management when needed rather than replicating the information into the Maximo Asset Management environment. The federated MBO retrieves the most up to date version of the data from an external application.

Integration APIs

The Maximo Integration Framework supports the use of REST (Representational State Transfer) application programming interface (API), and the sharing of lifecycle data between applications based on Open Services for Lifecycle Collaboration (OSLC) integration.

REST/JSON API enhancements

The REST APIs that were added in IBM Maximo Asset Management version 7.6.0.2 provide enhanced support for querying Maximo Asset Management data.

The new REST APIs for Maximo Asset Management are a rewrite of the existing REST APIs that were released after Maximo Asset Management version 7.1 and are integrated in Maximo Asset Management releases starting with version 7.6.0.2.

The new REST API resources evolved from the OSLC REST API and provide similar capabilities but remove the requirement to use OSLC namespaces. The new REST API is sometimes referred to as the REST/JSON API because of the end-to-end support of the JSON data format.

The following enhanced support and capabilities are available with the new REST APIs:

- Related object queries, multi-attribute text search, and custom queries
- Bookmarking, notifications, and image association
- Metadata support that uses the JSON schema
- Dynamic query views
- GROUP BY queries
- Custom JSON elements that are appended to the object structure JSON
- Access to Swagger documentation for APIs
- Integration with the Maximo Asset Management cache framework

The new REST APIs can be used when a Maximo Asset Management package that is not customized is installed and set up with users and groups.

For more information about the new REST APIs, see https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20Maximo%20Asset%20Management/page/Maximo%20JSON%20API on IBM developerWorks®.

REST API

The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

The REST API is part of the integration framework, and you can use it to integrate external applications with process automation engine applications. The REST API exposes business objects and integration object structures as REST resources. The

REST API can provide resource data in either XML or JavaScript Object Notation (JSON) format. External applications can use the REST API to query and update application data.

REST API resources can be used without any configuration. The object structures that the REST API can query or update have a value of INTEGRATION in the **Consumed By** field in the Object Structures application.

The REST API supports create, query, update, and delete operations on resources by using standard HTTP GET, POST, PUT, and DELETE methods.

Related information:

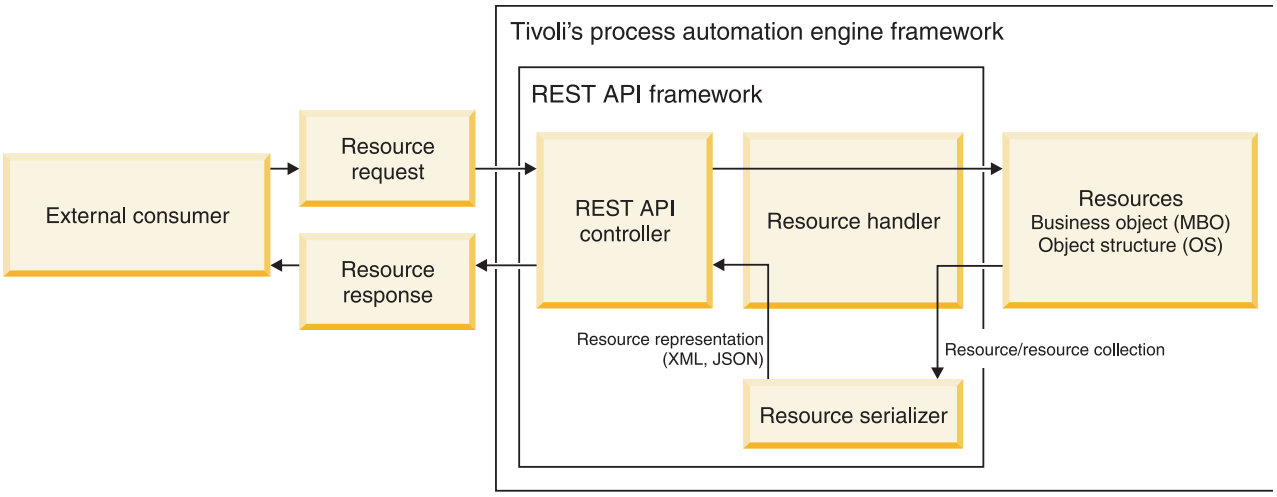
Using REST Api Details (opens in a new browser window or tab)

RESTing with Maximo (opens in a new browser window or tab)

REST API framework

The REST API is part of the integration framework and handles requests from external consumers.

The following diagram provides an overview of how the REST API handles requests.



When an external consumer initiates a request, a REST API controller directs the resource request to the appropriate resource handler. The resource handler interacts with resources to execute the request. When the request is complete, the resulting resource or resource collection is processed by the resource serializer. The resource serializer returns the representation as the resource response. Serializers are provided for XML and JSON.

Because the REST API is within the integration framework, the REST API can use process automation engine authentication, authorization, and system properties.

The REST API controller is a servlet. Performance and tuning for load and scalability is done at the application server level as it is for other web components.

Supported representations

The REST API supports XML and JSON as representations. Representations are supported by the implementation of serializer classes that are registered in system properties.

The XML and JSON representations are registered in the following REST API system properties:

- **mxe.rest.serializer.mbo.xml**
- **mxe.rest.serializer.os.xml**
- **mxe.rest.serializer.mbo.json**
- **mxe.rest.serializer.os.json**

You can extend serializers for custom processing. You can create serializers for existing resource types, and for individual instances of business object resources or object structure resources.

For object structure resources, the XML format for the REST API is similar to the format that is supported by the integration framework.

Related reference:

"REST system properties" on page 257
System properties are available to configure how the REST API works for your specific requirements.

Resource handlers and URIs

The REST API provides access to business objects and integration object structures. A handler class for each resource is registered in a system property.

The handler classes for the business object and object structure resources are registered in the **mxe.rest.handler.mbo** system property and the **mxe.rest.handler.os** system property.

You can extend handlers for custom processing. You can create handlers for other resource types, and for individual instances of business object resources or object structure resources.

REST API resources are addressed by using resource identifiers that are part of the Uniform Resource Identifier (URI). The URIs are used to address the entities that are represented as REST resources.

For example, the following URI accesses a collection of Person business objects. The `mbo` indicates the business object resource type and `person` is the resource:

```
.../maxrest/rest/mbo/person
```

For another example, the following URI accesses a collection of data for the Person object structure. The `os` indicates the object structure resource type and `mxperson` is the resource:

```
.../maxrest/rest/os/mxperson
```

GET method

Use the GET method to retrieve business object resources and object structure resources.

Syntax

```
GET uri?parameter=value&parameter=value&...
```

uri is a URI. The length of the URI path cannot exceed any system specified limit.

parameter is a query parameter.

value is the value of the query parameter.

Example: Retrieving a business object resource

The following method retrieves a business object resource:

```
GET /maxrest/rest/mbo/asset/123
```

The following XML is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSETNUM>11200</ASSETNUM>
    <SERIALNUM>3481-52</SERIALNUM>
    <ASSETTAG>3751</ASSETTAG>
    <LOCATION>BR200</LOCATION>
    <DESCRIPTION>HVAC System- 50 Ton Cool Cap/ 450000 Btu Heat Cap</DESCRIPTION>
    <ASSETUID>123</ASSETUID>
    .
    .
  </ASSET>
```

The resource response is shown in XML. The default format setting can be configured in the **mxe.rest.mbo.defaultformat** system property.

Example: Retrieving an object structure resource

The following method retrieves an object structure resource:

```
GET /maxrest/rest/os/mxasset/123
```

The following XML is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <QueryMXASSETResponse xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDateTime="2011-04-07T11:43:59-04:00"
    transLanguage="EN" baseLanguage="EN" messageID="1302195161355515345"
    maximoVersion="7 5 20110405-0030 V7500-718" rsStart="0" rsTotal="1"
    rsCount="1">
    <MXASSETSet>
      <ASSET>
        <ASSETNUM>11200</ASSETNUM>
        <SERIALNUM>3481-52</SERIALNUM>
        <ASSETTAG>3751</ASSETTAG>
        <LOCATION>BR200</LOCATION>
        <DESCRIPTION>HVAC System- 50 Ton Cool Cap/ 450000 Btu Heat Cap</DESCRIPTION>
        <ASSETUID>123</ASSETUID>
        .
        .
        <ASSETMETER>
          <ACTIVE>1</ACTIVE>
          <ASSETMETERID>63</ASSETMETERID>
          <LASTREADING>0</LASTREADING>
          <METERNAME>PRESSURE</METERNAME>
          .
          .
        </ASSETMETER>
        <ASSETSPEC>
          <ASSETATTRID>SPEED</ASSETATTRID>
          <ASSETSPECID>2138</ASSETSPECID>
          .
          .
        </ASSETSPEC>
        <ASSETSPEC>
```

```
          <ASSETATTRID>SHAFTDIA</ASSETATTRID>
          <ASSETSPECID>2139</ASSETSPECID>
          .
          .
        </ASSETSPEC>
      </ASSET>
    </MXASSETSet>
  </QueryMXASSETResponse>
```

The object structure resource data includes asset, asset meter, and asset specification data based on the configuration of the objects within the MXASSET object structure. Any fields that are configured to be included or excluded are implemented in the data.

The URI path can contain the ID of the resource as a way to select a specific resource. In the preceding example, the ID is 123. The ID value is a unique ID for the object that is registered in the MAXATTRIBUTE table. If no ID is provided, the response includes all occurrences of the resource, depending on the security restrictions on the user. The ID is the only business object field that can be part of the URI path for selection criteria. All other fields can be used as a query parameter in the URI for selection filtering.

Example: Selecting related child resources

The following method requests a PO resource for a business object resource:

```
GET /maxrest/rest/mbo/po/13
```

The following XML is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <PO xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <PONUM>1005</PONUM>
    <DESCRIPTION>Electrical Supplies</DESCRIPTION>
    .
    .
  </PO>
```

Example: Selecting related PO lines of a PO

The following method requests all of the PO lines that are related to the PO that is selected in the previous example:

```
GET /maxrest/rest/mbo/po/13/poline
```

The following XML is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <POLINEMboSet rsStart="0" rsTotal="2" rsCount="2"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <POLINE xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <PONUM>1005</PONUM>
      <ITEMNUM>11241</ITEMNUM>
      <STORELOC>CENTRAL</STORELOC>
      <ORDERQTY>3.0</ORDERQTY>
      <POLINENUM>1</POLINENUM>
      <POLINEID>10051</POLINEID>
      .
      .
    </POLINE>
    <POLINE xmlns="http://www.ibm.com/maximo"
```

```
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <PONUM>1005</PONUM>
      <ITEMNUM>29331</ITEMNUM>
      <STORELOC>PKG</STORELOC>
      <ORDERQTY>2.0</ORDERQTY>
      <POLINENUM>2</POLINENUM>
      <POLINEID>10052</POLINEID>
      .
      .
    </POLINE>
  </POLINEMboSet>
```

If you select POLINE, the relationship name that is specified in the URI (POLINE) identifies the path from the business object (PO) to a related business object (POLINE). In this example, the business object and the relationship have the same name. An object can have many relationships between two objects that can result in different data being selected for the related object. Relationships are defined in the MAXRELATIONSHIP table.

To retrieve only the line of a specific POLINE business object, include the ID of the POLINE business object in the URI:

```
GET /maxrest/rest/mbo/po13/poline/10052
```

In this example, `poline` represents the relationship name and `10052` is the ID of the POLINE business object that the relationship retrieves. You can use the same approach to specify the path from POLINE to ITEM or POCOST:

```
GET /maxrest/rest/mbo/po/13/poline/10052/item
GET /maxrest/rest/mbo/po/13/poline/10052/pocost
```

When you use the object structure resource, child objects cannot have relationships. The object structure supports a graph of related business objects. The relationships are part of the object structure definition.

You can traverse related business objects to move from one resource to another if a valid relationship exists between the source and target object.

Related reference:

"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

Query parameters and operators:

To fetch a resource whose unique ID is known, you can supply the ID of the resource as part of the URI path. You can also use query parameters to filter and fetch resource collections.

In a resource collection, the representation has a pointer to the unique ID of each resource. The ID is used by the consuming software to create the link to the resource. The link is used to update, delete, and query the resource.

A RESTful interaction often begins by fetching a resource collection and then drilling down to a particular resource in the collection.

You can use any field of a business object resource or a related business object resource as a query parameter. The business object query by example (QBE) framework restricts the relationship between the business objects to one level deep. Any field of an object structure resource can be used as a query parameter if the field belongs to an object that resides in the top two levels of the object structure.

If you use a business object attribute as a query parameter, the following operators can be used as part of the parameter.

Operator	Notation	Example
Equals	~eq~	status=~eq~APPR Use the equals operator to perform an exact match. A status=APPR query implies a like operator comparison. If you use status=APPR &_exactmatch=1, the processor is forced to do an exact match. This notation produces the same result as status=~eq~APPR.
Not equals	~neq~	status=~neq~APPR
Greater than	~gt~	quantity=~gt~2.5
Greater than equals	~gteq~	quantity=~gteq~2.5
Less than	~lt~	quantity=~lt~2.5
Less than equals	~lteq~	quantity=~lteq~2.5
Ends with	~ew~	description=~ew~APPR
Starts with	~sw~	description=~sw~APPR
Like	No notation required	description=APPR

Example: Querying by location

The following example queries for a location by using the location number as a query parameter without an operator:

```
GET maxrest/rest/mbo/locations?location=PT100
```

The following XML is returned by the query:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <LOCATIONSMboSet rsStart="0" rsTotal="2" rsCount="2"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <LOCATIONS>
      <LOCATION>PT100</LOCATION>
      <DESCRIPTION>PT100</DESCRIPTION>
      .
      .
    </LOCATIONS>
    <LOCATIONS>
      <LOCATION>PT1001</LOCATION>
      <DESCRIPTION>1001</DESCRIPTION>
      .
      .
    </LOCATIONS>
  </LOCATIONSMboSet>
```

Because no operator is specified, a like comparison is used, and the result has two locations. If the request uses the equals operator, only location PT100 is returned for the query request, because the operator enforces an exact match:

```
GET /maxrest/rest/mbo/locations?location=~eq~PT100
```

Example: Using multiple parameters

The following example shows the use of multiple parameters to query for an Issue YTD quantity that is greater than 1 and an order quantity that is greater than 5:

```
GET /maxrest/rest/mbo/inventory?issueytd=~gt~1&orderqty=~gt~5
```

Example: Using the ormode condition

You can request multiple values for a single field by using the `ormode` condition. The following example selects records where the issue YTD quantity is equal to 7 or less than 4:

```
GET /maxrest/rest/mbo/inventory?issueytd.ormode=~eq~7&issueytd.ormode=~lt~4
```

Example: No records selected

If a query request results in no records being selected, an empty result set (not an exception condition) is returned.

For example, the following request is for records that have an Issue YTD quantity of 99:

```
GET /maxrest/rest/mbo/inventory?issueytd=~eq~99
```

No records are selected as a result of the request, and therefore the following empty result is returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <INVENTORYMboSet rsStart="0" rsTotal="0" rsCount="0"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
```

_opmodeor parameter:

Use the **_opmodeor** parameter to evaluate multiple fields by using an OR condition between values instead of the default AND condition.

For example, the following request returns records where the issue YTD quantity is greater than 1 and the order quantity is greater than 5:

```
GET /maxrest/rest/mbo/inventory?issueytd=~gt~1&orderqty=~gt~5
```

The query selects six records, as shown in the rsTotal attribute:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <INVENTORYMboSet rsStart="0" rsTotal="6" rsCount="6"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <INVENTORY xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ITEMSETID>SET1</ITEMSETID>
      <MANUFACTURER>WES</MANUFACTURER>
      <SHRINKAGEACC>
        <VALUE>6600-810-800</VALUE>
        </INVENTORY>
        .
        .
    <INVENTORY xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    .
    .
```

The following modified request sets the **_opmodeor** parameter to true to return records where the issue YTD quantity is greater than 1, or the order quantity is greater than 5:

```
GET  /maxrest/rest/mbo/inventory?issueytd=~gt~1&orderqty=~gt~5&_opmodeor=1
```

The modified query selects 54 records:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <INVENTORYMboSet rsStart="0" rsTotal="54" rsCount="54"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <INVENTORY xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    .
    .
```

_rsStart and _maxItems parameters:

The **_rsStart** and **_maxItems** parameters are used together to control the number of records returned for a request and to allow paging through a large volume of records.

Example: Select all asset resources

The following request selects all the asset resources in the system:

```
GET maxrest/rest/mbo/asset?&_opmodeor=1
```

In this example, 757 assets are selected:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="0" rsTotal="757" rsCount="757"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>CAL100</ASSETNUM>
```

In the following request, the **_maxItems** parameter limits the number of rows selected. The number of rows is limited to 20, as shown by the rsCount value:

```
GET /maxrest/rest/mbo/asset?_maxItems=20
```

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="0" rsTotal="757" rsCount="20"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

To retrieve the next 20 rows, the following request adds the **_rsStart** parameter:

```
GET /maxrest/rest/mbo/asset?_maxItems=20&_rsStart=20
```

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="20" rsTotal="757" rsCount="20"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

The next 20 rows are retrieved, starting at row 20 and ending at row 39.

Related concepts:

"In-session scrolling" on page 240
Holding a resource collection in memory can improve performance when scrolling

through pages of data.

_orderbyasc parameter:

The **_orderbyasc** parameter controls the sort order of returned data.

The attributes that are used for the order by clause are shown in ascending order. Provide a comma-separated list of field names for the resource object to sort the result.

Use the **_orderbydesc** parameter in a similar way to sort in descending order.

Example: Sorting a response

The following request does not include the **_orderbyasc** parameter:

```
GET /maxrest/rest/mbo/asset?_maxItems=20&_rsStart=20
```

The following assets are returned:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="20" rsTotal="757" rsCount="20"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>7400</ASSETNUM>
      <SERIALNUM>A5252525555</SERIALNUM>
      <LOCATION>HWSTOCK</LOCATION>
      .
      .
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>7300</ASSETNUM>
      <SERIALNUM>A6687688888</SERIALNUM>
      <LOCATION>HWSTOCK</LOCATION>
      .
      .
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>7200</ASSETNUM>
      <SERIALNUM>A6387683888</SERIALNUM>
      <LOCATION>HARDWARE</LOCATION>
      .
      .
```

The following request includes the **_orderbyasc** parameter by specifying the location field of the asset business object:

```
GET /maxrest/rest/mbo/asset?_maxItems=20&_rsStart=20&_orderbyasc=location
```

The modified request returns the following assets in the requested sort order:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="20" rsTotal="757" rsCount="20"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>2045</ASSETNUM>
      <PARENT>19998</PARENT>
      <LOCATION>5THFLSWSTOCK</LOCATION>
      .
      .
    <ASSET xmlns="http://www.ibm.com/maximo"
```

```
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>2002</ASSETNUM>
      <SERIALNUM>K6LQI</SERIALNUM>
      <LOCATION>6THFLOOR</LOCATION>
      .
      .
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <ASSETNUM>2074</ASSETNUM>
      <PARENT>2002</PARENT>
      <LOCATION>6THFLOOR</LOCATION>
      .
      .
```

_includecols and _excludecols parameters:

Use the **_includecols** and **_excludecols** parameters to control the content of attributes that are returned in response to a query.

If neither parameter is specified, all of the attributes are returned for a business object resource. The **_includecols** and **_excludecols** parameters are only valid for a business object resource.

Example: Retrieving specific fields

If you use the **_includecols** parameter, only the attributes that are listed for the parameter are returned:

```
GET /maxrest/rest/mbo/asset?_includecols=assetnum,serialnum
```

The request returns only the asset number and serial number:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <ASSETMboSet rsStart="0" rsTotal="757" rsCount="757"
    xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <SERIALNUM>32R5G</SERIALNUM>
      <ASSETNUM>13150</ASSETNUM>
    </ASSET>
    <ASSET xmlns="http://www.ibm.com/maximo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <SERIALNUM>2342VV</SERIALNUM>
      <ASSETNUM>13160</ASSETNUM>
    </ASSET>
```

Example: Excluding a field

The **_excludecols** parameter identifies a list of business object fields to exclude from the query response:

```
GET /maxrest/rest/mbo/asset?_excludecols=serialnum
```

In this example, the request returns all of the fields of the asset resource except the serial number.

_dropnulls parameter:

Set the **_dropnulls** parameter to false to include fields that have a null value in the query response.

All resource fields are returned by a query unless they are restricted by other parameters or by the object structure configuration. If the **_dropnulls** parameter is not provided, null fields are dropped from the query response.

Example: Including fields that have null values

The following query includes the parameter:

```
GET  /maxrest/rest/os/mxasset?_dropnulls=0&_maxItems=1
```

The query returns null values as shown in the following XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
  <QueryMXASSETResponse xmlns="http://www.ibm.com/maximo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    creationDateTime="2011-04-08T09:25:11-04:00" transLanguage="EN"
    baseLanguage="EN" messageID="1302273234298496774"
    maximoVersion="7 5 20110405-0030 V7500-718" rsStart="0"
    rsTotal="757" rsCount="1">
    <MXASSETSet>
    <ASSET>
      <ANCESTOR />
      <ASSETID>1579</ASSETID>
      <ASSETNUM>CAL100</ASSETNUM>
      <ASSETTAG />
      <ASSETTYPE />
      <AUTOWOGEN>0</AUTOWOGEN>
      <BINNUM />
      <BUDGETCOST>0.0</BUDGETCOST>
      <CALNUM />
      <CHANGEBY>WILSON</CHANGEBY>
      .
      .
```

_format and _compact parameters:

The REST API supports XML and JSON representations (formats). You can use the **_format** parameter or content negotiation to specify a resource representation. You can use the **_compact** parameter to specify compact JSON format.

If a request does not specify a format, the default representation is used. The default is defined by the **mxe.rest.mbo.defaultformat** system property.

If a request specifies the **_format** parameter, the value is validated against the list of supported formats in the **mxe.rest.supportedformats** system property. If the requested format matches one of the supported formats, the requested format is used as the response format. If the requested format does not match the supported format list, an HTTP 406 error code is generated.

The **_compact** Boolean query parameter applies to the JSON format when the formatted response can be returned in a compact style. The structure of the response data is simplified and no metadata is returned for individual field values.

Example: Specifying JSON representation

The following request is for an Asset object in JSON representation:

```
GET  /maxrest/rest/mbo/asset/123?_format=json
```

The following response is returned:

```
{
  ASSET: {
    Attributes: {
      ASSETNUM: {
      content: "11200"
      }
      SERIALNUM: {
      content: "3481-52"
      }
      ASSETTAG: {
      content: "3751"
      }
      LOCATION: {
      content: "BR200"
      }
      DESCRIPTION: {
      content: "HVAC System- 50 Ton Cool Cap/ 450000 Btu Heat Cap"
      }
```

Example: Specifying compact format

The following request specifies compact format:

```
GET /maxrest/rest/mbo/asset/123?_format=json&_compact=1
```

This following response is returned:

```
{
  ASSET: {
    ASSETNUM: "11200"
    SERIALNUM: "3481-52"
    ASSETTAG: "3751"
    LOCATION: "BR200"
    DESCRIPTION: "HVAC System- 50 Ton Cool Cap/ 450000 Btu Heat Cap"
    VENDOR: "TRN"
    MANUFACTURER: "TRN"
    .
    .
```

Related concepts:

"Content negotiation of representations"
The REST API supports identification of the representation format by using content negotiation as an alternative to using the **_format** query parameter.

Content negotiation of representations:

The REST API supports identification of the representation format by using content negotiation as an alternative to using the **_format** query parameter.

For XML and JSON formats, the supported mimetype values are defined in the following system properties:

Property	Value
mxe.rest.format.json.mimetypes	application/json
mxe.rest.format.xml.mimetypes	application/xml,text/xml

The format is an alias for one or many mime types and is used to identify the serializer for a requested mime type. The REST API uses the **Accept** HTTP header value for content negotiation. The REST API matches the header value to one of the specified mime types that are defined in the system property. If there is a match, the REST API uses the value as the corresponding format for the response format. If there is no match with the supported format list, an HTTP 406 error code is generated.

The XML and JSON serializers always reply with the values `application/xml` for the XML representation and `application/json` for the JSON representation in the **Content-Type** header property. If you create a serializer to support another format, add your format to the list in the **mxe.rest.supportedformats** system property.

The values in the **mxe.rest.supportedformats** system property are related to the values that are provided in individual properties for the corresponding supported mime type values, such as **mxe.rest.format.NEW.mimetypes**. The mime type values are provided in the request through the **Accept** header property.

For example, the following system properties are used to add HTML as a resource representation format:

Property	Value
mxe.rest.supportedformats	`html,xml,json,image`
mxe.rest.format.html.mimetypes	`text/html`
mxe.rest.mbo.serializer	`com.ibm.tivoli.maximo.rest.MboHTMLSerializer`
mxe.rest.os.serializer	`com.ibm.tivoli.maximo.rest.OSHTMLSerializer`

When the **_format** parameter and the **Accept** header property are both provided for a query, the query parameter takes precedence.

Related concepts:

"_format and _compact parameters" on page 238
The REST API supports XML and JSON representations (formats). You can use the **_format** parameter or content negotiation to specify a resource representation. You can use the **_compact** parameter to specify compact JSON format.

In-session scrolling:

Holding a resource collection in memory can improve performance when scrolling through pages of data.

If you specify the **_usc** parameter with a value of 1 in a query, the server holds the resource collection in memory. The browser can request additional pages of data from memory instead of reselecting the data from the business object. The HTTP response includes the ID of the resource collection in the RLID header property.

Subsequent requests for additional pages of data can specify the **_maxitem** parameter, **_rsStart** parameter, and **_rlid** parameter, which holds the ID of the resource collection. The server retrieves the next page from the data that is held in memory for the specified ID.

The server cache data is held until the browser session ends or until the browser releases the data. For example, the data is released if the ID of the resource is specified by the **_rlid** parameter and the **_rlrq** parameter is set to true.

As an alternative to using the **_maxitems** and **_rsStart** parameters for in-session scrolling, you can create a **mxe.rest.***handler***.***resource***.pagesize** system property and assign to it a number of resources for each page of data. You can then use the **_page** query parameter in the request to select a specific page of data, such as page 3.

For example, if you create the **mxe.rest.mbo.asset.pagesize** system property and assign it the value 35, 35 assets are returned for each page of data.

Related concepts:

"_rsStart and _maxItems parameters" on page 235
The **_rsStart** and **_maxItems** parameters are used together to control the number of records returned for a request and to allow paging through a large volume of records.

Related reference:

"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

"REST query parameters" on page 254
By using query parameters, you can tailor and filter the responses.

Caching of GET requests:

By enabling the caching of GET requests, you can improve the response times of requests for resource data that were previously submitted by the same user.

When caching is enabled, the data is retrieved from the browser cache instead of from the business object on the server. Caching is used for a request if the entity tag (ETag) value of the request matches the value of the previous request. Caching is unique to each user and is controlled by the **Cache-Control** HTTP header property.

Enabling caching

You enable caching for individual resources by creating system properties. The system properties must be in the form **mxe.rest.***handler***.***resource***.cache**, where *handler* is `mbo` or `os` and *resource* is the name of the business object or object structure. The system properties must have a Boolean value of 1.

For example, the system property **mxe.rest.mbo.workorder.cache** set to a value of 1 enables caching for the work order business object.

Entity tag calculations and comparisons

When an initial request is made for a resource and caching is enabled for the resource, the framework generates an ETag value for the resource response. The ETag is calculated based on the MBO rowstamp attribute, which changes any time an MBO is modified. For a resource collection, the ETag is calculated based on a hash of the collection of rowstamp values from the MBOs in the collection.

For example, the response contains the following in the HTTP header:

```
HTTP/1.1 200 OK
ETag: "123456789"
Cache-control: private
Content-Length: 12195
```

The **Cache-control** is marked as `private` to ensure that only the current user can reuse the contents of the cache.

When the user makes a repeat request for the same resource or collection, the request is similar to the following:

```
GET maxrest/rest/mbo/po HTTP/1.1
Host: x.y
If-None-Match: "123456789"
```

The **If-None-Match** value is the ETag value from the previous request. The framework reads the **If-None-Match** header value and compares it to the new calculated ETag value for the resource or collection for the second request. If the values match, the server responds with an HTTP 304, implying that the client cache content is valid and can be used. The response includes the following line:

```
HTTP/1.1 304 Not Modified
```

If the ETag value does not match the **If-None-Match** header value of the request, the resource representation is constructed and sent to the client with a new ETag value to be used by the client for subsequent requests for the resource or collection:

```
HTTP/1.1 200 OK
ETag: "98999999"
Cache-control: private
Content-Length: 50004
```

Pessimistic caching mode

The default caching mode is pessimistic caching. In pessimistic caching mode, requests are serialized but the data is not sent from the server to the client when the ETag value of the request matches the value of the previous request.

Optimistic caching mode

You can enable optimistic caching by setting the **mxe.rest.***handler***.***resource***.optimistic** system property to 1. In optimistic caching mode, the ETag value is calculated before any data serialization occurs. Although optimistic caching can improve response times significantly because no serialization occurs, it requires more memory, especially for queries that request many records.

By default, optimistic caching determines the ETag value from the root object of an object structure. Therefore, if an object structure has related objects that do not update the root object, the ETag value might not be accurate. You can ensure that the ETag value is determined from all the objects in the object structure, including related objects, by setting the **mxe.rest.***handler***.***resource***.deepetag** system property to 1. However, by using this system property, the performance improvement is not as great and more memory is used.

Cache time limits

You can configure the length of time that the cache is valid for by setting the **mxe.rest.***handler***.***resource***.maxage** system property to a value measured in seconds.

Related reference:

"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

"REST system properties" on page 257
System properties are available to configure how the REST API works for your specific requirements.

PUT, POST, and DELETE methods

You can modify resources by using the HTTP PUT, POST, and DELETE methods.

The PUT, POST, and DELETE methods can be used to modify business object resources and object structure resources. However, the business rules of an object might prevent it from being updated by a REST API request. For example, a DELETE request on a work order resource might fail if the business object validations prevent deletions because of the current state of the work order.

Syntax

```
method uri HTTP/1.1
parameter=value&parameter=value&...
```

method is PUT, POST, or DELETE

uri is a URI and any associated query parameters

parameter is a parameter of the resource that is updated

value is the value of the parameter

PUT method:

Use the PUT method to update or insert a resource. An update request must provide the unique ID of the resource. To update an object structure resource, the ID of the main object is required.

To update or insert an object structure resource, you can specify the **_action** parameter to identify actions that the integration framework provides, such as the Change or AddChange actions.

Example: Updating an asset

The following method updates the asset that has an ID of 1234. The asset description is changed to `my_new_description` and the type is changed to `OPERATING`:

```
PUT maxrest/rest/mbo/asset/1234 HTTP/1.1
description=my_new_description&type=OPERATING
```

Example: Updating a child object

The following method updates the `assetspec` business object, which is a child of `asset`. The value is changed to `new_value` and the description is changed to `my_new_description`:

```
PUT maxrest/rest/mbo/assetspec/5678 HTTP/1.1
value=my_new_value&description=my_new_description
```

Example: Establishing the context of a parent object

To insert or update a child business object, you might need to establish the context of the parent business object context, such as in the following example in which a PO line description is updated and the ID of the PO is provided for context:

```
PUT maxrest/rest/mbo/po/1234/poline/5678 HTTP/1.1
description=my_new_description
```

POST method:

Use the POST method to update or insert a resource.

To update a resource, you must specify the ID of the resource. To create a resource, you must specify the primary key and all the required fields that do not have a default value, but no ID is required.

To update or insert an object structure resource, you can specify the **_action** parameter to identify actions that the integration framework provides, such as Change or AddChange.

To update or insert a child object of an object structure resource, the form data must identify each occurrence of the child object.

When you use the POST method to create a resource, specify the **_ulcr** query parameter with a value of 1 so that the response includes a link for the client to access the new resource. Otherwise, the content of the resource is included in the response. The link is included in the **Location** header property and the HTTP response code is 201 to identify that a link is provided instead of the data.

Example: Inserting an asset

The following method inserts asset 127 within the BEDFORD site by using a business object resource. The asset and site make up the primary key.

```
POST maxrest/rest/mbo/asset HTTP/1.1
assetnum=127&siteid=BEDFORD&description=my_new_description&type=OPERATING
```

Example: Updating an asset by specifying its ID

The following method updates an asset by providing the ID as part of the URI:

```
POST maxrest/rest/mbo/asset/1234 HTTP/1.1
description=my_new_description&type=OPERATING
```

Example: Specifying an action

The following method uses the **_action** parameter to specify a Change action:

```
POST maxrest/rest/mbo/asset/968 HTTP/1.1
_action=Change&description=my_new_description
```

Example: Updating a record that has multiple child objects

The following method adds two purchase order lines to a purchase order, and each line has two purchase order costs:

```
POST maxrest/rest/os/mxpo/1234 HTTP/1.1
description=new_po_desc&
poline.id1.polinenum=1&poline.id1.item=ABC&poline.id1.description=new_description&
poline.id1.pocost.id1-1.costlinenum=1&poline.id1.pocost.id1-1.gldebitacct=new_gl_acct_a&
poline.id1.pocost.id1-2.costlinenum=2&poline.id1.pocost.id1-2.gldebitacct=new_gl_acct_b&
poline.id2.polinenum=2&poline.id2.item=XYZ&poline.id2.description=new_description&
poline.id2.pocost.id2-1.costlinenum=1&poline.id2.pocost.id2-1.gldebitacct=new_gl_acct_c&
poline.id2.pocost.id2-2.costlinenum=2&poline.id2.pocost.id2-2.gldebitacct=new_gl_acct_d&
```

In the example, the identifiers identify the following parameters:

Group identifier	Parameters identified
id1	Parameters that belong to poline 1
id1-1	Parameters that belong to pocost 1 for poline 1
id1-2	Parameters that belong to pocost 2 for poline 1
id2	Parameters that belong to poline 2
id2-1	Parameters that belong to pocost 1 for poline 2
id2-2	Parameters that belong to pocost 2 for poline 2

Related reference:
"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

DELETE method:

The DELETE method requires the unique ID of the resource.

To delete an object structure resource, the ID of the main object is required.

Example: Deleting an asset

The following method deletes asset 1234:

```
DELETE maxrest/rest/mbo/asset/1234 HTTP/1.1
```

Concurrent updates of resources:

Processing can be controlled so that a resource is updated or deleted only if the resource has not been changed by another user or application after the initial query.

If the **_urs** parameter is set to a value of 1, the query response includes the rowstamp value for the resource.

XML example:

```
<PO rowstamp=1234567890>
```

JSON example:

```
{"ASSET":{"rowstamp":"1234567890","Attributes":{"ASSETNUM":{"content":"1001"}
```

JSON example in compact format:

```
{"ASSET":{"rowstamp":"1234567890","ASSETNUM":"1001",
```

For an object structure resource, a rowstamp value is provided for each object that is in the object structure.

On a subsequent request to update the resource, the rowstamp value can be provided by using the **_rowstamp** parameter for business object and object structure resources, or by specifying the **If-Match** header property for business objects.

During processing, the rowstamp value is compared to the initial rowstamp. If the values do not match, the update or delete operation fails and an HTTP return code of 412 is in the response.

Service method queries and updates

The REST API can be used to call methods that are exposed by application services.

Methods that are exposed by application services are typically marked with the Java Specification Request (JSR) 181 @WebMethod annotation, which exposes them as web service methods. The REST API provides a resource-oriented view into these service methods.

The service methods are in the following categories:

- Methods that create, update, or delete resources by using the HTTP POST method and modify the system state by doing so.
- Methods that fetch resources or collections of resources by using the HTTP GET method and do not modify the system state.
- Methods that return system information, such as date or version information, by using the HTTP GET method and do not access resources.

Service methods that use HTTP POST to update resources:

Methods can create, update, or delete resources by using the HTTP POST method.

Methods that update resources modify the system state.

The X-HTTP-Method-Override (XMO) header qualifies the HTTP POST. The XMO header contains the name of the method that is exposed by the JSR 181 annotation.

The REST framework looks up the service that owns the resource, finds the method that is identified in the XMO header, deserializes the POST form parameters, calls the method, and returns the serialized representation of the modified resource.

Example: Annotating a method

For example, the ASSET service has the annotated moveSingleAsset business method:

```
@WebMethod
public void moveSingleAsset(@WSMboKey("ASSET") MboRemote asset, String newLocation,
     String newSite, String newParent) throws MXException, RemoteException
```

The method moves an asset from its current location to a new location and therefore modifies the state of the system. The following request is used to call the method:

```
POST maxrest/rest/mbo/asset/1234 HTTP/1.1
x-http-method-override: "moveSingleAsset"
```

The form data request is the following:

```
~asset=this&~newLocation=Some_where&~newSite=some_site&~newParent=Some_one
```

When the request is made, the following occurs:

1. The asset business object that has the identifier 1234 is loaded.
2. The owning service (ASSET) is looked up.
3. The method that is named moveSingleAsset is found.

4. The `asset` that is in the URI is referenced to the ASSET business object by the @WSMboKey annotation, and the business object is provided as the first parameter value in the form data request.
5. The remaining values are deserialized and assigned, based on their names.
6. The modified asset 1234 is returned to the client in the requested representation.

Example: Overriding a method parameter name

The form parameter names are the same as the method parameter names, prefixed with the ~ (tilde) character, unless they are annotated with the `@WebParam(name="...")` annotation.

For example, consider a case where the method definition has an operation name defined in the annotation:

```
@WebMethod(operationName="moveAssetLocation")
public void moveSingleAsset(...)
```

In this case, the following request is used to call the method:

```
POST maxrest/rest/mbo/asset/1234 HTTP/1.1
x-method-override: "moveAssetLocation"
```

The form data request is the following:

```
~asset=this&~newLocation=Some_where&~newSite=some_site&~newParent=Some_one
```

The XMO header value must use the attribute value specified in the @WebMethod annotation: `moveAssetLocation`.

Service methods that use HTTP GET to query resources:

A method name or annotated operation name that starts with the keyword **`get`** can be accessed by the HTTP GET method.

For example, the following method from the LOGGING service returns the list of MAXLOGGER business objects as a business object set:

```
@WebMethod
@WSMboSet(name="MAXLOGGER")
public MboSetRemote getLoggerList()
```

The following request is used to call the method and returns the collection of business objects in the configured representation:

```
GET maxrest/rest/mbo/maxlogger?_qop=getLoggerList HTTP/1.1
```

In the following variation of the method, an operation name is provided:

```
@WebMethod(operationName="getFilteredLoggers")
@WSMboSet(name="MAXLOGGER")
public MboSetRemote getLoggerList(String someFilter)
```

The following request is used to call the method:

```
GET maxrest/rest/mbo/maxlogger?_qop=getFilteredLoggers&~someFilter=... HTTP/1.1
```

The `_qop` value specifies the annotated name. The `someFilter` parameter is prefixed with the ~ character as part of the request.

Virtual methods

If a method name does not start with the keyword **get**, it cannot be called directly. For example, if a client tries to call the moveSingleAsset operation by using HTTP GET, an HTTP 405 error is thrown. However, you can define a virtual method by using a system property.

For example, suppose you create the property **mxe.rest.mbo.maxqueue.action.getQueueData** and assign it the value `_operation=viewQueue|~queueName=this.queuename`.

For the maxqueue resource, the virtual method getQueueData maps to the viewQueue operation in the owning service of maxqueue, which is the MIC service. The | (pipe) character separates the attributes. You can therefore access the viewQueue operation by using the following HTTP GET call:

```
GET /maxrest/rest/mbo/maxqueue/4567?_qop=getQueueData&~selector=...&~count=-1
```

You can specify literal or derived values for a method parameter.

For example, the viewQueue method has a parameter **queueName** that maps to the maxqueue.queuename attribute. The maxqueue is the business object in the current context that is specified by the GET URI. Therefore, the framework evaluates the derived value this.queuename as *current_mbo_in_context*.queuename, which is the value of the queuename attribute of maxqueue that is identified by the unique ID 4567.

As a second example, suppose you create the **mxe.rest.mbo.po.action.approve** system property and assign it the value `_operation=changeStatus|~status=APPR`.

A virtual operation called approve is created and it maps to the operation called changeStatus on PO, which is the owner service of the PO business object. For simplicity in the example, the value for the status parameter is hardcoded to APPR. The hardcoded value would not be practical because the status parameter value must be translatable.

If you use virtual methods, you must ensure that HTTP GET methods do not map to operations that modify the system state or have any other side effects.

Predefined initiateWorkflow operation

The predefined operation initiateWorkflow initiates a workflow for a business object resource. The following two examples use this operation:

```
POST /maxrest/rest/mbo/po/6789 HTTP/1.1
x-http-method-override: "initiateWorkflow"

wfname=SOMEWF
```

```
POST /maxrest/rest/mbo/po/6789?wfname=SOMEWF HTTP/1.1
x-http-method-override: "initiateWorkflow"
```

The method initiates the active revision of a workflow named SOMEWF for the purchase order business object that has the unique ID 6789. If the workflow has a wait node or a task node, it might require other steps or a background event to restart it, which is outside the scope of the REST API.

Service methods that use HTTP GET to query system data:

Some service operations retrieve system data and are not resource-oriented.

For example, the following HTTP GET retrieves the date and time:

```
GET /maxrest/rest/ss/system/getDate
```

The following XML is returned:

```
<getDateResponse xmlns=....>
<result>some ISO 8601 date</result>
</getDateResponse>
```

For another example, the following HTTP GET returns version information:

```
GET /maxrest/rest/ssm/system/getMXServerVersion
```

The following XML is returned:

```
<getMXServerVersionResponse xmlns=...>
<return>Tivoli's process automation engine 7.5.0.0 Build 20110127-1121
DB Build V7500-673</return>
<return>...</return>
</getMXServerVersionResponse>
```

To obtain a list of all services you can use the following GET method:

```
GET /maxrest/rest/ss
```

You can then drill down to each of the services in the list to get a list of operations that can be called. For example, the following method returns the operations for the service that is called `system`:

```
GET /maxrest/rest/ss/system
```

HTTP header properties

Several HTTP header properties are relevant to the REST API.

Table 34. REST API HTTP header properties

Property	Description
`Accept`	Provided by the requester and used for content negotiation to determine the response format of the request.
`Accept-Language`	Provides the data in a language that is requested. This property applies only to application fields that support multiple languages. The values that are provided (such as EN or FR) must be supported by the application. The query parameter **`_lang`** can supply the language code and provide the same capability as the **`Accept-Language`** header property. The query parameter **`_locale`** enables the numbers and dates that are returned in the locale of the requester.

Table 34. REST API HTTP header properties (continued)

Property	Description
Cache-Control	Notifies the requester whether caching is enabled. If caching is enabled, the **Cache-Control** property has a value of `private` to ensure that only the current user can reuse the content in the cache. When caching is not enabled, the **Cache-Control** property has a value of `no-cache`. The client request can contain the header **Cache-Control: no-cache** to disable caching for a particular request even though caching is enabled for the resource.
Content-Length	Contains the length of the response.
Content-Type	Notifies the requester of the format of the representation that is being sent. For example, the value `application/xml` (for XML response format) or `application/json` (for JSON response format) can be specified.
ETag	If caching is enabled, contains the Etag value for the resource that is requested. The value is held by the browser cache of the requester for subsequent requests for the same resource.
If-None-Match	If caching is enabled, contains the Etag value for the resource that was previously requested so that the API can determine if the cache contents can be reused.
Last-Modified	Notifies the requester of the date and time that the resource was last modified.
Location	Contains the link to a resource (HTTP code 201) that is created by an HTTP POST.
_rlid	Contains the ID of the resource collection and is used for in-session scrolling.

Related concepts:

"Caching of GET requests" on page 241
By enabling the caching of GET requests, you can improve the response times of requests for resource data that were previously submitted by the same user.

"In-session scrolling" on page 240
Holding a resource collection in memory can improve performance when scrolling through pages of data.

Related reference:

"GET method" on page 229
Use the GET method to retrieve business object resources and object structure resources.

"REST query parameters" on page 254
By using query parameters, you can tailor and filter the responses.

"POST method" on page 244
Use the POST method to update or insert a resource.

"REST system properties" on page 257
System properties are available to configure how the REST API works for your specific requirements.

Response codes

Messages from REST resources are propagated from the resource instance to the response.

The following HTTP response codes are implemented by the REST API.

Table 35. HTTP response codes

HTTP code	Cause
200	Success.
201	Success. The response contains a link.
304	Success. The data is retrieved from the cache.
400	The request cannot be received because of an invalid URI, for example.
401	An authorization violation occurred because of the configuration.
403	An authorization violation occurred because the resource is blocked by a **mxe.rest.***resourcetype***.blockaccess** system property.
404	The resource cannot be found or an invalid resource type was provided.
405	The HTTP method cannot be used for the resource. For example, you cannot use the DELETE method on a business object set.
406	The requested representation is not supported.
412	The resource is being updated by another user.
500	All other errors.

The messages support the languages that are supported by Tivoli's process automation engine.

Security in the REST API

Tivoli's process automation engine provides the authentication layer and authorization support for the REST API.

No specific authentication-related processing occurs within the REST API framework.

The authentication layer can provide either J2EE authentication or native authentication for the REST API, and the application server provides HTTPS support.

J2EE authentication

To use HTTP basic authentication, modify the `web.xml` file for the maxrest web module by uncommenting the following lines:

```
<!--
   <security-constraint>
       <web-resource-collection>
           <web-resource-name>REST Servlet for Web App</web-resource-name>
           <description>Object Structure Service Servlet (HTTP POST)
    accessible by authorized users</description>
           <url-pattern>/rest/*</url-pattern>
           <http-method>GET</http-method>
           <http-method>POST</http-method>
       </web-resource-collection>
       <auth-constraint>
           <description>Roles that have access to Object Structure
    Service Servlet (HTTP POST)</description>
           <role-name>maximouser</role-name>
```

```
        </auth-constraint>
        <user-data-constraint>
            <description>data transmission guarantee</description>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>REST Web Application Realm</realm-name>
    </login-config>
-->
```

The preceding <security-constraint> section refers to a single role, maximouser, which is defined in the <security-role> section of the file. By default, the <security-role> section is not commented out:

```
<security-role>
 <description>MAXIMO Application Users</description>
 <role-name>maximouser</role-name>
</security-role>
```

In addition, change the value for useAppServerSecurity from 0 to 1:

```
<description>Indicates whether to use Application Server
   security or not</description>
 <env-entry-name>useAppServerSecurity</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>0</env-entry-value>
</env-entry>
```

Native authentication

If native authentication is used, a request can provide the credentials by specifying a MAXAUTH header property that contains a Base64-encoded user and password.

If explicit login or logout requests are necessary, use the following requests:

```
GET /maxrest/rest/login
GET /maxrest/rest/logout
```

Cross-module communication

You can communicate with the REST API from the maximo and maxrest web modules.

In cross-module communication, you typically do not want the user to re-enter their credentials. If you use application server authentication, this issue does not arise because the browser propagates the authentication token to the web modules for each request. For native authentication, the REST API is available as a library (commonweb) that any web module can access.

For example, the following URI queries an asset by using the maxrest web module: http://site.example.com:9999/maxrest/rest/mbo/asset/46.

In contrast, the following URI queries an asset by using the user interface web module: http://site.example.com:9999/maximo/rest/mbo/asset/46.

Authorization

For object structures, you can connect a resource to an authorizing application on the Object Structure tab of the Object Structures application. If you specify an authorizing application, all integration processing that uses the object structure is affected.

For business objects, you can connect a resource to an authorizing application in the Application Authorization for Objects window of the Object Structures application. The application authorizes all REST API requests for the specific MBO resource.

You can restrict resource access by specifying a list of resource names in the **mxe.rest.mbo.blockaccess** and **mxe.rest.os.blockaccess** system properties.

Related reference:

"REST system properties" on page 257
System properties are available to configure how the REST API works for your specific requirements.

Customization of the REST API

The REST API can be customized to support different resource representations, such as the XHTML or Atom formats. It can also be customized by adding attributes to the representation of a business object resource or by changing the structure of a representation.

To support a new resource representation, you must provide custom serializers that support the new formats.

If you add attributes, such as an attribute to the work order MBO resource in JSON representation, you must add a custom JSON serializer for the resource. To support additional query parameters, you must extend the MBO or OS handler.

Ensure that your customizations do not affect existing business processes or users. By creating a separate instance of the REST API for the resource types, you can avoid adverse affects.

For any customization, the class files must be included in the application EAR file as part of the commonweb library.

Example

The following steps customize the REST API:

1. Create a system property by copying the **mxe.rest.handler.mbo** property, giving the property the name **mxe.rest.handler.abcmbo**, and assigning it the value `com.ibm.tivoli.maximo.rest.ABCMboResourceRequestHandler`.

   If the customization is generic to all MBOs, the custom class extends the base MBO handler. The URI context is `abcmbo`, and therefore URLs are in the following form: `/maxrest/rest/abcmbo/workorder?...`

   This step creates an instance of the REST API for the MBO resource.
2. To handle query parameters that are specific to the work order MBO, provide a custom handler by creating the **mxe.rest.handler.abcmbo.workorder** system property and assigning it the value `com.ibm.tivoli.maximo.rest.WOMboResourceRequestHandler`.

The class could extend the default MBO handler by using ABCMboResourceRequestHandler for MboResourceRequestHandler, for example.

3. To customize the JSON serializer for the work order MBO, register the serializer by creating the **mxe.rest.serializer.abcmbo.workorder.json** system property and assigning it the value `com.ibm.tivoli.maximo.rest.WOMboJSONSerializer`.
4. To support a new format, such as Atom, add the value `atom` to the **mxe.rest.supported.formats** system property. The name `atom` uniquely identifies the serializer for the format. You can specify any name as long as a serializer that has the name does not exist.
5. Add a serializer class by creating the **mxe.rest.serializer.abcmbo.atom** system property and assigning it the value `com.ibm.tivoli.maximo.rest.MboATOMSerializer`.
6. To identify the new format by using the HTTP **Accept** header, provide the mime type mapping by creating the **mxe.rest.format.atom.mimetypes** system property and assigning it the value `application/atom+xml`.
7. Set the default format by assigning the value `atom` to the **mxe.rest.abcmbo.defaultformat** system property.

By following these steps, the customization does not affect other MBO or OS resource handlers because it is within the context of the `abcmbo` handler.

REST query parameters

By using query parameters, you can tailor and filter the responses.

You can specify one or more of the following query parameters to control the data that is selected. Boolean values must be set to `1` for true or `0` for false.

Table 36. REST query parameters

Name	Type or value	Description	Default
_compact	Boolean	If true, JSON data is represented in a compact representation.	False
_dropnulls	Boolean	If true, attributes are serialized only if the business object or object structure resource has a value. If false, all attributes are serialized, even if they are null.	True
_exactmatch	Boolean	If true, parameter values are evaluated for an exact match. If false, a LIKE evaluation occurs.	False
_excludecols	Comma-separated list of MBO attributes	Excludes the listed attributes from the resource response. Valid only for the business object resource type.	
_fd		Specifies an application table domain to apply as a filter when the query is run. Can be used with the **_fdorg** and **_fdsite** parameters.	
_fdorg		Specifies an organization ID to apply as a filter when the query is run. Must be used with the **_fd** parameter.	
_fdsite		Specifies a site ID to apply as a filter when the query is run. Must be used with the **_fd** parameter.	
_format	json or xml	Specifies the representation of the resource response.	xml

Table 36. REST query parameters (continued)

Name	Type or value	Description	Default
_generic	Boolean	If true, the resource response field data is returned in a generic format. Applies to JSON data only.	False
_glc	Boolean	If true, the general ledger (GL) component data is included for all Chart of Accounts application fields of the business object. For most Chart of Accounts fields, the individual component data is included. This parameter by default omits the component data and therefore might improve performance.	False
_includecols	Comma-separated list of MBO attributes	Specifies the attributes that are included as part of the resource response. Valid only for the business object resource type.	
_keys	Boolean	If true, only the key attributes of the business object are serialized. All other attributes are dropped from the resource response.	False
_lang		Specifies the language code of the requester. If this parameter is not specified, the **accept-language** parameter of the HTTP header is used to determine the language preference of the requester. If the HTTP header does not specify the parameter, the value from the default integration user is used.	
_lid and **_lpwd**		Specifies a login user ID and password. For development and testing use only. Valid only when native authentication is configured.	
_locale	Boolean	If true, the locale-specific text version of the business object attribute value is returned in addition to the strong typed value.	False
_maxItems	Numeric	Specifies the maximum number of business objects that are serialized in the resource collection. Use this parameter with the **_rsStart** parameter to page the response for large resource collections. The parameter name is case-sensitive.	
_md	Boolean	If true, metadata information, such as hidden, read-only, and required values, are returned.	False
_opmodeor	Boolean	If true, the OR operator is applied between multiple attribute parameters. If false, the AND operator is applied.	False
_orderby		Specifies how the fields of a resource are ordered. The value can be `asc` (ascending) or `desc` (descending). For an object structure resource, this parameter can be used only for the fields of business objects that are at the first or second level of the object structure.	
_orderbyasc	Comma-separated list of MBO attributes	Specifies the attributes that are used for the order by clause in ascending order.	
_orderbydesc	Comma-separated list of MBO attributes	Specifies the attributes that are used for the order by clause in descending order.	

Table 36. REST query parameters (continued)

Name	Type or value	Description	Default
_page		Specifies the page number that the request must return. This parameter can be used only if a **pagesize** system property is created for the resource that is being queried.	
_qop		Specifies a query method for the GET operation. Valid for MBO resources only.	
_rcol.*alias*		Specifies a relationship name field that is returned as part of the query response. The *alias* is the alias name for the related attribute. By specifying this parameter, you can query fields that are related to the MBO resource that is queried. For example, in the parameter **_rcol.polinecost=poline.linecost**, **polinecost** is the alias name and **poline.linecost** is the related attribute.	
_retainmbos	Boolean	For internal use only.	
_rlid	Boolean	Specifies the ID of a resource collection. This parameter is used only for in-session scrolling.	
_rlrq	Boolean	If true, the cache is released for the resource collection that is specified by the **_rlid** parameter.	False
_rootonly	Boolean	If true, only the main object of the object structure is serialized. Valid only for the object structure resource type.	False
_rowstamp		For update requests, the rowstamp business object can be provided to ensure that the object that is being updated has not changed since it was first queried.	
_rsStart	Numeric	Specifies the start index of the MBO in the resource collection that is serialized by the resource response. The parameter name is case-sensitive.	0
_tc	Boolean	If true, the total number of records is calculated and that number is assigned to the rsTotal attribute. The calculation adds to the processing time, depending on the size of the resource collection.	False
_tz		Specifies the time zone of the request. Use this parameter to correctly run date-related queries when the time zone of the request is different from the time zone of the application.	
_ulcr	Boolean	If true, a POST method returns a link to the new resource in the HTTP header **Location** property.	False
_urs	Boolean	If true, a query response includes the rowstamp value for the resource.	
_usc	Boolean	If true, in-session scrolling is used and the HTTP response includes the resource collection ID by using the **_rlid** property.	False

Table 36. REST query parameters (continued)

Name	Type or value	Description	Default
_uw		Specifies an SQL WHERE clause to be used as the selection criteria. The clause must be compatible with a clause that works in advanced searches in applications. Use with the **mxe.rest.whereclause.usepolicy** system property.	
_verbose	Boolean	If true, the resource response for JSON representation is returned as formatted.	False
_vt	Boolean	If true, the XML is validated for valid XML characters. If you set the parameter to false, no validation is run and therefore performance might improve. If this parameter is not specified, the **mxe.int.validatexml** system property determines whether XML is validated.	True

Related concepts:

"In-session scrolling" on page 240
Holding a resource collection in memory can improve performance when scrolling through pages of data.

"_format and _compact parameters" on page 238
The REST API supports XML and JSON representations (formats). You can use the **_format** parameter or content negotiation to specify a resource representation. You can use the **_compact** parameter to specify compact JSON format.

"_dropnulls parameter" on page 237
Set the **_dropnulls** parameter to false to include fields that have a null value in the query response.

"_includecols and _excludecols parameters" on page 237
Use the **_includecols** and **_excludecols** parameters to control the content of attributes that are returned in response to a query.

"_rsStart and _maxItems parameters" on page 235
The **_rsStart** and **_maxItems** parameters are used together to control the number of records returned for a request and to allow paging through a large volume of records.

"_opmodeor parameter" on page 234
Use the **_opmodeor** parameter to evaluate multiple fields by using an OR condition between values instead of the default AND condition.

"_orderbyasc parameter" on page 236
The **_orderbyasc** parameter controls the sort order of returned data.

Related reference:

"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

"REST system properties"
System properties are available to configure how the REST API works for your specific requirements.

REST system properties

System properties are available to configure how the REST API works for your specific requirements.

Predefined system properties

Use the following system properties to configure how the REST API works. The properties all start with **mxe.rest**, such as **mxe.rest.format.json.mimetypes**.

Table 37. Predefined REST system properties

mxe.rest.*property* **name**	**Description**	**Default value**
format.json.mimetypes	A comma-separated list of supported MIME types for JSON that is included in the HTTP **Accept** header property. The value is used only for content negotiation and is superseded by query parameters that specify a format.	application/json
format.xml.mimetypes	A comma-separated list of supported MIME types for XML that is included in the HTTP **Accept** header property. The value is used only for content negotiation and is superseded by query parameters that specify a format.	application/xml,text/xml
handler.mbo	The resource handler class file for business objects.	com.ibm.tivoli.maximo.rest.MboResourceRequestHandler
handler.os	The resource handler class file for object structures.	com.ibm.tivoli.maximo.rest.OSResourceRequestHandler
handler.ss	The standard service handler class file that supports access to system data.	com.ibm.tivoli.maximo.rest.MaxServiceResourceRequestHandler
mbo.blockaccess	A comma-separated list of business objects that the REST API cannot access. This property overrides any configured authorization restrictions.	
mbo.defaultformat	The default representation for all business objects. This property is superseded by the value of the **Accept** header that is used in content negotiation and by query parameters that specify a representation.	xml
mbo.imglib.defaultformat	The REST default representation for the IMGLIB business object.	image
os.blockaccess	A comma-separated list of object structures that the REST API cannot access. This property overrides any configured authorization restrictions.	
os.defaultformat	The default representation for all object structures. This property is superseded by the value of the **Accept** header property that is used in content negotiation and by query parameters that specify a representation.	xml

Table 37. Predefined REST system properties (continued)

mxe.rest.*property* **name**	**Description**	**Default value**
serializer.mbo.imglib.image	The serializer class for the IMGLIB business object that is in image format.	com.ibm.tivoli.maximo.rest.ImageLibSerializer
serializer.mbo.json	The serializer class for business objects that are in JSON format.	com.ibm.tivoli.maximo.rest.MboJSONSerializer
serializer.mbo.xml	The serializer class for business objects that are in XML format.	com.ibm.tivoli.maximo.rest.MboXMLSerializer
serializer.os.json	The serializer class for object structures that are in JSON format.	com.ibm.tivoli.maximo.rest.OSJSONSerializer
serializer.os.xml	The serializer class for object structures that are in XML format.	com.ibm.tivoli.maximo.rest.OSXMLSerializer
serializer.ss.json	The serializer class for standard service system data that is in JSON format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseJSONSerializer
serializer.ss.xml	The serializer class for standard service system data that is in XML format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseXMLSerializer
ss.defaultformat	The default representation for all standard service system data. This property is superseded by the value of the **Accept** header property that is used in content negotiation and by query parameters that specify a representation.	xml
supportedformats	The list of supported representations.	xml,json,image
webappurl	The URL for the token authentication web application. This property is for internal use only.	
whereclause.usepolicy	Specifies how the **_uw** query parameter is used. If the value is `parse`, the SQL WHERE clause is inspected to prevent cross-site scripting and SQL injection attacks. If the value is `noparse`, the clause is not inspected. If the value is anything else, the WHERE clause is ignored.	parse

Optional system properties for caching

You can create the following properties to configure caching in the REST API. The properties all must start with **mxe.rest**, such as **mxe.rest.mbo.wo.cache**

Table 38. Optional REST system properties for caching

mxe.rest.*property* **name**	**Type**	**Description**	**Examples**
handler.*resource*.**cache**	Boolean	Enables pessimistic caching for the specified handler and resource. Defaults to false.	**mxe.rest.mbo.wo.cache** set to 1 **mxe.rest.os.mxwo.cache** set to 1
handler.*resource*.**deepetag**	Boolean	If optimistic caching is enabled, specifies that all objects in the object structure are evaluated instead of just the root object. Defaults to false.	
handler.*resource*.**optimistic**	Boolean	Enables optimistic caching for the specified handler and resource. Applies to collections of one or more resources. For an object structure, only the main business object is evaluated for a change in state. Child business objects are not evaluated. Non-persistent attributes of a business object are excluded from the evaluations of state changes. If you enable optimistic caching, you can also specify the *handler*.*resource*.**deepetag** system property. Defaults to false.	**mxe.rest.mbo.wo.optimistic** set to 1 **mxe.rest.os.mxwo.optimistic** set to 1
handler.*resource*.**maxage**	Integer, in seconds	Specified the maximum amount of time, in seconds, that a resource collection is maintained in the cache.	**mxe.rest.mbo.wo.maxage** set to 60 **mxe.rest.os.mxwo.maxage** set to 60

Related concepts:

"Caching of GET requests" on page 241
By enabling the caching of GET requests, you can improve the response times of requests for resource data that were previously submitted by the same user.

"Supported representations" on page 228
The REST API supports XML and JSON as representations. Representations are supported by the implementation of serializer classes that are registered in system properties.

Related reference:

"REST query parameters" on page 254
By using query parameters, you can tailor and filter the responses.

"HTTP header properties" on page 249
Several HTTP header properties are relevant to the REST API.

External service calls

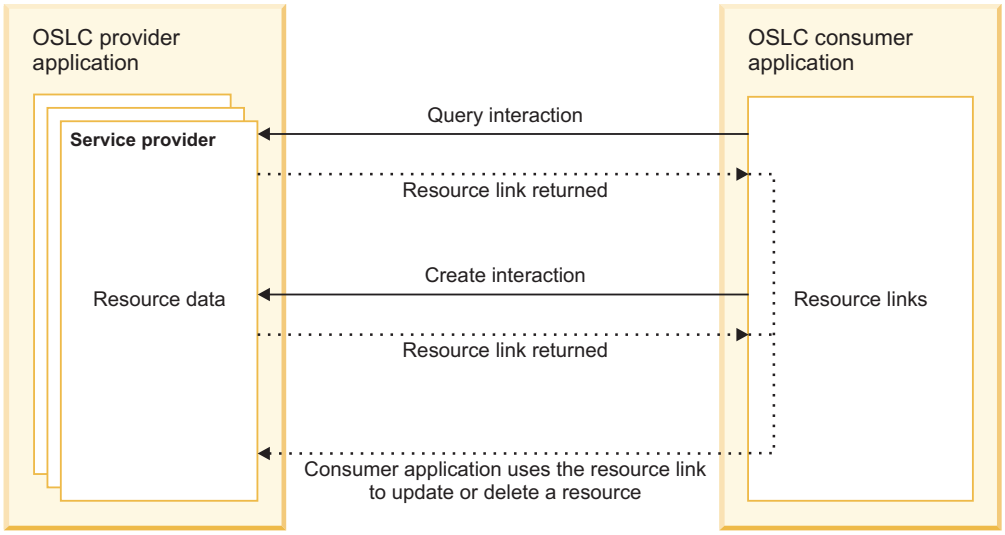
External REST APIs can be called by using a configured HTTP end point.

The HTTP handler that supports any configured HTTP end point enables the use of REST-based services that use an XML payload and that use POST and GET methods. The end point can be configured to send messages through a publish channel or an invocation channel.

OSLC integration

Maximo Integration Framework supports the sharing of lifecycle data between applications based on Open Services for Lifecycle Collaboration (OSLC) integration. With OSLC integration, a consumer application can perform create, request, update, and delete operations on the data resources that a provider application makes available for integration through an OSLC service provider.

An OSLC provider application makes containers of associated data resources available for integration through service providers. Consumer applications then use these service providers to query resources and to create, update, and delete resource data. The following figure shows the typical interactions in an OSLC integration. The consumer application sends a query to the service provider for resource data. The service provider provides a link to the resource data in response. The consumer application uses the link to create, update, or delete resource data.



Related information:

- Open Services for Lifecycle Collaboration community
- OSLC on OASIS Open Standards Network
- OSLC platform community (IBM)

OSLC implementation in Maximo Asset Management

OSLC specifications for representing and exchanging linked data between applications are maintained and developed by the OASIS consortium. Maximo Asset Management supports version 2.0 of the OSLC core specification that you can enable and extend with Maximo Integration Framework.

OSLC specifications define how lifecycle applications represent, link to, and access resources based on established internet and linked-data standards including representational state transfer (REST) architecture, resource description framework (RDF) specifications, and hyper-text transfer protocol (HTTP) methods. Specifications are designed to be minimal in nature to address common integration

use cases and integrators can extend these specifications or add new ones to address specific integration scenarios. The Maximo Asset Management OSLC core specification focuses on general requirements and a number of additional specifications represent the resources for particular functional domains, such as change management, or requirements management.

Maximo Asset Management is configured as an OSLC provider with the following components:

- The provider application makes data resources available for integration based on integration object structures that include the primary business object and any associated child business objects.
- OSLC resource shape documents describe resource objects in RDF/XML format. A shape document indicates what are the required characteristics of a resource and can also describe many aspects of the resource object, including its dependencies, attributes, and properties. A resource shape document can include links to the shape documents for any child objects of the resource.
- Resources are grouped by functional domain. Functional groupings can be based on the existing domain specifications that are developed and maintained by the OASIS OSLC group. Integrators can extend these existing domain specifications or create additional domain specifications to support business requirements.
- The resources in a domain are made available to consuming applications through service providers.
- A single service provider is supported for each domain. Service providers support the OSLC creation factory and query capability operations that provide consuming applications with the URI to create or search supported resources.
- The security framework supports authentication and authorization for OSLC services. Native and J2EE authentication are supported. Authorization control is provided by the object structure for a resource and application-level and user-based authorization are applied.

An OSLC consumer uses the following HTTP methods and header information to interact with the OSLC provider:

- Sends a GET request to perform login if explicit login is required and includes an authorization property in the HTTP header if native authentication is used. If explicit logout is required, a GET request is also used.
- Sends GET requests that include OSLC query parameters to a service provider URI to query OSLC resources.
- Sends a POST request to create an instance of an OSLC resource. The request includes a JSON document that conforms to the published shape document for the OSLC resource. When the resource instance is created, the OSLC provider sends a HTTP response that includes the URI for the new resource.
- Sends a PUT or PATCH request to update an OSLC resource. A PUT request updates the entire resource. A PATCH request updates part of the resource.
- Uses entity tags (ETags) in HTTP headers to ensure that a resource entity is current and can combine ETags with If-Match headers in PUT and PATCH requests to enable conditional updates.
- Can request stable paging in GET requests for collection resources so that the response redirects the consumer application to a URI where multiple pages are loaded. For collection resources, performance improves because the consumer application can load stable pages without accessing the database for each page.

HTTP transactions

OSLC I nteractions between consumer applications and the provider application use standard HTTP requests and responses. Supported HTTP request methods include GET, POST, PUT, and PATCH. HTTP responses provide information in HTTP headers and HTTP error codes are returned when requests are not successful.

OSLC resource queries:

OSLC defines a lightweight query syntax that is based on SPARQL standard to query resources. The OSLC query parameter that is set in a HTTP GET request determines the type of information that the service provider sends in response.

Query properties parameter:

The **oslc.properties** query parameter returns a list of properties for an OSLC resource and provides a partial representation of the resource.

The **oslc.properties** parameter is not applicable to collection resources. A collection resource is an OSLC resource that has other OSLC resources as members.

Example: Requesting attributes

The following request specifies that the values for the shortTitle and isTask attributes are returned in the results:

```
http://www.example.com:9999/maximo/oslc/os/
oslcwodetail/337?oslc.properties=oslc:shortTitle,spi_wm:istask
```

This request returns the following information:

```
{
  oslc:shortTitle: "1024"
  spi_wm:istask: true
  rdf:about: "<some uri>"
}
```

Query WHERE clause parameter:

The **oslc.where** query parameter specifies a WHERE clause for filtering the result set of a query. For example, you might want to see a collection of OSLC work order resources that were created within a time range where the work orders are approved by management.

The OSLC WHERE clause supports the following comparison operators:

Symbol	Description
=	Equality
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

OSLC WHERE clause parameters have the following characteristics:

- Dates are expressed in ISO 8601 format.

- The OSLC specification supports *and* as the boolean operator between boolean expressions. The boolean operator *or* is not supported.
- String data type values are surrounded by quotation marks but decimal data type values do not have quotation marks.
- Decimal data type values are not surrounded by quotation marks. In the following example, the literal value for status is in quotation marks because the status property has a data type of string. The quantity value does not have quotation marks because it is a decimal data type. Integers and boolean values also do not require quotation marks. For example, `spi_wm:status="Closed" and m:quantity>10.5 and m:active=true` where m:active has a boolean data type.
- The WHERE clause supports *or* within a single property by using the *in* operator. For example, the following query returns all work orders that are in either APPR or WAPPR status:

  ```
  spi_wm:status in ["APPR","WAPPR"]
  ```

Example: Searching for work orders that were created within a time range and are approved

The following WHERE clause query returns a list of work order resources that were created within a specific time range:

```
spi_wm:status="APPR" and dcterms:created>"2003-07-07T09:50:00-04:00"
and dcterms:created<="2004-07-07T09:50:00-04:00"
```

Example: Searching for all work orders created by a user

The following WHERE clause query returns all work orders that were created by a specific user or person:

```
dcterms:creator= <http://host:port/oslc/os/oslcperson/
<URI id for person Kelly Reese>
```

The URI value is delimited by angle brackets and is not surrounded by quotation marks, unlike string literal values. The URI points to the person resource named Kelly Reese. Although the use of a URI in the query syntax works for peer system integrations, interactive or UI-based consumers of the OSLC API would not know any of the relevant URIs. For example, an interactive user would know an asset number but would not know the URI that corresponds to that asset. The following WHERE clause query is expressed without a URI:

```
dcterms:creator{foaf:givenName="Kelly" and foaf:familyName="Reese"}
```

givenName and familyName are properties of the Person resource that is referred from the creator property. This WHERE clause demonstrates that you can search based on linked resource properties.

Example: Searching for all work orders where the parent property of the work order resource is null

The following WHERE clause query returns all root-level work orders:

```
spi_wm:parent!="*"
```

In this example, `*` is a wildcard that refers to any resource and the `parent!="*"` query is the semantic equivalent of a `parent="NULL"` query.

A derivative of this query checks for work orders where the parent value is not NULL. The syntax of this query is `spi_wm:parent="*"`. You can also perform a

LIKE search with the OSLC WHERE clause query syntax. To search for all of the work orders that have a shortTitle value such as *Inspect*, you use the following query: `oslc:shortTitle ="%Inspect%"`.

To search for work orders that have a shortTitle value that begins with or ends with a word that you specify, you move the `%`. For example, use `oslc:shortTitle ="Inspect%"` to search for work orders that start with the word *Inspect* and use `oslc:shortTitle ="%Inspect"` to search for work orders that end with the word *Inspect*.

Query search parameters:

The **`oslc.searchTerms`** query parameter returns resources that contain specified terms. For example, you might want to see all of the work orders that have the words *database* and *performance* in their descriptions.

To search for terms, the field that you search against must be configured for search in the OSLC Resources application.

Example: Searching for work orders that refer to database performance

The following request queries work order resources by selecting the short title, status, and description where the description contains the terms database and performance:

```
http://host:7001/maximo/oslc/os/oslcwodetail?oslc.select=oslc:shortTitle,
spi_wm:status,dcterms:description&oslc.searchTerms=database,performance
```

The query returns the following information:

```
 -
"rdfs:member": [
-
{
 "oslc:shortTitle": "1001",
 "rdf:about": "http://host/maximo/oslc/os/oslcwodetail/76",
 "spi_wm:status": "INPRG",
 "dcterms:description": "performance is key<!--RICH TEXT -->"
},
-
{
 "oslc:shortTitle": "2004",
 "rdf:about": "http://host/maximo/oslc/os/oslcwodetail/58",
 "spi_wm:status": "INPRG",
 "dcterms:description": "all the database and stuff<!--RICH TEXT -->"
},
-
{
 "oslc:shortTitle": "6003",
 "rdf:about": "http://host/maximo/oslc/os/oslcwodetail/8",
 "spi_wm:status": "APPR"
},
-
{
 "oslc:shortTitle": "1004",
 "rdf:about": "http://host/maximo/oslc/os/oslcwodetail/155",
 "spi_wm:status": "INPRG",
 "dcterms:description": "Performance is key"
},
-
{
 "oslc:shortTitle": "1006",
 "rdf:about": "http://host/maximo/oslc/os/oslcwodetail/73",
 "spi_wm:status": "APPR"
}
```

```
],
"rdf:about": "http://host/maximo/oslc/os/oslcwodetail"
-
"prefixes":{
 "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
 "oslc": "http://open-services.net/ns/core#",
 "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
 "spi_wm": "http://jazz.net/ns/ism/work/smarter_physical_infrastructure#",
 "dcterms": "http://purl.org/dc/terms/"
 }
}
```

Query sort parameter:

The **oslc.orderBy** query parameter defines how the results of a query are ordered. For example, a list of work orders can be ordered by date or by ID.

Example: Specify the sort order

The following **oslc.orderBy** query returns work orders based on the creation date in ascending order and the estimated duration in descending order:

```
+dcterms:created,-m:estimatedDuration
```

In this query,+ indicates an ascending sort order and - indicates a descending sort order. The following **oslc.orderBy** parameter is not valid: `dcterms:created,-m:estimatedDuration` because there is no default sort order in OSLC query syntax. There must be an explicit + or - with the property name. The **oslc.orderBy** parameter is not supported for nested properties. For example, `dcterms:creator{+foaf:name}` is not supported.

Query select parameter:

The **oslc.select** query parameter requests a partial resource representation of the resources in a collection resource. The **oslc.select** parameter always applies to a collection resource. You specify the list of properties to include in the request.

Example: Partial resource request

The **oslc.select** parameter provides a comma-separated list of qualified property names. The **oslc.prefix** parameter is not supported. The following query requests a partial resource:

```
oslc.select=oslc:shortTitle, dcterms:creator
```

The query returns the following information:

```
{
 "rdf:about": "some uri",
 "rdfs:member": [
 {
  "oslc:shortTitle": "1022",
  "rdf:about": "some workorder uri",
  "dcterms:creator":
  {
      "rdf:about": "some person uri"
  }
```

Example: Selecting properties from referenced resources

With the **oslc.select** parameter, you can select properties from referenced resources. The following query requests the name of the creator:

```
oslc.select= oslc:shortTitle, dcterms:creator{foaf:name}
```

The query returns the following information:

```
{
 "rdf:about": "some uri",
 "rdfs:member": [
 {
  "oslc:shortTitle": "1022",
  "rdf:about": "some workorder uri",
  "dcterms:creator":
  {
      "rdf:about": "some person uri",
      "foaf:name": "Todd Winston"
  }
```

In this example, the foaf:Person resource is the name of the person that is specified in the creator property value. To get all the properties from the resource, you can use `oslc.select=*`. The same syntax can be applied to the **`oslc.properties`** parameter when you search for an OSLC resource.

Creation of a resource instance:

A consuming application uses the HTTP POST method to create an instance of an OSLC resource. Other applications can then share the resource.

The consuming application uses the HTTP POST method to send a JSON document that conforms to the published shape of the resource. The OSLC JSON format is associated with the `application/json` MIME type. When the request is processed successfully, the header of the HTTP response from the provider application includes the URI of the newly-created resource.

Example: Creating an instance of a resource

The following method creates the `oslcwodetail` resource:

```
POST /maximo/oslc/os/oslcwodetail
....
{
   "dcterms:creator": {
      "rdf:resource": "http://host:port/maximo/oslc/os/oslcperson/_V0lOU1RPTg--"
   },
   "dcterms:title": "Check-out Leaking",
   "spi_wm:orgid": "EAGLENA",
   "spi_wm:siteid": "BEDFORD",
   "spi_wm:woclass": "WORKORDER",
   "oslc:shortTitle": "T5050",
   "spi:asset": {
      "rdf:resource": "http://host:port/maximo/oslc/os/oslcasset/_MTMxNDUvQkVERk9SRA--"
   },
   "spi:location": {
      "rdf:resource": "http://host:port/maximo/oslc/os/oslcoperloc/_U0hJUFBJTkcvQkVERk9SRA--"
   },
   "spi:status": "WAPPR"
}.....
```

If the request is processed successfully, the consumer application receives the following response:

```
201 Created
Location: http://host:port/maximo/oslc/os/oslcwodetail/_ABCD--
ETag: 1234567
```

The message body of the HTTP response is empty. To get details of the newly-created resource, the consumer application must send an HTTP GET request to the URI that is specified in the HTTP header location property. The consuming

application can use the ETag property in the HTTP header to send a conditional update request to the provider application.

OSLC requests can fail for various reasons, such as business validation, authentication, or authorization. If a request fails, the consuming application receives an HTTP error code as a response, such as a `400 Bad Request` error and the message body contains the details of the error.

Modification of resources:

The HTTP PUT method replaces an OSLC resource and the HTTP PATCH method partially updates an OSLC resource.

Replacement of an OSLC resource:

The HTTP PUT method replaces all of the properties of an OSLC resource, including literal properties and local resource properties. The PUT method also deletes any local resource properties that are not included in the request.

When you use the PUT method to replace an OSLC resource, the following rules apply:

- All literal properties that are specified in the HTTP request document are updated. Any literal property that is not specified as part of the request is not explicitly affected by the request. However, literal properties can be implicitly affected by the business logic that is associated with the resource.
- All local resource properties are replaced by their corresponding values in the request. When a resource property is included in a PUT request, the value replaces the value on the server. If a resource property is not included in a HTTP request, the corresponding property is deleted on the server.
- You cannot explicitly update reference resources but you can update properties that refer to the resource, and the properties follow the update model of literal properties.

In the following examples, a work order resource has one literal property, **`title`**, and one resource property **`wplabor`**. The **`wplabor`** property points to the local resource WPLABOR and is associated with two wplabor records. If a PUT request contains the **`title`** property and no **`wplabor`** property, the title is updated and the wplabor data is deleted.

Example: Updating a literal property

The following method updates the literal property, **`title`** and deletes the **`wplabor`** property and the associated data:

```
PUT /maximo/oslc/os/oslcwodetail/abc

{
   "dcterms:title": "Check-out Leaking – Modified for Test"
}
```

If the request is processed successfully, the consuming application receives the following HTTP response:

```
204 No Content
ETag: 123456
```

The title is changed to `Check-out Leaking – Modified for Test`. Because the **`wplabor`** data is not included, the wplabor records are deleted.

Example: Updating a resource property

The following method updates the resource property, **wplabor**:

```
PUT /maximo/oslc/os/oslcwodetail/abc

{
   "spi:wplabor": [
 {
   "spi_wm:wplaborid": "0000000067",
 "spi_wm:rate": 18.5
            }
]
}
```

If the request is processed successfully, the consuming application receives the following HTTP response:

```
204 No Content
ETag: 123456
```

The request initiates a search for a wplabor record with the ID 0000000067. If this wplabor record exists, it is updated. If no matching record is found, a new wplabor record is created with the ID 0000000067. All other wplabor data for this work order resource is deleted. Because the **title** property is not included, the title is not part of the request and the value is unaffected.

Partial update of an OSLC resource:

The HTTP PATCH method updates part of an OSLC resource. Unlike the PUT method, the PATCH method does not delete local resource properties that are not included in the request. The x-method-override header is required to implement the PATCH method.

When you use the PATCH method to replace an OSLC resource, the following rules apply:

- All literal properties that are specified in the request document are updated. Any literal property that is not specified as part of the request is not explicitly affected by the request. However, literal properties can be implicitly affected by the business logic that is associated with the resource.
- Local resource properties are updated or replaced if there are corresponding property values in the PATCH request. If a resource property is not included in the request, the corresponding local resource is not explicitly affected by the request. If a resource property is included in the request, the incoming value replaces or updates the value in the server.
- You cannot explicitly update reference resources but you can update properties that refer to the resource, and the properties follow the update model of literal properties.

HTTP PATCH requests are used in the following scenarios:

- The PATCH request replaces an array (local resource) property with the content in the request. This scenario is the default implementation.
- You can find and match array resource elements from the request with corresponding resource elements on the server. Depending on whether a match is found, the array resource elements are updated or inserted. An array element is never deleted from the local resource property. To use a PATCH request to match array elements, the consumer application sets the PATCHTYPE HTTP request header to the value MERGE (case sensitive).

Example: Updating a literal property

The following method updates the title property of a work order:

```
POST /maximo/oslc/os/oslcwodetail/abcd
x-method-override: PATCH

{
   "dcterms:title": "Check-out Leaking – Modified for Test"
}
```

Unlike the PUT method, this PATCH method does not update other properties of the work order.

Example: Updating and merging a resource property

The following method updates the resource with the PATCHTYPE header set to MERGE:

```
POST /maximo/oslc/os/oslcwodetail/abcd
x-method-override: PATCH
PATCHTYPE: MERGE

{
   "dcterms:title": "Check-out Leaking – Modified for Test",
    "spi:wplabor": [
 {
   "spi_wm:wplaborid": "0000000067",
 "spi_wm:rate": 18.5
           }
]
}
```

The request initiates a search for a wplabor record with the ID `0000000067`. If such a wplabor record exists, the record is updated. If no match is found, a new wplabor record is created with the ID `0000000067`. Because the PATCHTYPE header is set to `MERGE`, the other wplabor records for this work order resource remain unchanged.

Example: Making a conditional update

The following method updates the resource if the ETag value is 1234567:

```
POST /maximo/oslc/os/oslcwodetail/abcd
x-method-override: PATCH
if-match: 1234567
```

If the ETag value is `1234567`, the work order resource is updated and an `HTTP 204` response is sent to the consumer application.

If the ETag value is not `1234567`, the server responds with an `HTTP 412 Precondition failed` message. This message indicates that the resource was updated by some other process and the requesting consumer application has an out-of-date copy of the resource. The consumer application must perform a GET request on the **abcd** resource to get a fresh copy of the resource.

HTTP headers:

OSLC provides various values in the header section of HTTP requests and responses to exchange additional information in transactions. These header values

support features such as stable paging, conditional updates of resources, or to ensure that duplicate transactions do not occur following a HTTP connection failure.

Conditional updates based on Etag values and If-Match headers

An entity tag (ETag) is a value that is included in a HTTP header response that represents the current state of a resource. When an OSLC consumer application makes a GET request, the response header includes an ETag value. The consumer application includes the ETag value as part of an If-Match header in subsequent PUT and PATCH update requests to ensure that changes are conditional and are made only if the record has not changed since the ETag value was created. The conditional update process detects bad updates and race conditions, for example when two consumers try to update the same resource.

The consumer application stores the ETag header value and sends it as part of HTTP If-Match header for a subsequent update request. The server evaluates the If-Match header and determines whether the consumer application has an old version or the most recent version of the resource. If the server determines that the consumer application has the most recent version of the resource, the update is implemented unless any business validation or database constraints are found. If the server determines that the request contains an old version of the resource, it returns an HTTP 412 precondition failed response. The consumer application must get the resource again and submit an update request that is based on the updated ETag.

The consumer application can submit an update request without an If-Match header or with the If-Match header value set to * (asterisk). Submitting this request is semantically equivalent to having no If-Match header in the update request. In both cases, the update is unconditional. If the resource that is referred by the URI exists and no business validation or database constraints are found, the update is implemented.

Stable paging for collection resources

OSLC stable paging defines a paging pattern where the resource that is paged does not change when it is being paged to the consumer application, for example to a mobile device. Stable paging is supported only for collection resources and is not the default paging format of HTTP servers. To request stable paging, the OSLC consumer application includes the **stablepaging** query parameter in the HTTP request or includes a stablepaging header in the request. The server loads the requested collection resource in memory and the HTTP response redirects the OSLC consumer application to the URI where the loaded collection resource is stored. Subsequent requests for the next pages always load from the in-memory resource and never from the database. Stable paging results in better performance as the consumer application pages through the results.

Stable paging requires that the subsequent request for the next page is served by the same server process or cluster member that served the initial request for stable paging. The same server process is required because the resource is loaded in the memory of that server process, not in the other cluster members. If a subsequent request is load balanced to other members, the consumer application receives a 404 Not Found error. To avoid this error, ensure that in-session requests from a consumer application are all served by the same server process.

Example of stable paging

The client requests an asset resource:

```
GET/oslc/os/oslcasset?stablepaging=true
```

The request produces the following response:

```
303 Redirecting
Location: http://host:port/../olscasset?stableid=1234
```

The client performs a GET request on the redirect URI:

```
GET../oslcasset?stableid=1234
```

The response is the first page of the collection. The URI for the next page is embedded in the oslc:ResponseInfo object as part of the message. As the client moves through the pages, the pages expire after they load. If the client is at page 3, page 3 cannot be reloaded, nor can page 1 or page 2. Any attempt to reload those pages generates an HTTP 410 error. The loaded resource stays in memory until it expires based on an idle expiry time that is controlled by using the mxe.oslc.idleexpiry system property. The idle expiry time indicates the period that the resource was not accessed by the client. The client can request member properties by using the **oslc.select** clause and order the resulting collection by using the **oslc.orderBy** query parameter.

Setting the PATCHTYPE header to merge for partial resource updates

To minimize message size, for example for mobile applications, consumer applications can use PATCH requests to limit updates to changed data only. To update the child objects of an object structure, for example to update the work log that is associated with a work order, the consumer application sends a PATCH request, where the PATCHTYPE header property is set to merge.

Returning resource properties in HTTP response headers

Tivoli's process automation engine implementation of OSLC includes a new HTTP header property, called properties, that supports the return of resource attribute values in HTTP responses. A consumer application can set the propeties header and include a comma-separated string of properties in a POST, PUT, or PATCH request. The HTTP response includes th requested attribute values in the HTTP response. The properties header can be used, for example, to receive the number of a work order in the HTTP response to the POST request that created the work order.

Avoiding duplicate transactions following connection failures

Tivoli's process automation engine implementation of OSLC includes a new HTTP header property, called transactionid, that a consumer application can use to assign a unique ID to each transaction. The provider application stores the transaction IDs in the OSLC transaction table. Following a network connection failure, if the consumer application resends a POST or PATCH request, the server validates the transaction ID against the values in the transaction table. If the transaction ID already exists, the server returns a transaction conflict error in the HTTP response.

HTTP response codes:

OSLC uses standard HTTP response codes. For example, the HTTP 404 response is normally returned when a web page is not found, and in OSLC the 404 response code is returned when the resource cannot be found.

Some existing error codes are mapped to HTTP codes by default. An implementor can map additional response codes if required.

The following HTTP response codes are implemented by OSLC:

HTTP code	OSLC explanation
200	Success
201	Success. The response contains a link.
204	Resource successfully updated. There is no response entity.
400	Error handling request. This error might be due to the request content or URI. For example, there might be a business logic validation error on the server side.
401	Authentication failure.
403	Forbidden. The user password expired.
404	Resource cannot be found or an invalid resource type was provided.
405	HTTP method cannot be used for the resource.
406	Requested representation is not supported.
410	Stable resource page expired.
412	Resource on the client side is stale and must be refreshed from the server. The conditional update failed because the resource was updated by another user or process.
500	All other server errors.

The messages support the languages that are supported by Tivoli's process automation engine.

OSLC configuration:

The configuration of Maximo Asset Management as an OSLC provider application involves specifying resources within domains, the provisioning of service providers for those domains, security configuration, and setting up logging to collect information to assist with troubleshooting.

Specification of OSLC resources:

OSLC resources are based on integration object structures that include the primary business object and any child business objects that are used by an application. Attributes of the objects in the object structure are mapped to the RDF types and predicates for the resource and specify the namespaces for the resource types.

Each OSLC resource type has properties that are RDF predicates that can belong to a vocabulary specification that corresponds to the namespace for the properties. The attributes can map to the properties that are defined in vocabulary specifications such as the Dublin Core Metadata Initiative.

Each resource has a name, an associated object structure that is consumed by OSLC, and the default namespace URI for the resource. Each attribute to be integrated is mapped from an object structure with the following values:

- The value type and namespace serve as the RDF type for the resource.
- The name and namespace values serve as the predicate for the RDF predicate for the resource.

A resource shape document is an RDF/XML document that describes a resource, including all of its properties, attributes, and dependencies. A shape document in OSLC is an electronic way to see what a resource looks like including all of its dependencies, attributes, and properties. For example, a work order shape document includes properties that describe supporting resources such as asset, task, labor, and work log. Shape documents can cover a wide variety of areas such as assets, companies, purchase orders, and work orders. A shape document also shows what is required. A resource shape document can include links to the shape documents for child objects.

Example: Work order resource shape document

A work order resource shape document lists all of the properties, attributes, and dependencies of a work order. The following code shows an excerpt from a work order shape document called `oslcwodetail`. Three properties are included in this portion of the resource shape document but the document can have many more properties listed. The three properties in this portion of the shape document represent three types of nodes that are found in RDF documents. The first property, genforporevision, is a local node. The second property, multiassetlocci, is a blank node which means that no URI or literal is listed for this property in the RDF. The third property, asset, is a URI node.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:oslc="http://open-services.net/ns/core#"
   xmlns:dcterms="http://purl.org/dc/terms/>"
 <oslc:ResourceShape rdf:about="http://host/maximo/oslc/shapes/oslcwodetail">
  <oslc:describes rdf:resource="http://jazz.net/ns/ism/work/smarter_physical_
    infrastructure#WorkOrder"/>
   <oslc:property>
    <oslc:Property>
       <oslc:name>genforporevision</oslc:name>
       <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
       <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/asset/smarter_physical_
         infrastructure#genforporevision"/>
       <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
       <dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
         PO Revision</dcterms:title>
     </oslc:Property>
   </oslc:property>
   <oslc:property>
     <oslc:Property>
       <oslc:name>multiassetlocci</oslc:name>
       <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-many"/>
       <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/asset/smarter_physical_
         infrastructure#multiassetlocci"/>
       <oslc:valueType rdf:resource="http://open-services.net/ns/core#LocalResource"/>
       <oslc:representation rdf:resource="http://open-services.net/ns/core#Inline"/>
       <oslc:valueShape rdf:resource="http://host/maximo/oslc/shapes/oslcwodetail/multiassetlocci"/>
       <oslc:range rdf:resource="http://jazz.net/ns/ism/asset/smarter_physical_
         infrastructure#MultuAssetLocationCI"/>
     </oslc:Property>
   </oslc:property>
   <oslc:property>
     <oslc:Property>
       <oslc:name>asset</oslc:name>
       <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
       <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/ism/work/smarter_physical_
         infrastructure#asset"/>
       <oslc:valueType rdf:resource="http://open-services.net/ns/core#Resource"/>
       <oslc:representation rdf:resource="http://open-services.net/ns/core#Reference"/>
       <oslc:valueShape rdf:resource="http://host/maximo/oslc/shapes/oslcasset"/>
       <oslc:range rdf:resource="http://jazz.net/ns/ism/asset/smarter_physical_infrastructure#asset"/>
```

```
      <dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
        Asset</dcterms:title>
    </oslc:Property>
  </oslc:property>
```

Service provider provision:

OSLC resources are enabled within a domain and are available to consuming applications through the OSLC service provider for the domain. A single service provider for each domain is supported.

In Maximo Asset Management, you can discover what service providers are available by entering the `http:/host:port/maximo/oslc/sp` URI in a browser. The response includes the URI for the service provider document for each domain that is configured for OSLC integration.

The URI for the service provider document for the work management domain, for example, is `http://host:port/maximo/oslc/sp/WorkManagment`. The service provider document is in RDF/XML format and describes the available resources and namespace mappings, saved queries, and the operations that are supported for the resources that it supports. An OSLC consumer application can use the service provider document to determine which resources are available and what services it supports, such as query or creation. The current implementation supports create, request, update, and delete operations at data level but resources in delegated user interface scenarios are not supported.

In the following sample response, the OSLC service provider is referred from the `rdfs:member` property. The service provider document for the domain shows the URI for work management:

```
<rdfs:member rdf:resource="http://host/maximo/oslc/sp/WorkManagement">

<rdf:RDF>
<rdf:Description rdf:about="http://host/maximo/oslc/sp">
<rdfs:member rdf:resource="http://host/maximo/oslc/sp/WorkManagement">
</rdf:Description>
</rdf:RDF>
```

Namespaces

OSLC defines common namespaces. The prefixDefinition property exposes all the prefix-to-namespace mappings that the service provider uses to describe the resources that it manages. The following table shows a sample of the namespaces that are available:

Prefix	Namespace
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
oslc	`http://open-services.net/ns/core#`
dcterms	`http://purl.org/dc/terms/`
asset	`http://open-services.net/ns/asset#`
foaf	`http://xmlns.com/foaf/0.1/`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
rr	`http://jazz.net/ns/ism/registry#`
spi	`http://jazz.net/ns/ism/asset/smarter_physical_infrastructure#`

The following excerpt from the service section of the service provider document shows the OSLC and RDF namespaces:

```
<oslc:ServiceProvider rdf:about="http://host:7001/maximo/oslc/sp/WorkManagement">
  <oslc:prefixDefinition>
    <oslc:PrefixDefinition>
      <oslc:prefixBase rdf:resource="http://open-services.net/ns/core#"/>
      <oslc:prefix>oslc</oslc:prefix>
    </oslc:PrefixDefinition>
  </oslc:prefixDefinition>

  <oslc:prefixDefinition>
    <oslc:PrefixDefinition>
      <oslc:prefixBase rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
      <oslc:prefix>rdf</oslc:prefix>
    </oslc:PrefixDefinition>
  </oslc:prefixDefinition>
```

The following table shows a sample mapping for a work order that uses common namespaces:

OSLC prefix	Existing prefix
oslc:shortTitle	wonum
dcterms:title	description
dcterms:description	description_longdescription
dcterms:creator	reportedby
dcterms:created	reportdate
dcterms:identifier	workorderid

The following table shows the mappings for the person object that is referred to from dcterms:creator in the workorder.

OSLC prefix	Existing prefix
foaf:name	displayname
foaf:givenName	firstname
foaf:familyName	lastname

Creation factory and query operations

An OSLC service provider supports the creation factory and query capability operations for the resources that are available in the service provider document. A creation factory provides the oslc:creation Creation URI that you use to create new resources with a HTTP POST request. The oslc:queryBase query URI enables the selection of a resource collection that is managed by the service provider. When the resource is obtained, either by query or creation, the resource can be updated or deleted.

If the resource supports creation, there can be one creation factory operation. The following excerpt from a service provider document shows the creation factory operation, the URI for the resource shape, and the URI for the creation resource operation that creates the shape.

```
<oslc:creationFactory>
 <oslc:CreationFactory>
   <oslc:resourceType rdf:resource="http://jazz.net/ns/ism/work
/smarter_physical_infrastructure#WorkOrder"/>
  <oslc:resourceShape rdf:resource="http://host:port/maximo/oslc/shapes/oslcwodetail"/>
  <oslc:creation rdf:resource="http://host:port/maximo/oslc/os/oslcwodetail"/>
  <oslc:label>Create WorkOrder</oslc:label>
```

```
   <dcterms:title>OSLC creation factory for WorkOrder</dcterms:title>
 </oslc:CreationFactory>
</oslc:creationFactory>
       ......
```

The query URI is oslc:queryBase, and the following example shows a search for a work order by using the request:

```
<oslc:queryBase rdf:resource="http://host/maximo/oslc/os/oslcwodetail"/>
```

```
   <oslc:queryCapability>
         <oslc:QueryCapability>
           <oslc:resourceType rdf:resource="http://jazz.net/ns/ism/work/smarter_physical_
              infrastructure#WorkOrder"/>
           <oslc:queryBase rdf:resource="http://host/maximo/oslc/os/oslcwodetail"/>
           <oslc:labelQuery> WorkOrder</oslc:label>
           <dcterms:title>OSLC query capability for WorkOrder</dcterms:title>
         </oslc:QueryCapability>
       </oslc:queryCapability>
```

Saved queries:

When an OSLC resource is defined based on an object structure that references an application, all the public saved queries from the application are made available through the service provider. The queries are exposed as OSLC query capabilities in the service provider document for that OSLC resource.

Object structures can be optionally connected to an application. In the application of the resource, you define custom queries and save those queries for later use. You can make the queries public so that the queries are available to all users who are authorized to use that application. To run a saved query, you perform an HTTP GET request on the query base URI that is provided in the query capability for the saved query. A list of OSLC resources that match the saved query are returned.

Each query capability property exposes a URI called oslc:queryBase property that enables a consumer to search the resources that are managed by the service provider. Each query capability supports only one resource type called oslc:resourceType. The OSLC specification allows a query capability to support multiple resource types. The query capability lists all of the public saved queries for the application that is registered with the object structure that implements the OSLC resource. If there are no public saved queries for the registered application or if no application was registered, only one query capability provides the generic query collection URI. There is always at least one query capability for each resource type that is supported by the service provider.

Example: Saved queries in the Work order application

You create a query in the Work Order application to search for work orders that you own. You make the query public so that the query appears in the service provider document as `savedQuery=OWNER+IS+ME`. TheOSLC top-level object structure must also be work order.

```
<oslc:service>
      <oslc:Service>
        <oslc:queryCapability>
          <oslc:QueryCapability>
            <oslc:resourceType rdf:resource="http://jazz.net/ns/ism/work/smarter_physical_
infrastructure#WorkOrder"/>
            <oslc:queryBase rdf:resource="http://host/maximo/oslc/os/oslcwodetail?
savedQuery=OWNER+IS+ME"/>
            <oslc:label>Query OWNER IS ME</oslc:label>
            <dcterms:title>OSLC query capability for My Work Orders</dcterms:title>
          </oslc:QueryCapability>
        </oslc:queryCapability>
        <oslc:queryCapability>
          <oslc:QueryCapability>
```

```
            <oslc:resourceType rdf:resource="http://jazz.net/ns/ism/work/smarter_physical_
infrastructure#WorkOrder"/>
            <oslc:queryBase rdf:resource="http://host/maximo/oslc/os/oslcwodetail"/>
            <oslc:label>Query WorkOrder</oslc:label>
            <dcterms:title>OSLC query capability for WorkOrder</dcterms:title>
          </oslc:QueryCapability>
        </oslc:queryCapability>
        ......
```

OSLC security:

Authentication and authorization support for OSLC services is provided by the Maximo Asset Management security framework. J2EE-based authentication such as LDAP is supported through the application server. The application server also provides support for HTTPS.

Native authentication

The consumer request can provide the *user:password* values that are base64 encoded and are in the MAXAUTH HTTP header property.

J2EE authentication

To configure J2EE authentication, you modify the `web.xml` file, set security constraints and set the useAppServerSecurity property to true.

You modify the `web.xml` file for the maximouiweb web module by uncommenting the following lines:

```
<!--
<servlet>
 <display-name>OSLC Servlet for Web App</display-name>
 <servlet-name>OSLCServlet</servlet-name>
 <servlet-class>
    com.ibm.tivoli.maximo.oslc.provider.MaximoOslcProviderServlet
  </servlet-class>
 <init-param>
  <param-name>char_encoding</param-name>
  <param-value>UTF-8</param-value>
 </init-param>
</servlet>
-->


<!--servlet-mapping>
 <servlet-name>OSLCServlet</servlet-name>
 <url-pattern>/oslc/*</url-pattern>
</servlet-mapping
```

To configure security constraints, you enter the following code for `<web-resource-collection>`:

```
<web-resource-collection>
 <web-resource-name>OSLC Servlet</web-resource-name>
 <description>
    OSLC Object Structure Servlet accessible by authorized users
  </description>
 <url-pattern>/oslc/*</url-pattern>
 <http-method>GET</http-method>
 <http-method>POST</http-method>
 <http-method>PUT</http-method>
 <http-method>HEAD</http-method>
</web-resource-collection>
```

The useAppServerSecurity property must be set to true:

```
<env-entry>
 <description>
    Indicates whether to use Application Server security or not
  </description>
 <env-entry-nameuseAppServerSecurity</env-entry-name>
  <env-entry-typejava.lang.String</env-entry-type>
 <env-entry-valuefalse</env-entry-value>
</env-entry>
```

Explicit login and logout

If the consumer application needs to run explicit login commands, you use the following request:

```
GET /maximo/oslc/login
```

If you are using native authentication, you must add the MAXAUTH HTTP header property to the login request. If the consumer application needs to run explicit logout commands, you use the following request:

```
GET /maximo/oslc/logout
```

Authorization

Authorization control is provided at the object structure level of the resource. You associate the object structure to an application in the Object Structure application. The security processing of the resource data is then based on both the configuration of security of the application and the user group of the user who made the request. When OSLC resources are processed, any object attribute that is configured as hidden through security is not included in the response to an OSLC request.

OSLC logging:

Log correlation can record information when OSLC resources are created or updated. The information that is logged includes the total response time and the size of the resource in bytes. The information can be used for performance analysis. For example, you can use the log information to analyze response times.

To enable log correlation so that information is recorded during GET, POST and PUT requests, go to **System configuration** > **Platform configuration** > **System properties application** and ensure that the mxe.logging.CorrelationEnabled property is set to 1.

The following table shows some of the information that might be logged when you enable the mxe.logging.CorrelationEnabled property. Some of the attributes in the table might not be available for every request.

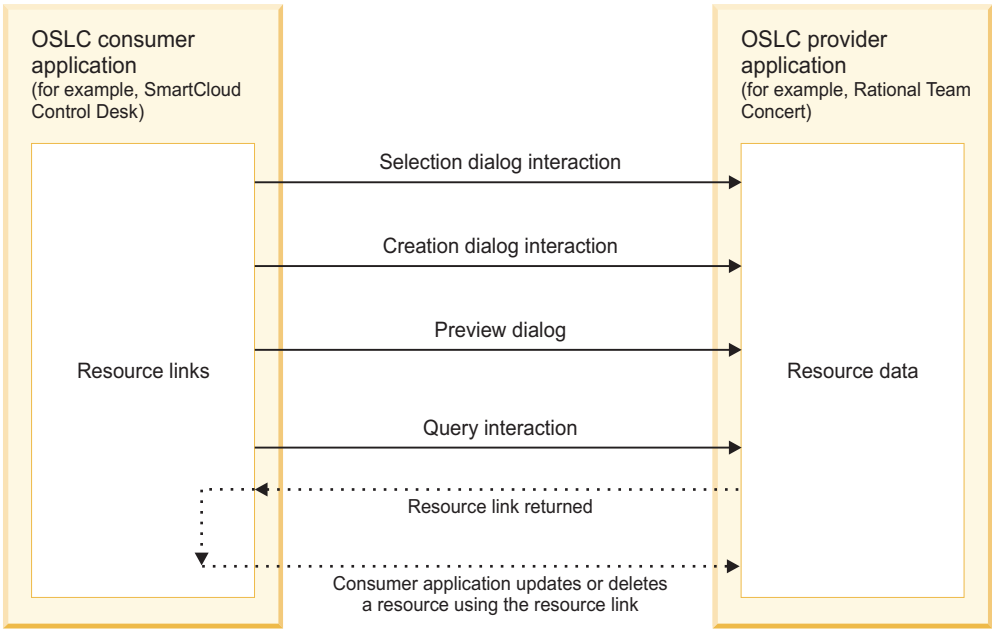
Attribute	Description
ClientIP	IP address of the client that makes the request of the server through OSLC.
InvokeAndSerTime	Time that is taken for invocation and serialization.
SendResponseTime	Time that is required for the server to respond to a client request.
RequestURI	Universal resource identifier of the client.
RequestParams	Parameters that are passed in the URI of the client.

Attribute	Description
ElapsedTime	Total time that is required from client request to server response.
LoginID	User name that is associated with a client authentication request.
ResourceSize	Size of the resource in bytes.
EndUserClientIP	IP address of the end user who makes the request of the server.
EndUserMetaData	Data that is associated with the actual end client user. Requests are often delivered through an intermediary server. This attribute, when present, always contains end user data.
ClientPort	Port number that the client uses to make a request of the server.

Integrating as an OSLC consumer

Your application, an OSLC consumer, can be configured to support three interaction types; selection dialog, creation dialog, and query. The resource links are obtained by the consumer application from the provider application.

The following figure illustrates the interactions between the consumer and provider applications. As an OSLC consumer, the application can query or create resources in the provider application and retain the links to those resources. With the links, the consumer application can make requests to the provider application to query, update, or delete the resources.



Creation of OSLC provider records:

You must create a provider record in the OSLC Providers application for each product that you register in the Registry Services Resource Registry. You can use the provider record to create interactions that enable users to share and update data between products.

Registry Services:

Registry Services is a Jazz™ for Service Management integration service that provides the mechanisms through which Open Services for Lifecycle Collaboration (OSLC) services, resources, and consumers can use the OSLC interface to work in an integrated service management environment.

Registry Services uses the provider registry to track the capabilities that are available in the environment and uses the resource registry to track the resources that can be managed for the environments. It tracks the resources that can be managed for the environments by using the resource registry. Tivoli's process automation engine uses the provider registry and the resource registry to integrate multiple products.

Provider registry

The provider registry is a Registry Services directory that contains services that are provided by multiple products across different domains. The provider registry tracks all the service providers in an environment, provides information about how to contact them, and lists the type of action they perform.

As an OSLC consumer, when you create a selection window, a creation window, or a query interaction, you can configure the interaction to discover service providers in the provider registry.

The OSLC provider currently offers no support for registering the application's service providers to the provider registry.

Resource registry

The resource registry is a Registry Services directory that contains resources that are managed or tracked by multiple products across different domains.

As an OSLC consumer, you can use the resource registry to query resources. When you create a query interaction, you can configure the public URI of the provider record to point to the resource registry, and you can also configure the default PROVIDERREGISTRY end point.

The OSLC provider currently offers no support for registering the application's resources to the resource registry.

Service providers:

A service provider is a container or collection of related data, such as a project, a user database, or a module. Service providers support the grouping of similar resources, such as defects or tasks, that can be configured for integration.

A service provider in an OSLC provider application contains the resource data that can be linked to consumer application data through integration of the applications. To integrate a consumer application and a provider application, the consumer must discover or identify the service providers that are available in the provider application.

The OSLC Providers application supports the following ways of discovery of service providers:

- Catalog list

- Single service provider
- Provider registry

To use the catalog list or the single service provider for discovery, in the OSLC Providers application, you specify a URI in the **Service Provider List URI** field. The URI can link to a service provider catalog, a service provider listing, or an individual service provider.

To use the provider registry for discovery, in the OSLC Providers application, you select the **Use Provider Registry** check box. In the End Points application, a predefined HTTP endpoint that is named PROVIDERREGISTRY is used to identify the connection information to the provider registry.

The PROVIDERREGISTRY endpoint specifies a provider registry where OSLC provider applications can register service provider data. The endpoint requires some configuration before it can be used. The HTTP endpoint for the provider registry is distinct from the OAuth or HTTP endpoints that are used during the running of interactions between integrated applications.

Resource types and shape documents:

A *resource type* identifies the type of data, such as a change request, that is linked between integrated applications. A resource type can have an associated shape document. A shape document is similar to an XML schema in the way that it defines the data structure of the resource.

In the OSLC Providers application, you use the **Add/Modify Resource Types** action to create resource types. You create a resource type for each type of data that at least one provider supports, and that you want to link to. The provider application must support the resource type through a service provider for the resource type to be available for an OSLC interaction.

A resource type can be used with multiple interactions for a provider. Resource types are available to use with multiple OSLC providers that support the same resource type.

Resource data from the provider application can be made available to the consumer application as usages. A *usage* is a subclass of the resource data. For example, a resource type of change request might have usages of defect and task.

A shape document is a Resource Description Framework (RDF) file that provides a description of the resource data types that can be used in an interaction. When you create a resource type, you can import a shape document for the resource type. The shape document contains a list of attributes of the resource type.

Endpoints in OSLC integration:

An *endpoint* is the entry point to a service, a process, an application, or a topic destination. Two types of endpoints can be used in OSLC integration. An integration endpoint is used for OSLC integration between a consumer application and a provider application. It specifies how a consumer application communicates with the OSLC provider application. The integration endpoint can be an OAuth endpoint or an HTTP endpoint. A provider registry endpoint is used only if the method specified for service provider discovery is through the Provider Registry.

Integration endpoints

When you create an OSLC provider record, you must specify the endpoint for the provider application. The endpoint is used for all interactions between the consumer application and the provider application. OSLC integration with some providers, such as Rational Team Concert™, must use an endpoint with a handler type of OAuth. Other providers might require an HTTP endpoint, or support either an HTTP endpoint or an OAuth endpoint.

OAuth is an open standard for authorization between sites or applications. OAuth enables the sharing of resources that are stored on one site with another site without having to provide complete authentication credentials. Additional information about OAuth is readily available on the web.

You create the endpoint definition in the End Points application. The endpoint specifies the attribute values from the provider application that enable the integration and data linking.

Important: Use only English-language characters and numerals in an OAuth endpoint name.

The values of the OAuth endpoint properties are based on the configuration of the provider application.

Table 39. Properties of OAuth endpoints

Property	Description
ACCESSTOKENURL	You get the access token URL from the OSLC provider application. This value is required.
AUTHORIZATIONURL	You get the authorization token URL from the OSLC provider application. This value is required.
CONSUMERKEY	The OAuth key that is to be used by the consumer application, from the OSLC provider application. This value is required.
CONSUMERSECRET	The OAuth secret or password that is to be used by the consumer application, from the OSLC provider application. This value is stored in an encrypted format. It is required.
COOKIES	The cookies to pass through to the endpoint.
HEADERS	The header properties to pass through to the endpoint. For example, the language code of the consumer side of the integration might be included in the header properties.
HTTPMETHOD	The HTTP method that is used during the request. The default value is GET. This value is required. Ensure that **Allow Override** is selected for the HTTPMETHOD property.
REQUESTTOKENURL	The URL of the OAuth security provider. You get this value from the OSLC provider application. This value is required.
URL	The URL of the service provider usage type. This value is required. Ensure that **Allow Override** is selected for the URL property.

Provider registry endpoint

You can specify that your product does discovery of service providers in the OSLC provider through the Provider Registry. In this case, you use a predefined HTTP endpoint for discovery, in addition to the OAuth or HTTP endpoint that is used for integration. The name of the endpoint, PROVIDERREGISTRY, is registered in the mxe.oslc.prqueryep system property.

The IBM open service delivery platform (OSDP) has a version of their Provider Registry that is available under the Transparent Development program. For more information about the registry and to download the current version, see https://www.ibm.com/developerworks/servicemanagement/iosdp/index.html. When implemented, this registry could be configured as the PROVIDERREGISTRY endpoint.

The PROVIDERREGISTRY endpoint is predefined in the End Points application, but not all properties are configured. You must fully configure the endpoint to support service provider discovery through the Provider Registry.

Table 40. Properties of the PROVIDERREGISTRY HTTP endpoint

Property	Description
CONNECTTIMEOUT	The amount of time, in seconds, to wait for the connection before giving a timeout error.
COOKIES	The cookies to pass through to the endpoint.
HEADERS	The header properties to pass through to the endpoint. For example, the language code of the consumer side of the integration might be included in the header properties.
HTTPEXIT	The Java exit class that is provided to support processing that is specific to the Provider Registry.
HTTPMETHOD	The HTTP method that is used during the request. The default value is GET. This value is required. Ensure that **Allow Override** is selected for the HTTPMETHOD property.
PASSWORD	If the registry is secured, the password for the corresponding user name is required.
READTIMEOUT	The amount of time, in seconds, to wait for a response to a request made by the consumer application before giving a timeout error.
URL	The URL of the Provider Registry. Required. The default value is https://oslc-registry/oslc/pr. The value must be updated to reflect the local implementation of the registry. Ensure that **Allow Override** is selected for the URL property.
USERNAME	If the registry is secured, the user name and the corresponding password are required.

When you configure the PROVIDERREGISTRY endpoint definition, you must specify values for at least these two properties:

- HTTPMETHOD: Configured to GET.

- URL: Has a default value. Update the value to reflect the local implementation of the registry.

You also might need to configure the USER and PASSWORD values, depending on the security configuration of the Provider Registry.

Single sign-on

If you plan to use single sign-on, you must set the COOKIES property for your HTTP endpoint to `LtpaToken2`.

Designing an OSLC interaction:

OSLC integration establishes links between the data in one of your user applications and the data in an external OSLC provider application. You use the OSLC Providers application to integrate applications by designing OSLC interactions.

The OSLC Providers application supports the creation of OSLC interactions that are generated from the user interface. You can design the following three types of interactions that let your users link their application data to provider resource data:

- A creation interaction lets the user create resource data in the OSLC provider application and link that resource data to an application record.
- A selection interaction lets the user select existing resource data in the OSLC provider application and link that resource data to an application record.
- A query interaction lets the user select a pre-defined query in the OSLC provider application and retrieve resource data.

The following records must be in place before you can design an interaction:

- An OAuth or HTTP integration endpoint record for the provider application must exist in the End Points application.
- A record for the OSLC provider application must be defined in the OSLC Providers application.
- One or more provider resource types must be available. You add resource types on the Add/Modify Resource Types window in the OSLC Providers application.

The Create OSLC Interaction wizard in the OSLC Providers application takes you through three steps to design an OSLC selection interaction. The wizard adds an extra step for an OSLC creation interaction and two extra steps for an OSLC query interaction.

Example: Designing an OSLC creation or selection interaction:

You want to design an OSLC interaction so that users of the Service Requests application can create a defect in Rational Team Concert, the provider application. The defect that is created is linked to a service request in the Service Requests application.

Preparatory work

In the OSLC Providers application, you create a provider record for Rational Team Concert. You specify the OAuth endpoint and the public URI for Rational Team Concert, and you specify how the service provider is identified.

By using the **Add/Modify Resource Types** action, you add the resource type or types that are supported in Rational Team Concert and specify that users can link from the Service Requests application.

Define interaction and select usage URI: step 1

You select the **Create OSLC Interaction** action.

You specify `Create Defect` as the name of the OSLC interaction for the Rational Team Concert record. You select **CREATIONDIALOG** as the interaction type.

You select a resource type, such as **change request**, from the value list. The list is populated with the resource type or types that you created earlier.

The Rational Team Concert login page is displayed because you must log in to Rational Team Concert to continue.

After you log in, the **Usage URI** value list is populated from Rational Team Concert with the resource usage types for the selected resource type. You select the defect usage type. You can optionally specify a usage URI.

Select service provider and association property: step 2

In step 2, you specify whether all Rational Team Concert service providers are available for the interaction, or whether only a single service provider is available. You can also identify an association property, so that a link can be established from the provider application, Rational Team Concert, to the consumer application, Service Requests.

You use the default selection, which is that all service providers are available. When a user of the Service Requests application initiates the interaction in the Service Requests application and multiple service providers support the selected resource and usage combination, a selection window opens. For example, the selection window can show a list of Rational Team Concert projects that support defects. The user can select the service provider or container in which to create the defect.

The other option is to specify an individual service provider URI. In that case, the Service Requests application connects directly to the service provider that you specify.

To enable a link from Rational Team Concert to the Service Requests application, you must specify the association property. The association property is tied to the Rational Team Concert resource. If a shape document is associated with the resource type, values that you can choose for the association property come from the shape document. If no shape document is associated with the resource type, you can specify an association property. The association property is displayed in the Rational Team Concert user interface, and the property is populated with a link to the service request. A Rational Team Concert user can click the link to open the Service Requests application and view the related service request in Rational Team Concert.

Select consumer application for OSLC interaction: step 3

In step 3, you specify the consumer application that will interact with the provider application, Rational Team Concert. The consumer application is the Service Request application.

The default setting is that user interface changes in the consumer application are generated automatically by the wizard. If you maintain the default setting, you must specify the application tab to which the interaction table and button are added.

You specify that the tab, table window, and button are to be added to the **Related Records** tab of the Service Requests application. By default, the push-button name is the name that you specified for the interaction: Create Defect. You can change the button name.

For the button to be visible to users, you must specify the security groups that are authorized to run the interaction. Only users in the groups that you authorize can see the button and create a defect in Rational Team Concert.

Alternatively, you can clear the **Create Interaction Tab** check box and use the Application Designer to create all user interface changes in the Service Requests application.

Specify interaction field mappings: step 4

Step 4 is available for creation interactions only. You can optionally map data from the consumer application to target fields in the provider application. When the user clicks the button to create a resource, the fields are mapped to the provider window that is displayed in the consumer application. For example, you want Service Requests users to see the service request number and description that is prefilled in fields on the Rational Team Concert Create Resource window. You can provide more field mappings, or revise the field mapping, even after you complete the interaction.

The shape document for the resource identifies attributes that support links. From your application, select only fields that contain links to map to target fields that support links. Ensure that the target field is a field that is visible in the user interface of the provider application window so that the mapped data is visible.

You want to map the service request description to the defect description in the Rational Team Concert window. You specify `:description` as the source mapping expression, and `description` as the target resource property.

Create multiple interactions

You can create multiple interactions for a consumer application. For example, you might also want to create a selection interaction for the Service Requests application. In the selection window, a user can click a button to select an existing resource, such as a defect, in Rational Team Concert. If you create multiple interactions for the same resource type and usage combination in the same application tab, the push buttons are added to the same tab and table window.

If you also design a selection interaction for the Service Requests application, users have the following options:

- Create a defect, and link the service request to the new defect.

- Select an existing defect, and link the service request to that.

You also might design an additional creation interaction for a supported resource type or usage type, such as task. A different resource type and usage combination generates a separate tab and table window for the additional interaction button.

Review new interactions in the consumer application

After you complete the interaction, you must log in again. The new login grants the updated security access that you specified in step 3. After you log in, you navigate to the Service Requests application to review the user interface changes. At any time, you can use the Application Designer application to refine or add to the user interface changes that were auto-generated in the Service Requests application.

You can test the button or buttons to ensure that the Rational Team Concert window is shown in the Service Request application as expected. If the window is not shown, or data is in unexpected fields, faulty data mapping might be the cause. You can review and revise data mapping for the interaction in the Mapping table of the OSLC provider record.

If users of the Service Request application cannot see the newly added buttons that link to Rational Team Concert, ensure that the users have logged in again and that they belong to the security groups that you authorized. Also ensure that the users have valid Rational Team Concert login credentials.

Modification of OSLC creation or selection interactions

After you complete the design of the OSLC interaction, the changes that you can make to it are limited. You can update the association property, and you can revise the field mapping.

Specifying a new association property establishes a new property name that provides a link from the provider application to the consumer application.

In the Interaction Mappings table, you can revise any existing mapping, delete mapping, or add mapping. The shape document for the resource identifies attributes that support links. From the consumer application, you select only the fields that contain links to map to target fields that support links. Ensure that the target field is one that is visible on the user interface of the provider application window, so that the mapped data is visible.

You also can specify a Java class to use as part of the mapping logic. You can use a Java class when the field mapping definitions that you provide are not sufficient. Specify the Java class in the **Interaction Mapping Class** field. A mapping class can be provided whether field mapping exists or not.

Deletion of OSLC creation and selection interactions

Deletion of the interaction is possible only if there are no existing links within the consumer application that reference the interaction. All related application links must be deleted before an interaction can be deleted from an OSLC provider record.

When you delete an interaction, the corresponding user interface changes that were made when the interaction was designed are updated. The button for the

interaction is removed from the application. The table window and tab are also removed, but only if the wizard-created tab and table window where the button existed does not contain buttons for other interactions. The relationship and signature option that were created for the interaction are also deleted.

If you manually configured the user interface changes for the interaction, you must use the Application Designer to remove those changes.

Orphan links

Links that are generated between the object record for the consumer application and the provider application resource are stored in the OSLCLINK database table.

If you delete an object record that has links, the OSLCLINK table then contains one or more orphan links that are associated with the deleted object record.

The OSLCDeleteLinks cron task deletes orphan links. By default, the OSLCDeleteLinks cron task is set to run once a day. You can view and edit the cron task in the Cron Task Setup application

Example: Designing an OSLC query interaction:

You want to design an OSLC interaction so that users of the Assets application can run a query and retrieve resource data from a provider, such as IBM Tivoli Application Dependency Discovery Manager.

Preparatory work

In the OSLC Providers application, Resource Registry is the default provider for query interactions. The PROVIDERREGISTRY endpoint is specified as part of the default settings, but you must configure the end-point properties and specify the public URI for Resource Registry. You must also specify how the service provider is identified.

By using the **Add/Modify Resource Types** action, you add the resource type or types that are supported in Resource Registry and associate a shape document with the resource type if available. You can also specify that users can link to the Resource Registry from the Assets application.

Define interaction and select usage URI: step 1

You select the **Create OSLC Interaction** action.

You specify `COMPUTER` as the name of the OSLC interaction and select **QUERYCAPABILITY** as the interaction type.

You select a resource type, such as **COMPUTERSY**, from the value list. The list is populated with the resource type or types that you created earlier.

You can optionally specify a usage URI. The **Usage URI** value list is populated from the Resource Registry with the resource usage type or types for the selected resource type. If you do not specify a usage URI, the default usage type for the resource type is used.

Select service provider and link property: step 2

In step 2, you specify whether all service providers are available for the interaction, or whether only a single service provider is available. You can also identify a link property and a link label, so that a link can be established from the provider application, Resource Registry, to the consumer application, Assets.

You use the default selection, which is that only the Resource Registry service provider is available. When a user of the Assets application initiates the interaction by running a query and multiple service providers support the selected resource and usage combination, a selection window opens. For example, the selection window can show a list of service providers that contain data that is on the specified resource. The user can select the service provider on which to base the query.

The other option is to specify an individual service provider URI. The query is run based on the service provider that you specify. The link property is used to retrieve the resource data from the Resource Registry for the provider. The link and the link label are displayed to the user in the consumer application when the query results are provided.

To enable a link from Resource Registry to the Assets application, the provider application must support the use of a link property. The link property is tied to the resource in the Resource Registry. If a shape document is associated with the resource type, values that you can choose for the link property come from the shape document. If no shape document is associated with the resource type, you can specify a link property and link label.

Select consumer application for OSLC interaction: step 3

In step 3, you specify the product application that interacts with the provider application, Resource Registry. The consumer application can be any application in the product, but this example uses the Assets application.

The default setting is that the push button is generated automatically in the application that is specified. If you maintain the default setting, you must specify the application tab to which the interaction button is added to support the query interaction.

You specify that the button is to be added to the **Main** tab of the Assets application. By default, the push-button label is the same as the name that you specified for the interaction: Computer. You can change the button label.

For the button to be visible to users, you must specify the security groups that are authorized to run the interaction. Only users in the groups that you authorize can see the button and run the query to retrieve the resource data.

Specify queries and query parameters: step 4

In step 4, you build the queries for the interaction. The queries are used to query the resource data in the Resource Registry. The Select clause identifies the attributes that are selected from the resource. You can specify an asterisk (*) to retrieve all values or specify a comma-separated list of resource attributes from the shape document to filter the list of values to be retrieved.

You can define the WHERE condition of the query by using one of the following methods:

- Implement a Java class to define the WHERE condition.
- Populate the WHERE condition.
- Configure the WHERE condition by using the parameter values. The parameter values are selected from the main object of the application and compared against the attributes of the resource data in the provider application.

The queries are run in the order that is specified in the queries table. If the first condition is not met, all subsequent queries are processed in sequence until a query condition is met. When a condition is met, the resource data is retrieved and returned to the user in the consumer application. You can also identify the **orderby** query parameter that orders the data returned in the response of the query.

Queries that are created during the design of the interaction are not validated. Therefore, you might create queries that generate messages during run time. For example, you might create the following query condition:

Query	Query Type	Property	Operand	Mapped Expression	Is Literal?
10	Parameter	http://open-services.net/ns/crtv#%7Dmanufacturer	IN	IBM, Dell	Checked

But during run time, an error is generated because the query is searching for one attribute `"IBM, Dell"` rather than two attributes `"IBM","Dell"`. The following example shows the correct attribute format:

Query	Query Type	Property	Operand	Mapped Expression	Is Literal?
10	Parameter	http://open-services.net/ns/crtv#%7Dmanufacturer	IN	"IBM","Dell"	Checked

Specify the interaction group: step 5

In step 5, you can optionally add the query interaction to an interaction group and specify a condition for the group if one does not exist.

An interaction group allows you to group multiple related queries together under a single user interface control. By specifying an interaction group, you allow the user to execute multiple interactions to different providers based on the condition that is associated with the interaction within the group.

All the interactions in the group are based on the same main object and resource type. If you add a query interaction to an interaction group, that interaction is only run as part of the group. The query interaction is not run independently. The order of interactions in the group determines the sequence in which the query for the interaction group is run.

You specify the `CICOMPUTERSY` interaction group and the `XYZ` condition. If you created an interaction group, you can specify extra details for the interaction group by using the **Add/Modify Interaction Groups** action. In the Assets application, when the user clicks the button to run a query and a condition is met, the query results are displayed in the Preview window.

Review new interactions in the consumer application

After you create the query interaction, you must log in again. The new login grants the updated security access that you specified in step 3. After you log in, you can navigate to the Assets application to review the user interface changes.

You can test the button to ensure that it is shown in the Assets application as expected. You can run the query to verify that the query is valid and that resource

data is returned. If no results are returned, the query parameters might be incorrect. Review the query parameters in the OSLC Providers application and update them as needed.

If users cannot see the newly added button after they log in again, ensure that the users belong to the security groups that you authorized.

Modification of OSLC query interactions

After you complete the design of the OSLC query interaction, the changes that you can make to it are limited. You can update the link property, update the query parameters, and you can revise the field mapping.

Specifying a new link property establishes a new property name that provides a link from the provider application to the consumer application.

In the Query Mappings table, you can revise any existing mapping, delete mapping, or add mapping. The shape document for the resource identifies attributes that support links. From the consumer application, you select only the fields that contain links to map to target fields that support links. Ensure that the target field is one that is visible on the user interface of the provider application window so that the mapped data is visible.

You also can specify a Java class to use as part of the mapping logic. You can use a Java class when the field mapping definitions that you provide are not sufficient. Specify the Java class in the **Query Mapping Class** field. A mapping class can be provided whether field mapping exists or not.

The SELECT clause of the query interaction can be updated after the interaction is created.

Deletion of OSLC query interactions

Deletion of the query interaction is possible only if the interaction is not tied to an interaction group. You must remove the interaction from a group before you delete the interaction.

When you delete an interaction, the corresponding user interface changes that were made when the interaction was designed are updated. The button for the interaction is removed from the application. The relationship and signature option that were created for the interaction are also deleted.

Creating interaction groups:

You can use interaction groups to collate multiple query interactions under a single user interface control, such as a button. These query interactions can be for a single provider or multiple providers.

Before you begin

Before you can create an interaction group, you must create one or more query interactions.

About this task

Each query interaction in the group is executed and returns data to the user based on its conditions. If a query interaction contains a condition that is based on the current application data, the interaction group uses that data to execute only the appropriate queries. The query results from multiple queries are combined and displayed in the resulting view.

To create an OSLC interaction, you must be in the base language. You can also create an interaction group in the query interaction wizard.

Procedure

1. In the OSLC Providers application, select the **Add/Modify Interaction Groups** action.
2. Add a new row, and specify a name for the interaction group and the main object that you want to associate with the group.
3. Optional: If you want to associate the interaction group with multiple providers, select the **Support the Combined View of Multiple Providers** check box.
4. On the **Interactions** tab, specify at least one interaction to be run as part of the group. The interactions that are available for selection are based on the main object that is specified for the interaction groups.
5. On the **Applications** tab, specify the application to which you want to add the interaction group.
6. Optional: If the main object for the interaction group is not a main object for the application that is specified, you must specify a relationship. If a relationship does not exist between the application and the main object that you specified for the interaction group, you must create or modify a relationship on the **Relationships** tab in the Database Configuration application.
7. Specify a label for a button, a Detail Menu item, or both depending on where you want the interaction to be available from in the consumer application.
8. On the **Security Groups** tab, select the security group that applies to the application and interaction group.
9. To add an interaction group to a Detail Menu item, select the **Application Menu** tab, specify the location in the consumer application where the group is to be added, click **OK**, and save the group. The existing fields in the consumer application are shown in the **Select Detail Menu** dialog box.

Example: Running an OSLC interaction:

As an application user, you can run OSLC interactions between applications that an integration designer integrated. You start theOSLC interaction by clicking a button in the consumer application. Depending on the type of interaction, you can create a resource record in the external provider application or link to an existing resource in the provider application.

In this example, your application is the consumer and Rational Team Concert is the provider. However, this process can also work with anyOSLC provider.

You use the Service Requests application to enter requests for fixes to the software that your team is developing. Your integration designer used the OSLC Providers application to integrate the Service Requests application with the Rational Team Concert application. Two OSLC interactions were created. In the Service Requests

application, there is now a tab that is labeled **Defects** in the **Related Records** tab. The tab has a Defects window and two push buttons in the window: **Create Defect** and **Select Defect**.

You can now enter a service request and create an associated defect in Rational Team Concert at the same time, without leaving the Service Requests application. Or you can associate the request with an existing resource in Rational Team Concert, by using the **Select Defect** button.

Creating a defect

In the Service Requests application, you create a service request, provide a description and other information, and save the request. You open the **Related Records** tab, and click the **Create Defect** button on the **Defects** tab.

Because you are creating a resource record in an external application, Rational Team Concert, you must sign in to Rational Team Concert. The Rational Team Concert login window is displayed. After you provide your user ID and password, the Create Resource window is displayed. The Create Resource window is a Rational Team Concert window, now available to you in the Service Requests application as a result of the OSLC integration.

You specify the type of resource to create, a defect. You also must provide information in the **Summary** field, and specify the project or container to file the defect against.

Your integration designer mapped the **Service Request Description** field to the Rational Team Concert Description property when the interaction was created. As a result, your service request description is in the **Description** field on the Create Resource window. You can edit the description if necessary.

Previewing provider application resource record data

Creation of the record in Rational Team Concert also creates a row in the Defects table in the Service Requests application. The **Description** field contains information about the linked resource in Rational Team Concert. The table row also shows the URL of the linked resource record.

The **Description** field has an information icon, a small blue circle with an “i” in it. You can hover the cursor over the information icon to display a window that shows data from the defect record in Rational Team Concert. (If you are not signed in to the provider application, hovering over the information icon displays a message and a link to sign in.*If the provider application does not support OSLC previews, a second icon appears and you can click on this icon to see a preview.*)

You can use the preview window to check the status and other information about the Rational Team Concert defect record from within the Service Requests application.

Selecting a defect

In the Service Requests application, you enter a second service request. The request is to fix an instance of a problem that was previously reported, and for which a defect exists in Rational Team Concert. For the second request, you want to link to the existing defect, rather than create a defect record. On the Defects table on the **Related Records** tab, you click **Select Defect**.

You might need to select the container, such as the Rational Team Concert project, for the defect, if the provider application supports multiple containers.

On the Select Resource window, you specify **Defect** as the type of resource that you want to select. Provide search information, such as the defect number or description, to display the defect or defects that match your search criteria. When you select a defect and click **OK**, the link to the Rational Team Concert defect resource is established. A row for the defect is added to the Defects table in the Service Requests application. You can use the information icon to display defect information, just as you can for the row that you added by creating a defect.

Public URI changes:

If an OSLC provider application is moved to another server or its public URI changes for another reason, you can change the public URI on the OSLC provider record. When you update the public URI, URIs that are based on the public URI also change, which prevents links from breaking.

In the OSLC Providers application, you use the Change Public URI action to specify the new public URI for an OSLC provider. After you change the public URI, URIs that are based on the public URI, such as service provider URIs and resource URIs, are also changed. Updating the links to match the new public URI maintains the links as active, viable links.

Migration of OSLC integrations:

You can migrate an OSLC provider definition and its related interactions from one instance of your product to another. For example, you might set up an OSLC integration in a development or test environment. After you establish that the integration is working properly, you can migrate it to a production environment.

Migration of OSLC integrations takes advantage of existing Migration Manager application features and is largely done in the Migration Manager application. The OSLC Providers application has an action to generate the package that is to be migrated.

Before you run the Migrate OSLC Provider action, you do preparatory work in the Migration Manager application. An OSLCPROVIDER package definition is provided in the Migration Manager application. For the OSLCPROVIDER package definition, you specify a target database or file destination for the package that you generate in the OSLC Providers application.

You use the Migrate OSLC Provider action in the OSLC Providers application to generate the package that you then manage in the Migration Manager application. The package contains the OSLC provider record, the associated interactions, and the associated user interface changes in the integrated consumer application or applications.

After you run the Migrate OSLC Provider action, a system message is displayed. The final line of the log message shows the name of the package. The package is listed on the **Package** tab of the OSLCPROVIDER package definition in the Migration Manager application in the source environment.

In the target environment, you use the Migration Manager application to import and deploy the package.

If you use a custom Java mapping file in your OSLC integration, the custom Java mapping file does not get included in the package. You must manually move the custom mapping file and the corresponding class file to the target environment, or manually add them to the package.

The Migrate OSLC Provider action generates a package for the current OSLC provider record only. If you have multiple OSLC provider records, you must migrate each one individually.

The Migration Manager does not support the migration of interaction groups. To migrate the objects in an interaction group, you can build a new object structure that uses the following objects:

```
OSLCINTGROUP
 OSLCGRPAPPS
  OSLCGRPAPPMENU
 OSLCGRPMEMBERS
```

You must include the new object structure in the OSLCPROVIDER migration package.

Manual UI modification:

Designing OSLC interactions between applications requires multiple user interface (UI) changes in the consumer application. Several scenarios might lead you to use the Application Designer application to make or refine UI changes as part of the integration.

Manual implementation of UI changes and security for interactions:

If you choose to not have the wizard auto-generate user interface (UI) changes in the consumer application, you must modify the UI yourself. The user interface must make the interaction available to users through a push button. If you do not select security groups in the wizard, you must specify the security groups that have access to the interaction.

Even when you do not auto-generate user interface changes, creation of an OSLC interaction creates a link relationship and a push-button signature option. The link relationship name is based on the specified resource type or usage. The signature option name is based on the interaction name. The relationship name and the signature option name are listed in the interaction details on the OSLC Provider tab. You need these two values when you manually implement UI changes.

Use the Application Designer application to modify the user interface of the consumer application. You must create a push button so that the OSLC interaction can be executed to create a link. And you must provide a tab with a table to hold the links to resources in the provider application. Typically, you provide the button and the link table together on the same tab.

Use the Security Groups application to specify the groups that have access to the push button. Only users in the groups that you authorize can see the button and use it to display the provider application window and run the interaction.

User interface refinements:

You might want to make refinements or additions to the consumer application user interface (UI) even when the UI changes are auto-generated.

After you design an OSLC interaction that auto-generates the user interface changes, your review of the application UI might reveal that revisions are warranted. For example, you might want to edit the labels on the new tab, table window, or push button. Or you might want to adjust the positioning of UI elements. Use the Application Designer application to modify or add UI elements.

During interaction design, you can specify that all service providers are to be available for the interaction in the consumer application. In these cases, a Select Container window is displayed when the interaction is initiated. The window is used to specify which service provider or container to use for the interaction. You might want to provide help grid text for the Select Container window. For example, you might add help grid text to indicate that the choices represent projects, or databases, or some other resource container. You also might relabel the window title to add clarity to what the user is to select.

Translation of untranslated UI elements:

You use the Application Designer application to modify and translate some user interface (UI) elements, such as tooltips for icons, that are added by the interaction design.

Some user interface elements that are added by the design of an OSLC interaction do not appear in the language appropriate for the interface, because interactions must be created in the base language. The elements are added after the original UI elements were translated, before deployment of the application. To translate the UI elements into the language of the application UI, use the Application Designer application to modify the untranslated text elements.

OSLC properties:

You can use system properties to help manage Open Services for Lifecycle Collaboration (OSLC) application integration.

Table 41. OSLC properties

Property	Description	Default value
mxe.oslc.collectioncount	Represents the total count of the OSLC collection.	0
mxe.oslc.defaultep	Represents the default OSLC endpoint.	OSLCDEFAULT
mxe.oslc.defaultformat	Represents the default format for OSLC.	oslcjson
mxe.oslc.dfltconsumerversion	Represents the default OSLC version that the consumer uses.	2.0
mxe.oslc.dfltversion	Represents the default OSLC version for an OSLC provider.	2.0
mxe.oslc.enableprovider	Enables the OSLC provider.	1
mxe.oslc.idleexpiry	Represents the time in seconds that the system can be idle before the sessions expires.	300
mxe.oslc.prcreateep	Represents the Provider Registry Create Endpoint.	
mxe.oslc.preferproviderdesc	Specifies whether the OSLC Provider description is preferred for resource registry reconciled URIs.	true
mxe.oslc.prettyjson	Represents the pretty print JSON.	1
mxe.oslc.prettyrdf	Represents the pretty print RDF.	1
mxe.oslc.prqueryep	Represents the Provider Registry Query Endpoint.	PROVIDERREGISTRY
mxe.oslc.webappurl	Represents the Provider Public URI.	http://localhost/maximo/oslc/

Integration queries

The integration framework supports queries from external systems. The external system sends an XML message to query the integration framework and the integration framework returns an XML message as a response to the query. You can execute a query for object structure and enterprise services by using HTTP, Java™ Remote Method Invocation (RMI), or a Simple Object Access Protocol (SOAP) request in a web service.

Support for XML queries is based on the system Query By Example (QBE) capability that is available from the **List** tab of most applications. XML-based queries provide the same query support that is provided in the applications except for attribute searches that are available in some system applications. Integration query time outs are controlled by the `mxe.db.Query.Timeout` system property.

Related concepts:

"REST API" on page 227
The Representational State Transfer (REST) application programming interface (API) provides a way for external applications to query and update application data in Tivoli's process automation engine.

Query services

Object structure services and enterprise services support query operations. An external source can use a service to run a query and to retrieve data from a system. In both cases, the object structure schema defines the XML content for the query request and the query response.

For an object structure that has more than two levels of business objects, the query framework supports the use of business object attributes in the top two levels of the request XML. The query response XML contains all the objects in the object structure.

All system-provided object structures with a **Consumed By** value of INTEGRATION, support the query operation by default. You can configure an object structure to support the query operation only and no other operations that support updating objects. You can use the Object Structures application to create additional object structures which provide support for the query operation.

Creating an enterprise service query

An external source can use an enterprise service to run a query and to retrieve data from a system.

About this task

A sample enterprise service query, `MXINVBALQInterface`, is provided. You can use this sample as a reference when you create query enterprise services.

Procedure

1. If one does not exist, create an object structure containing the objects that the query needs to access.
2. Define an enterprise service that implements the object structure you intend to use for the query.
3. Specify Query as the operation on the enterprise service.

4. Associate the enterprise service with an external system, and enable the external system and its enterprise service.

Web service queries

Query services that are created in the Enterprise Services application and the Object Structures applications can be deployed as web services. To support querying, you must configure enterprise web services to bypass the JMS queues.

A successful response to a query that is run in a web service returns the query result set. If the result set is empty (it contains no records), the XML that is returned in the SOAP body contains the following empty MXPERSONSet tag:

```
<max:QueryMXPERSONResponse xmlns:max="http://www.ibm.com/maximo"
  creationDateTime="2011-04-28T21:49:45"
  baseLanguage="EN" transLanguage="EN" messageID="12345"
  maximoVersion="7.5" rsStart="1" rsCount="10" rsTotal="10">
  </max:MXPERSONSet>
</max:QueryMXPERSONResponse>
```

If an error occurs, an HTTP response code of 500 is returned, along with a SOAP fault detailing the error message.

Use the following URL for the query web service:

`http://`*`hostname:port`*`/meaweb/services/`*`web service name`*

- The `host:port/meaweb` is the value of the integration web application URL property.
- The web service name is the name of the web service.

You deploy query enterprise and object structure services as web services using the Web Services Library application.

Query XML structure

The name of the root element of a query is the concatenation of the operation (Query) and the name of the associated object structure. For example, QueryMXPERSON, where MXPERSON is the object structure.

Root element

The name of the root element of a query is the concatenation of the operation and the name of the object structure, for example, QueryMXPERSON, where MXPERSON is the object structure. The following table lists the attributes that can apply specifically to the root element of a query, a response to a query, or both. All attributes in this table are optional.

Attribute	Description	Type	Applicable to
uniqueResult	Specifies whether the query expects one record or multiple records in a response. Value 0 (default): The query can return multiple records. Value 1: The query can return a single record; otherwise, an error occurs.	Boolean	Query

Attribute	Description	Type	Applicable to
maxItems	If the query can return multiple records, this attribute limits the number of records to be returned at one time. If this attribute is not specified, the response contains the entire result set.	PositiveInteger	Query
rsStart	In the query request: Use with maxItems to specify the first record to be returned in a response. If maxItems equals 10 and rsStart is not specified, the response returns results 0 through 9. To receive results 10 through 19, resend the query with rsStart equals 10. If rsStart is not specified, the response starts with the first record in the result set. If the number of records in the query result set is lower than the value of rsStart, the response returns no records.	Integer	Query
rsStart	In the query response: This value matches the rsStart value in the corresponding request. If the corresponding request contains a maxItems value, the rsStart value in requests for additional records is rsStart + rsCount + 1. If this attribute is not specified, the response starts with the first record in the result set and includes the number of records specified by the rsCount attribute.	Integer	Response
rsCount	The number of records that are returned in the message. rsCount does not reflect any rows that may be skipped as part of the response processing layer.	Integer	Response
rsTotal	The total number of records in the result set. If the query does not specify a maxItems value, the rsTotal value is the same as the rsCount value.	Integer	Responses (output)

For example, the response to this query request returns records 11 through 20 of the query result set by virtue of setting the rstart value to 11 and the maxItems value to 10.

```
<max:QueryMXINVBAL xmlns:max="http://www.ibm.com/maximo"
  creationDateTime="2011-049-28T21:49:45"
  baseLanguage="EN" transLanguage="EN" messageID="12345"
  maximoVersion="7.5" uniqueResult="0" maxItems="10" rsStart="11">
  <max:MXINVBALQuery orderby="string" operandMode="OR">
```

The following query result set contains a total of 35 rows, as noted by rsTotal, but only rows 11 through 20 are returned.

```
<max:QueryMXINVBALResponse xmlns:max="http://www.ibm.com/maximo"
  creationDateTime="2011-04-28T21:49:45"
  baseLanguage="EN" transLanguage="EN" messageID="12345"
  maximoVersion="7.5" rsStart="11" rsCount="10" rsTotal="35">
  <max:MXINVBALSet>
    .
    .
    .
  </max:MXINVBALSet>
```

Query operator

The QueryOperatorType data type supports the use of different operators. Your query request XML can use the different operators to filter the data that is returned to the querying external source.

QueryMXPERSON element

The QueryMXPERSON element has the following types:

- The QueryMXPERSON element is type QueryMXPERSONType.
- QueryMXPERSONType has element MXPERSONQuery, which is type MXPERSONQueryType.
- MXPERSONQueryType has elements for all the configured attributes of the PERSON object and all of its child objects (EMAIL, PHONE, and SMS).

The QueryMXPERSON operation element uses the following attributes:

- uniqueResults – Is a Boolean value that when set to 1 (True), directs the query to return a single and unique record when the value is set to 1 (True). If more than one record is found, an error is returned. When the attribute is not provided, the default value is 0 (false).
- maxItems – When this value is set to 10 on the query, it limits the number of records that are returned in the query to 10, even when the result set of the query may be greater than 10. When the attribute is not defined, all rows in the result set of the query are returned.
- rsStart – When this value is set to 11 on the query, all the records in the result set are returned starting with record 11. The query result skips records 1 - 10. When the attribute is not defined, the records are returned starting with record 1 in the result set.

The MXPERSONQuery content element uses the following additional attributes:

- orderby – Using this value is equivalent to using an Order By in a SQL statement. The attribute can contain a list of comma-separated field names that also include ASC and DESC options. When the attribute is not defined, the query returns records in the order that it was retrieved by the database.
- operandMode – This value has two valid values, AND and OR. The system uses this value when one or more fields is used for evaluation in a query execution. When you use the AND value, all field evaluations must be true. When you use

the OR value, only one of the field evaluations must be true. When the attribute is not defined, the default value is AND.

QueryMXPERSONResponse element

The QueryMXPERSONResponse element has the following types:

- Element QueryMXPERSONResponse is type QueryMXPERSONResponseType.
- The QueryMXPERSONResponseType has element MXPERSONSet, which is type MXPERSONSetType.
- MXPERSONSetType has elements for all the configured attributes of the PERSON object and elements for child objects defined in the object structure (PHONE, EMAIL, XYZ, and SMS).

The QueryMXPERSONResponse element has the following attributes:

- rsStart – This value contains the value set on the query request. If the value is not defined on the request, the default response value is 1.
- rsCount – This value contains the number of records that are returned in the query response.
- total – This value contains the number of records in the final query result set. The total value can be more than the number of records that are returned when the maxItems attribute is used in the query request.

Query selection criteria

The object structure element of a query request contains the selection criteria for the query. A query can select records based on a single value, a range of values, or a provided 'where' clause. The integration framework supports the use of query operators such as = or >.

Selection criteria apply only to attributes of objects in the top two levels of the object structure; that is, the primary object and its immediate child objects. However, the response includes data from all the objects in the object structure.

Field selection

A field-based query compares the value in a database field with the value in the XML field of the query request.

The following sample query searches for employees:

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The following sample query searches for employees where the PERSONID is equal to ATI and STATUS is equal to ACTIVE.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
<PERSONID operator ="=">ATI</PERSONID>
<STATUS operator ="=">ACTIVE</STATUS>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The operandMode attribute of the MXPERSONQuery element defines the statement that is run with an AND or an OR condition between the field evaluations. The default condition that is used by the system is the AND condition. Additionally, the operandMode attribute can be provided at a field level. In this case, a field element can have multiple occurrences and can be evaluated for different conditions, such as a quantity field being evaluated as less than 2 or greater than 10.

The following sample query searches for employees where PERSONID is like %*ATI*%. The operand represents the default behavior and requires no operator value.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
<PERSONID>ATI</PERSONID>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The following sample query searches for inventory balances where the bin number is not null.

```
<QueryMXINVBAL>
  <MXINVBALQuery>
<INVBALANCES>
<BINNUM operator ="!="></BINNUM>
</INVBALANCES >
</MXINVBALQuery>
</QueryMXINVBAL>
```

The following sample query searches for the inventory balances where the bin number is null.

```
<MXINVBAL>
<INVBALANCES>
<BINNUM>NULL</BINNUM>
</INVBALANCES >
</MXINVBAL>
```

The following sample query uses the equivalent of a SQL IN clause to search for the employees whose status is ACTIVE or INACTIVE.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
<STATUS>ACTIVE,INACTIVE</STATUS>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The following sample query searches for employees where the PERSONID starts with the letter A.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
<PERSONID operator ="SW">A</PERSONID>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The following sample query searches for employees where the PERSONID ends with the letter Z.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
<PERSONID operator ="EW">Z</PERSONID>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

Field evaluation

The operator attribute compares the value of a database field with one or more values and has the format `operator = value`.

The value attribute can have the following values.

Value	Description
=	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
EW	Ends With
SW	Starts With

Use the less than and greater than attributes with numeric values and date fields only.

For example, to find all employees that have an ACTIVE status, format the query as follows:

```
<QueryMXPERSON>
<MXPERSONQuery>
<PERSON>
<STATUS operator ="=">ACTIVE</STATUS>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

Range selection

A query can search for records with a value that falls within a range of values. The format depends on whether the selection criterion is open-ended or it contains an upper and lower range.

The following sample query searches for purchase orders where the DEPARTMENT is greater than 1000.

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
< DEPARTMENT operator="&gt;=">1000</DEPARTMENT>
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

The following sample query searches for person records where the PERSONID is greater than 1000 and less than 20000. The query uses two instances of a single field element, the first with the From selection criteria, and the second with the To selection criteria:

```
<QueryMXPERSON>
 <MXPERSONQuery>
<PERSON>
< PERSONID operator="&gt;=">1000</PERSONID >
< PERSONID operator="&lt;=">20000</PERSONID >
</PERSON>
</MXPERSONQuery>
</QueryMXPERSON>
```

Where clause selection

A query can search for records based on a SQL 'Where' clause that can be provided as part of the integration query XML. The Where clause offers support for more complex queries and the querying of classification-related data.

The use of the 'Where' clause is mutually exclusive from the use of individual elements. If a query request provides data for the Where element and other elements in the XML, only the WHERE element is used in the query execution.

The WHERE element is located on the same level as the top-level object in an object structure (purchase order), as in the following example:

```
<QueryMXPO xmlns="http://www.ibm.com/maximo">
   <MXPOQuery>
    <PO> </PO>
     <WHERE> </WHERE>

   </MXPOQUERY>
</QueryMXPO>
```

The WHERE element is available in the schema only when the operation is Query and can exist only once. To join multiple tables in the query, use ‘exists' within the Where clause, in the same manner as the WHERE clause from the List tab of an application. The following sample query, using the MXPO object structure, retrieves purchase orders that have a PO Line for Item number 1002:

```
<QueryMXPO xmlns="http://www.ibm.com/maximo">
 <MXPOQuery>
  <WHERE>(siteid = 'BEDFORD') and (exists (select 1 from maximo.poline where
  (i temnum='1002') and (ponum=po.ponum and revisionnum=po.revisionnum and
  siteid=po.siteid)))</WHERE>
 </MXPOQUERY>
</QueryMXPO>
```

The result of this query retrieved two purchase orders (not all elements are included in the example):

```
<?xml version="1.0" encoding="UTF-8"?>
<QueryMXPOResponse xmlns="http://www.ibm.com/maximo" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" creationDateTime="2012-03-21T11:20:13-04:00"
transLanguage="EN" baseLanguage="EN" messageID="1332343214013260273" maximoVersion=
"7 5 20110413-2230 V7500-721" rsStart="0" rsTotal="2" rsCount="2">
  <MXPOSet>
    <PO>
      <DESCRIPTION>Window and installation for office building</DESCRIPTION>
      <ORDERDATE>2000-04-20T14:00:00-04:00</ORDERDATE>
      <ORGID>EAGLENA</ORGID>
      <PONUM>1013</PONUM>
      <POTYPE>STD</POTYPE>
      <VENDOR>JK</VENDOR>
 .
 .
 .
      <POLINE>
        <DESCRIPTION>5 ft. X 6 ft. window pane</DESCRIPTION>
        <ITEMNUM>1002</ITEMNUM>
```

```
      <ITEMSETID>SET1</ITEMSETID>
      <ORDERQTY>1.0</ORDERQTY>
      <POLINENUM>1</POLINENUM>
.
.
.
      <POCOST>
        <COSTLINENUM>1</COSTLINENUM>
.
.
.
      </POCOST>
    </POLINE>
    <POLINE>
      <DESCRIPTION>Installation of window pane</DESCRIPTION>
      <ITEMNUM />
      <ITEMSETID>SET1</ITEMSETID>
      <ORDERQTY>6.0</ORDERQTY>
      <POLINENUM>2</POLINENUM>
.
.
.
      <POCOST>
        <COSTLINENUM>1</COSTLINENUM>
.
.
.
      </POCOST>
    </POLINE>
  </PO>
  <PO>
    <DESCRIPTION>Window and installation for Office Building</DESCRIPTION>
    <ORDERDATE>2003-02-27T10:07:24-05:00</ORDERDATE>
    <ORGID>EAGLENA</ORGID>
    <PONUM>1029</PONUM>
    <POTYPE>STD</POTYPE>
    <VENDOR>JK</VENDOR>
.
.
.
    <POLINE>
      <DESCRIPTION>5 ft. X 6 ft. window pane</DESCRIPTION>
      <ITEMNUM>1002</ITEMNUM>
      <ITEMSETID>SET1</ITEMSETID>
      <ORDERQTY>1.0</ORDERQTY>
      <POLINENUM>1</POLINENUM>
.
.
.
      <POCOST>
        <COSTLINENUM>1</COSTLINENUM>
.
.
.
      </POCOST>
    </POLINE>
    <POLINE>
      <DESCRIPTION>Installation of window pane</DESCRIPTION>
      <ITEMNUM />
      <ITEMSETID>SET1</ITEMSETID>
      <ORDERQTY>6.0</ORDERQTY>
      <POLINENUM>2</POLINENUM>
.
.
.
      <POCOST>
        <COSTLINENUM>1</COSTLINENUM>
```

```
.
.
.
        </POCOST>
      </POLINE>
    </PO>
  </MXPOSet>
</QueryMXPOResponse>
```

Interface tables

Interface tables are an option for integration with systems that use database tables to exchange data. This integration option applies only to enterprise services and publish channels and is always processed asynchronously, by using JMS queues.

Within an external system, there can be one or more publish channels and enterprise services that are being used for integration with interface tables. Any channel or service that is using an interface table must be associated with an object structure that is configured to support flat files and all alias conflicts must be resolved.

Location of interface tables

The endpoint definition for an external system or a publish channel points to the database where its interface tables are stored. The database can be the local application database or a remote database. The predefined content includes the `MXIFACETABLE` interface table endpoint which points to the application database. You can add additional endpoints for remote databases.

Names of interface tables

The integration framework registers interface table names to an enterprise service or a publish channel. Default names for interface tables are not provided. Apply the following guidelines when naming interface tables:

- Publish channels and enterprise services that use the same object structure can use the same interface table name or different interface table names.
- Publish channels and enterprise services that use a different object structure must use different interface table names.

Interface queue tables

The interface queue tables identify the sequence in which a receiving system processes the records in the respective interface tables. Two queue tables exist, one for inbound transactions and the other for outbound transactions. Some transactions depend on the successful processing of a previous transaction, for example you must create a user before you can add that user to a security group. The receiving system must process the records in the same sequence in which the sending system created the records.

Table 42. Interface queue table

Interface queue table	Direction
`MXOUT_INTER_TRANS`	Outbound
`MXIN_INTER_TRANS`	Inbound

External applications that pull data for outbound messages can use the outbound interface queue table (mxout_inter_trans). However, the external applications can also choose to use other methods of consuming outbound messages that meet their integration requirements.

All inbound and outbound transactions must have a record that is inserted into the corresponding inbound or outbound queue table. This record contains a TRANSID value, a unique identifier that identifies the interface table to which the transaction data is written. The corresponding interface table uses the TRANSID value to identify the record or records that are associated with the transaction. You can identify the contents of a transaction by looking up all the records with a given TRANSID value in the corresponding interface table.

The sequence of TRANSID identifies the sequence in which records are processed by the integration framework. For example, when users and security groups are entered into the system, the TRANSID values for the user record must be lower than the TRANSID values for the security group records that reference that user.

The difference between the MXIN_INTER_TRANS and MXOUT_INTER_TRANS queue tables is the direction of the interface table records that they track. The external system must write to the MXIN_INTER_TRANS queue table, and the integration framework must read from it. The integration framework writes to the MXOUT_INTER_TRANS queue table, and the external system reads from it.

The external system can use the MXOUT_INTER_TRANS table or retrieve outbound records from interface tables. The interface queue tables are generated the first time that you create interface tables for an endpoint. Each endpoint has its own interface queue tables and a counter for maintaining the outbound TRANSID value.

Creation of interface tables

When an enterprise service and a publish channel use the same interface table, the Create Interface Tables window displays a list of interface tables based on the uniqueness of the interface table name and its corresponding endpoint.

You can create interface tables for enterprise services and publish channels when the associated object structures are marked as flat supported. The **Support Flat Structure** check box must be selected on the object structure. All alias conflicts must also be resolved for the object structure before you can create an interface table.

You can create interface tables for data synchronization on enterprise services and on publish channels. Interface tables do not support Query and Invoke operations. You can create interface tables for a specific endpoint. You must identify where the tables are created.

The database location that is referenced by the endpoint can be a local database or a remote database. When you create interface tables on a local database, the columns are registered in the system data dictionary. Local interface tables that use a database table and a database column show all updates (except insertions and deletions) to a base column attribute (such as data type) when you run the database configuration operation. When columns are added to or deleted from the base table, you must regenerate the corresponding enterprise service and the

publish channel interface tables to apply the column changes. No changes are applied in the remote databases. You must regenerate remote interface tables to apply the column changes.

Regeneration of interface tables

When columns are added or deleted from the system database tables, you must regenerate all local and remote interface tables that are associated with those object structures.

You regenerate interface tables by using the **Create Interface Tables** action in the External Systems application. If you select the **Rename Existing** check box, the application backs up existing data in the corresponding interface table to the INTERFACETABLENAME_BAK table.

If necessary, you can restore the data to the new table. Depending on the configuration of a multitenancy environment, you may need to request the assistance of the system provider to access the data that you want to restore. If you do not back up the table, the table is dropped and the data is lost when you regenerate the table. You cannot regenerate an interface table when the MXIN_INTER_TRANS queue table contains a record that points to that interface table. When a row exists in that queue table, the corresponding inbound transaction is ready to process, or the inbound transaction is in error.

The interface table creation process does not check for records in the MXOUT_INTER_TRANS queue table.

You can set the mxe.int.usedbinfoforifacetb property so that the processing of interface tables does not require an exact match between the object structure and the interface table. When the property is set to 1 (true), the interface process refers to the existing table to determine what to insert into the table rather than the object structure definition. This process avoids invalid column name errors when the endpoint is writing to the interface table.

If you add a new attribute to an interface table, the table must be regenerated to align with the object structure definition.

A utility is available on the Maximo Asset Management Content Library that can check an interface table and its related object structure to identify when they are not synchronized. This utility can also update the object structure to exclude columns when those columns do not exist in the interface. See this link for more information: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM%20Maximo%20Asset%20Management/page/Maximo%207.5%20Interface%20Table%20Utility.

Deletion of interface tables and records

When related inbound transaction records are successfully processed in an interface table, the corresponding record is deleted from the MXIN_INTER_TRANS queue table. The deletion indicates that the transaction was delivered successfully to the inbound JMS queue.

Records are deleted from the MXIN_INTER_TRANS queue table and not from the individual interface tables. The system administrator determines when and how to delete records from the interface tables.

For outbound transactions, the external system must manage the deletion and archiving of data in the queue table and interface tables. You cannot delete interface tables in the user interface or by deleting the corresponding object structure. An administrator must manage the archiving of data in the interface tables and can drop the table, if necessary.

Format of interface tables

An interface table has the same format as its corresponding object structure, including persistent and nonpersistent columns and excluding any columns that are excluded from the object structure.

The interface table includes additional columns that identify the sequence in which the sending system writes, and the receiving system processes the records in the various interface tables.

Key columns

If the interface table represents a hierarchical object structure with parent-child object relationships, the table does not include any part of the child object key columns that are included in the parent object key columns. For example, PERSONID is a key column in the PERSON, PHONE, EMAIL, and SMS records. The PERSONID column appears only at the parent (PERSON) level in the MXPERSON_IFACE interface table.

Duplicate columns and aliases

The XML representation of a hierarchical object structure contains duplicate column names, but interface table and flat file representation do not. If an object structure has duplicate non-key column names in both a parent object and a child object, a duplicate column name error occurs when the interface table or a flat file record is generated.

To resolve this issue, change the alias name for one of the duplicate column names. Every system database column can have an alias alternate name. When an alias exists, the system uses the alias when interface tables and flat files are generated. Change the alias to eliminate the duplicate column name error.

Most columns do not have an alias, but some columns have aliases that support the predefined enterprise services or publish channels.

A column within an object structure can have a single alias. If the object structure is used by multiple publish channels and enterprise services, a change to an alias affects every interface table that is associated with the object structure.

The columns in the predefined object structures have system-assigned aliases. Check for duplicates when you create a hierarchical object structure or when you add an object to a predefined object structure. The Add/Modify Alias window shows the fields and aliases for the objects in a selected object structure, and identifies any duplicate alias names with a check in the duplicate column. If a duplicate alias exists, overwrite its value in the **ALIASNAME** column.

Restricted columns

The HASLD field is an internal system column that is excluded from all object structures. Do not include this column in any object structure that is associated

with an interface table. The LANGCODE field is also excluded from the predefined object structures.

Integration processing columns

The following table shows the columns that are used in the interface table sequencing, retrieval, and processing. Some columns are in either the interface queue tables or the interface tables; some are in both places.

Table 43. Integration processing columns

Column name	Interface queue table	Interface table	Description
IFACENAME	Yes	No	The IFACENAME column contains the name of the enterprise service or the publish channel that is used in a transaction. The column is populated in outbound transactions. For inbound transactions, the external system must populate the column with the name of the enterprise service or the publish channel that corresponds to the row that is inserted into an interface table.
TRANSID	Yes	Yes	The TRANSID column in an interface queue table is a sequential number that uniquely identifies an integration transaction. The TRANSID and the interface table name, identifies a unique transaction. The interface queue table can contain one record with a TRANSID value. The corresponding interface table can have one or more records with the TRANSID, depending on the number of records that are written to that interface table as part of that enterprise service or the publish channel. If a transaction writes to multiple interface tables, the interface queue table contains a separate record with a unique TRANSID value for each interface table. Each interface queue table maintains its own TRANSID counter. The outbound TRANSID value is initialized when the interface queue table records are generated. You must create and maintain the TRANSID counters that populate the inbound queue tables and the interface table records. If the external systems do not correctly manage the inbound TRANSID counters, sequential processing is not guaranteed. Ensure that the TRANSID values that the external system generates does not duplicate the TRANSID value that is generated. Errors occur if duplicate TRANSID values exist and when you process the same object structure in both an inbound and an outbound direction by using a single interface table. Each endpoint has its own set of interface queue tables and its own outbound TRANSID counter.
TRANSSEQ	No	Yes	When multiple records in an interface table share the same TRANSID value, the TRANSSEQ column provides a secondary sequence number that indicates the sequence in which those records should be processed.
EXTSYSNAME	Yes	No	The EXTSYSNAME column in the interface queue tables can contain inbound or outbound data. For inbound transactions, the column contains the name of a valid and enabled external system that is defined in the integration framework. For outbound transactions, the column contains the name of the external system that is the destination of the transaction.

Table 43. Integration processing columns (continued)

Column name	Interface queue table	Interface table	Description
ACTION	Yes	No	The ACTION column contains the action to perform on the external system (outbound) or on the integration framework (inbound). The following actions can be used: • Add: Inserts the data that is provided in the message. • Delete: Deletes the parent data, and any related child data, from the database. • Change: Updates parent and child data contents of the message, but does not delete existing child data that is not explicitly specified in the message. • Replace: Replaces the existing records with the contents of the message and deletes existing child data that is not referenced in the message. • AddChange: If the primary record does not exist, processes as an add action and, otherwise, processes as a change action. • Null: If the primary record does not exist, processes as an add action and, otherwise, process as a replace action.
IMPORTMESSAGE	Yes (used inbound only)	No	The IMPORTMESSAGE column holds any error message that was produced when the interface table row was moved to the inbound queue.
TRANSLANGUAGE	Yes	No	The TRANSLANGUAGE column identifies the language of the transaction. For an outbound transaction, this value indicates the language of the user who initiated the transaction. For an inbound transaction, this value indicates the language of the transaction. Any attributes that support a multilanguage environment are expected to be in the language that the TRANSLANGUAGE value defines.
MESSAGEID	Yes (used outbound only)	No	The MESSAGEID column is a unique identifier that the system assigns to every outbound transaction.
IFACETBNAME	Yes (used outbound only)	No	The IFACETBNAME column is the name of the interface table that corresponds to the IFACENAME column. This column applies to outbound transactions only.

Long description columns in an Oracle Database

Long description columns are stored in a CLOB (character large object) column in an Oracle Database. Interface tables contain two versions of each CLOB column, one with data type CLOB, and one with data type ALN with a character length of 4000. In the following example, the name of the CLOB column is the column alias. The name of the alphanumeric column is the column alias with the suffix *2*.

Table 44. Long description columns in an Oracle Database

Data type	Name of description column
CLOB	PERSON_DESCRIPTION_LD
ALN	PERSON_DESCRIPTION_LD2

The system populates both columns in outbound transactions. For inbound transactions, the integration framework uses the value in the ALN column if it is

not null; otherwise, the value in the CLOB column is used.

Interface table polling

A predefined cron task polls the interface queue table and uses the IFACENAME, EXTSYSNAME, and TRANSID values to place the corresponding records into the appropriate inbound JMS queue for processing. The interface polling process checks that the names of the external system and enterprise service are valid and enabled.

You must configure the cron task to initiate interface table polling. You can also perform additional tasks to enhance performance during interface polling.

Interface table polling cron task

The IFACETABLECONSUMER cron task is a predefined cron task that polls the inbound interface queue table for new records to process.

The interface table polling process checks that the external system and enterprise service names are valid and currently enabled. If they are not, the record is marked in error and remains in the interface table.

If you disable interface table polling, new records remain in the interface tables. The messages that were sent to the inbound JMS queue are processed.

You must set up a mechanism to retrieve outbound transactions from the interface tables. You can use a polling program, as the system does for inbound transactions, triggers, or any other mechanism.

The cron task has the following parameters that you can configure. All parameters are optional.

Table 45. Interface table polling cron task parameters

Parameter	Description
EXITCLASS	Java exit class that enables the manipulation of data before it is written to an inbound queue.
ENDPOINT	Endpoint that is associated with the interface table. The default value is the predefined endpoint value that points to the local database.
ENTERPRISESERVICE	Enterprise service to be polled. The default (null value) is all enterprise services. If you specify a value for this parameter, you also must specify a value for the EXTSYSNAME parameter. The values limit the polling thread to a specific enterprise service instead of the default behavior, which polls for all enterprise services.
TARGETENABLED	Ensure that the value is at the default of 0 (false). The functionality of this flag is superseded by the donotrun functionality. Use the donotrun parameter in the cron task framework to control which servers the cron task runs on.
EXTSYSNAME	External system to be polled.

Table 45. Interface table polling cron task parameters (continued)

Parameter	Description
QUEUETABLE	Enterprise service queue table. The default value is MXIN_INTER_TRANS.

Advanced interface table polling

You can perform advanced configuration of the interface table polling process to improve its performance when reading data from interface tables.

If you send inbound messages through the continuous JMS queue and do not require messages to be maintained in first-in-first-out sequence, you can improve the performance of interface table polling.

Multiple cron tasks

The interface table polling process uses instances of the IFACETABLECONSUMER cron task to read all messages from all interface tables for all external systems that write to the tables. To improve performance, you can introduce multithreaded polling by configuring multiple instances of the IFACETABLECONSUMER cron task in the Cron Task Setup application.

For each instance, you define selectors by assigning values to the EXTSYSNAME and ENTERPRISESERVICE parameters, for example, EXTSYSNAME=EXTSYSNAME1 and ENTERPRISESERVER=MXPERSONInterface. All selectors must be mutually exclusive among the multiple instances to avoid processing a message more than once.

The performance benefit of multithreaded polling can be further realized in a clustered configuration. The IFACETABLECONSUMER instances can run on different servers to balance the load. To designate an instance of the IFACETABLECONSUMER cron task to run on a specific application server, set the TARGETENABLED parameter to 1 and, on the application server, set `-DIFACETBCONSUMER..<instance name>=1`.

If you configure multiple instances of a cron task the selectors must be mutually exclusive, so that messages are not processed multiple times. Selectors must retrieve all the enterprise service transactions that you use, so that no messages are left unprocessed.

Multiple queues

You can improve performance by setting up multiple interface queue tables. For example, you can write each interface to a separate queue table and define separate cron tasks to process the queue tables independently of one another. You also can set up separate queue tables for each external system and, within each queue table, define selectors for each interface. Depending upon the complexity of your integration, you can use multiple queue tables instead of multiple selectors. To set up multiple queue tables, create the queue tables in the same database as the interface tables, and include all the columns that are in the MXIN_INTER_TRANS queue table. You must design the external system to write to the appropriate queue tables. Ensure that the external system does not insert an interface table message into more than one queue table so that the message is not processed multiple times.

Processing interface tables on an external system

To enable an external system to use interface tables, you must create interface tables, define backup procedures for restoring interface tables, and configure the archiving of interface tables.

Enabling inbound processing

To configure inbound processing of interface table data from an external system, you must create a TRANSID counter and create records for the interface table and queue.

Before you begin

To use interface tables, you must create the tables and configure the IFACETABLECONSUMER cron task.

Procedure

1. Create and initialize the outbound TRANSID counter.
2. Create records for each interface table that an inbound transaction writes to, populating each record with the following information:
   a. The transaction data
   b. The incremented TRANSID value
   c. If multiple records exist for the same interface table, the incremented TRANSSEQ value
3. Create an MXIN_INTER_TRANS queue record with the following information:
   a. The same TRANSID value that is contained the interface table record
   b. The name of the enterprise service that corresponds to the interface table, in the IFACENAME column
   c. Optional: The ACTION value
   d. The identifier of the external system, in the EXTSYSNAME column
4. Perform a single commit, to commit all records for a transaction at one time.

Enabling outbound processing

To configure outbound processing , you must set up and configure a process, such as a polling program or a trigger, to retrieve transactions from the MXOUT_INTER_TRANS queue table.

Before you begin

To use interface tables, you must create the tables.

Procedure

1. Set up a process to retrieve interface table transactions by using the MXOUT_INTER_TRANS queue table. You can use a polling program, a trigger, or any other mechanism.
2. For the polling program to process transactions sequentially configure it to read the records in the MXOUT_INTER_TRANS queue table in the TRANSID sequence.
3. Enable each record in the MXOUT_INTER_TRANS queue table:
   a. Access the interface table that you just identified, and retrieve the first record in which the TRANSID value matches the TRANSID value in the current MXOUT_INTER_TRANS queue record. If the interface table

contains multiple records with the same TRANSID value, retrieve and process them in TRANSSEQ sequence.

b. Process data according to the value in the ACTION column of the interface queue table.

4. Commit all records for a single database transaction.
5. Delete the current record from the MXOUT_INTER_TRANS queue table.

What to do next

Implement error management, based on your external system requirements.

Integration modules

An integration module provides a mechanism for a process management product to invoke an external operational management product. You can configure an integration module to automate logical management operations, such as software deployment, by using an operational management product.

A process management product can invoke an integration module from an application, a workflow process, or an escalation process. The integration module then invokes an operational management product. The operational management product automates a service management process, such as a software deployment. The implementation returns the results to the process management product.

If predefined modules are provided with your operational management product, use the process solution installer to load them into the integration framework. If you are familiar with the programming interface for the operational management product, you can create integration modules when one does not exist.

Integration module components

An integration module is composed of defined integration framework components. You can configure the integration module to be implemented as a Java class or as an invocation channel.

When you install a predefined integration module, integration module artifacts with the Java class and XSL files are provided. An integration module also provides logical management operation definitions.

Integration module definitions

When you install an integration module, the options that you select are used to form the module definitions.

The integration module definition includes the following information:

- The name, version, and description of the integration.
- Operational management product and version.
- Handler that identifies the protocol that the integration module uses to invoke an operational management product.
- Integration module implementation choice, either a Java class or invocation channel.

The following components are contained in the integration module definition:

- The logical management operations that the integration module supports.

- The integration module properties that are unique to the integration module. Use the properties to configure the behavior of the integration module.

Operational management products

An operational management product automates system processes, such as software deployment. Operational management product services are run on assets, such as servers. The assets are referred to as configuration items.

Multiple instances of operational management products can exist in a production environment. More than one operational management product can manage a single configuration item. Each operational management product has its own unique identifier, a source token, for each configuration item.

The database includes a repository of operational management product and configuration item information. This information includes relationships between the operational management products and configuration items, and the source tokens that drive the integration between process management products and operational management products.

Logical management operations

Logical management operations identify the actions that integration modules support, such as Get Status and Deploy Software. A logical management operation consists of a data source, target, and specific object field attributes that the source and target use during a process invocation.

Logical management operations act as interfaces between the process management product and the integration module. You can design and develop integration modules and process management products independently. You can install logical management operation definitions with a product and you can also create logical management operations in the Logical Management Operations application.

The definition of a logical management operation contains the following properties:

Property	Description
Name	The name of the action, such as Get Status or Deploy Software.
Namespace	A unique qualifier, such as `com.ibm.tivoli.deployment`.

Property	Description
Invocation pattern	• Synchronous: The process management product issues a request and the integration module returns the results of the operation immediately. • Asynchronous one-way: The process management product issues a request, and no response is returned. • Asynchronous deferred response: The process management product issues a request, and a token identifies the instance of the request. The process management product passes the token as input to another logical management operation, which then obtains the status of the original request. • Asynchronous callback: The process management product issues a request, and a token identifies the instance of the request. The operational management product uses a token to perform a call-back to identify and to report the status of the original request. The call-back, which is provided by the integration module, inserts or updates a business object.
Source business object	The input object for the logical management operation.
Response business object	The output of the object for the logical management operation.
Business object attributes	The specific attributes of the objects that are needed either for input or output and their data types.

Input objects and input object attributes identify the data that the process management product passes to the integration module. If the logical management operation is configured with input attributes and does not contain an input object, the process management product can pass any business object that has the required input attributes. If an input object is specified, the process management product must pass the business object to the integration module. The response object and attributes identify the data that the integration module returns.

Implementation prerequisites

Before any integration module implementations occur, you must ensure the logical management operation is associated to one or more integration modules. The integration module then must have an association with an operational management product. Finally, the logical management operations on each operational management product must be enabled.

Logical management operation associations

Typically, there is a one-to-one communication between a logical management operation and an operational management product function. However, a single

logical management operation invocation can cause the integration module to invoke an operational management product multiple times.

An integration module that uses an invocation channel can run logical management operations only in the following circumstances:

- The source object that is defined on the logical management operation must match the main object of the request object structure that you define on the invocation channel.
- If the invocation channel is not configured to process a response, you cannot associate a response object with the logical management operation.
- The response object that you define on the logical management operation must match the main object of the response object structure that you define on the invocation channel. The match must occur when you configure the invocation channel to process a response.

Operational management product associations

The integration module can support multiple logical management operations for an operational management product. When you run the integration module, the name and namespace values of the logical management operation specify the action to perform.

Operational management product and integration module associations are created when the OMPPRODUCTNAME and OMPVERSION values are added to the integration module. If multiple versions of the operational management product are used with the integration module, use a null value for the OMPVERSION.

Logical operation management enablement

More than one integration module can implement the same logical management operation on an operational management product. Use the ISPRIMARY attribute to identify the preferred integration module for the logical management operation on the operational management product. Only one integration module can have the ISPRIMARY value set to a true for any given logical management operation - operational management product combination.

Implementation properties

The integration framework can automatically process integration data when you implement an integration module. You can customize the implementation if you want to bypass parts of the automatic process.

Integration module parameters

The integration framework provides integration module input parameters, including source and response objects and object sets, and endpoint names. These parameters are passed when you run an integration module as a Java class or as an invocation channel.

The following table lists the `psdi.iface.omp.IMConstants` interface parameters and their names.

Name	Description
IM	The instantiated ServiceInvoker object which can be a Java integration module or an invocation channel.

Name	Description
IMNAME	The name of the integration module.
IMVERSION	The version of the integration module.
LMONAME	The name of the logical management operation that is invoked.
LMONAMESPACE	The name space of the logical management operation that is invoked.
OMPGUID	The globally unique identifier (GUID) of the operational management product that the integration module uses.
ENDPOINTNAME	The name of the endpoint that the integration module uses to communicate with the operational management product.
ENDPOINTPROP	A map of *String.psdi.iface.mic.MaxEndPointPropInfo* properties that override the endpoint properties.
USERNAME	The user name that the integration framework uses to communicate with the endpoint.
PASSWORD	The password for USERNAME.

Integration module process flow

Java-based integration modules populate the logical management operation response object or response object set with the results of the operation. For invocation channels, you can specify a mapping from the operational management product response to the response object or set. The integration framework copies the data into the response object or set.

Integration modules must quickly return to their callers. If the external service is a long-running service, the integration module must create another thread. The new thread makes the call to the operational management product, while the original thread returns to the caller.

Integration modules perform the following processing tasks:

1. Retrieve input from the source object or source object set.
2. If applicable, retrieve the integration module-specific properties.
3. Perform all processing logic that the logical management operation requires before it calls the operational management product.
4. Call the operational management product.
5. If applicable, handle the operational management product response.
6. If applicable, populate the response object or response object set with the return data.
7. Return processing information back to the caller.

Endpoints

An integration module that contains an invocation channel must use an endpoint. A Java class integration module can use an endpoint or use a custom approach to external service invocations.

Endpoints contain URL properties and handler properties that specify the transport mechanism to use. An endpoint with a web service handler has different properties than an endpoint with a command-line handler. Integration module endpoints and endpoint properties can be overwritten at run time when the process management product passes endpoint properties as input to the integration module. The USERNAME and PASSWORD properties that are returned from the credential mapper can also override the endpoint properties.

Java integration modules can communicate directly with an operational management product or external integration module, without the use of an endpoint. If you do not use an endpoint, you eliminate the need for the Java integration module to convert source objects into the required XML format. When you exclude endpoint use, you can use any communication protocol that is supported by the operational management product. Integration modules are not required to use endpoints. You can configure the integration module to communicate with an operational management product by using integration module properties.

The integration framework provides predefined handlers that support communication protocols such as HTTP and web services. If a predefined handler is not available to support the operational management product service protocol, implement an external integration module. The external integration module acts as an interface to the operational management product service that can use one of the available handler protocols. An alternative solution is to write a custom handler that supports the operational management product service protocol.

Configure a different endpoint for each operational management product to establish individual protocols for each product. The handler that you define for the endpoint must match the handler that you configure for the integration module.

Integration modules based on invocation channels require an endpoint. The endpoint name is one of the parameters the integration framework passes as input to the invocation channel. The caller of the integration module can override the endpoint name.

Regardless of whether you use an endpoint handler to communicate with the operational management product, the integration module must handle scenarios that require multiple invocations of the operational management product for a single logical management operation execution.

Invocation channel or Java class implementation

You can implement an integration module either as a Java class or as an invocation channel. Before developing an integration module, review the advantages and disadvantages to invocation channel or Java class usage.

Invocation channel and Java class comparison

Before you develop an integration module, consider the advantages and disadvantages of using a Java class or an invocation channel.

Advantages and disadvantages of using an invocation channel

Implementing an integration module by using an invocation channel has the following advantages:

- An invocation channel is useful when you pass complex data to the endpoint. It is also useful when you can define a clear mapping between the fields in the source object and the input the endpoint requires.
- Can support hierarchical object structures on input and output transactions.
- The integration framework handles the conversion of objects to XML, and XML to objects.
- Can be configured to use processing classes, user exits, and XSL mapping for inbound and outbound transactions.
- The integration framework performs endpoint invocation.
- Can be invoked directly without an association to an integration module or a logical management operation.
- A simple integration can be implemented using system configuration that does not require you to restart the application server.

Implementing an integration module by using an invocation channel has the following disadvantages:

- Requires more registration and configuration of system artifacts, even when the invocation is simple.
- Multiple applications cannot use the integration module with different business objects.
- Requires additional knowledge of the integration framework.
- Supports only a single invocation of the configured endpoint. It requires additional coding to support multiple invocations.
- Requires additional coding to support a long-running service. By default, the invocation channel waits for a response from the endpoint.

Advantages and disadvantages of using a Java class

Implementing an integration module by using a Java class has the following advantages:

- Requires less registration and configuration of system artifacts.
- Can be used by different applications while using different business objects.
- Can be designed to make multiple external invocations.
- Can use different communication protocols for different logical management operations and operational management products.
- Supports the use of another thread to accommodate a long-running service.
- Can implement multiple logical management operations, even when the input configuration and the output configuration is different.
- Is less likely to require the use of an external integration module.

Implementing an integration module by using a Java class has the following disadvantages:

- Requires you to perform more Java coding.
- Does not support an object structure with a parent and child relationship as input or output. Only the parent object can be used as input, and the Java code must find the child relationship.
- Conversions of objects to XML and XML to objects must be coded when you use an endpoint handler.
- Customization must be built into the design of the integration module. Customization cannot be added later without redeploying the code.

- Use of any integration framework components, such as an endpoint handler, must be coded in the Java class.

Invocation channel implementation

Invocation channels use object structures in its implementations to expand the message data content. The object structure uses the object that the process management product passes as the source object to build an entire record that can consist of multiple related objects.

The objects that are identified in the object structures are used in the XML message creation. When you use an invocation channel, the data must conform to the XML schemas specified for the associated object structures, and you must define an endpoint for the invocation channel.

Invocation channels support optional inbound and outbound processing exit classes. The integration framework uses processing exit classes to implement additional logic. The inbound and outbound processing exit classes must be instances of the Java class `psdi.iface.migexits.ExternalExit`.

For outbound transactions, the integration framework calls the following method:

```
public StructureData setDataOut(StructureData irData)
```

For inbound transactions, the integration framework calls the following method:

```
public StructureData setDataIn(StructureData erData)
```

When you override these methods, you can perform additional integration module processing. The properties that are passed to the invocation channel are available to the processing classes.

The following outbound processing class code shows you how to retrieve the operational management product globally unique identifier when you run an invocation channel:

```
import psdi.server.MXServer;
import psdi.iface.omp.IMConstants;
import psdi.iface.omp.OmpServiceRemote;
import psdi.iface.mic.*;
import psdi.iface.migexits.*;
.
.
.
public class OutboundCIExit extends ExternalExit implements IMConstants
{
public StructureData setDataOut(StructureData irData)
throws MXException, RemoteException
{
IntegrationContext cntx = IntegrationContext.getCurrentContext();
String ompGUID = cntx.getStringProperty(OMPGUID);
.
.
.

}
}
```

Additional features of the invocation channel include user exit Java classes and an XSL mapping layer. You can configure the XSL mapping layer to do XML mapping or data transformation.

If the object structure that you need for your integration module invocation does not exist, you can create an object structure in the Object Structures application. You also can use XML with the MXINTOBJECT object structure to create object structures as part of the installation process.

Java class implementation

Integration modules can be implemented to use Java class files. Using a Java class file eliminates the need for integration component registration and configuration. Additionally, all the underlying integration module implementations are transparent to the process management product.

Java class integration modules must implement the `psdi.iface.mic.ServiceInvoker` Java interface. The service invoker Java interface is included in the `businessobjects.jar` file. Include the integration module Java class in the system class path at run time.

The service invoker Java interface has variations of the following method signature:

```
public byte[] invoke(Map String,Object metaData, MboRemote sourceMbo,
MboRemote targetMbo, String endPointName) throws MXException,
RemoteException;
```

- `metaData` is a map of the name and value properties that includes:
  - The integration module name and version.
  - The logical management operation name and name space.
  - The operational management product globally unique identifier.
  - The endpoint name and any endpoint properties that are being overwritten.
- `sourceMbo` is the source object that you defined on the logical management operation.
- `targetMbo` is the response object that you defined on the logical management operation
- `endPointName` is the name of the endpoint that you use for communication with the operational management product.

If you configure the integration module to implement multiple logical management operations, the integration module must determine which logical management operation is being called. At run time, the integration module retrieves the LMONAME and LMONAMESPACE properties from the `metaData` input map:

```
import psdi.iface.omp.IMConstants;
.
.
.
String lmoName = metaData.get(IMConstants.LMONAME);
String lmoNamespace = metaData.get(IMConstants.LMONAMESPACE);
```

The integration module can retrieve logical management operation data from the source object. The following example code retrieves logical management operation values from the source object:

```
String guid = sourceMbo.getString("GUID");
int packID = sourceMbo.getInt("PACKID");
boolean hasSubs = sourceMbo.getBoolean("HASSUBS");
```

In the example, the logical management operation has an alphanumeric input attribute called globally unique identifier, an integer attribute called PACKID, and a Boolean attribute called HASSUBS.

Service invoker methods can take a MboSetRemote set of values as a source input, instead of a single MboRemote value. In some cases, the integration module passes all of the objects in the object set to the operational management product. In other cases, the integration module passes only the first object in the set. There are no set rules that apply to the integration module behavior, but you must clearly define the expected behavior in the logical management operation description.

Integration module processing

Process management products use a system action to initiate an integration module. The action can be associated with an escalation or a workflow, or can be initiated from a menu or a button. The process management product provides an action class that invokes the integration module and processes the response that the integration module returns.

Identification of integration components

The initiation action class identifies the components required for the integration, including the logical management operation, the operational management product, and the integration module.

The logical management operation runs a process, such as software deployment, on a configuration item by using an operational management product. The process management product provides the necessary input to the integration module, based on the input business objects and attributes that are defined on the logical management operation.

The Java action class identifies an operational management product that runs the logical management operation on the selected configuration item. When you install configuration items and operational management products, the component relationships are also installed.

An integration module record identifies which logical management operations are supported on an operational management product. When the integration modules, configuration items, and operational management products are registered, the process management product Java class performs a lookup. The Java class determines which integration modules that it invokes based on the configuration items that it uses.

The integration framework includes a service that provides the following utility methods to assist the class with integration module lookups.

Utility method	Function
psdi.iface.app.omp.OmpSetRemote **getOMPListForIM** (*String imName*, *String imVersion*)	Retrieves a list of the operational management products that are associated with the integration module.
psdi.iface.app.im.MaxIMSetRemote **getIMListForLMO** (*String lmoName*, *String lmoNamespace*)	Retrieves a list of the integration modules that implement the specified logical management operation.
psdi.iface.app.im.MaxIMSetRemote **getIMListForLMOWithOMP** (*String lmoName*, *String lmoNamespace*)	Retrieves a list of the integration modules that implement the specified logical management operation on any operational management products.
psdi.iface.app.im.MaxIMSetRemote **getIMListForOMP** (*String ompGuid*)	Retrieves the list of the integration modules that implement at least one logical management operation on the operational management product.

Utility method	Function
Collection **getIMListForOMPAndLMO** (*String ompGUID*, *String lmoName*, *String lmoNamespace*)	Retrieves the list of integration modules that implement the logical management operation on the specified operational management product. Returns a collection of `psdi.iface.omp.OmpImLmoRelInfo` objects.
Collection **getIMListForOMPAndLMO** (*String ompHostname*, *String ompProductname*, *String ompManufacturer*, *String lmoName*, *String lmoNamespace*)	Retrieves the list of integration modules that implement the logical management operation on the specified operational management products. Returns a collection of `psdi.iface.omp.OmpImLmoRelInfo` objects.
`psdi.iface.omp.OmpImLmoRelInfo` **getPreferredIM** (*String ompGUID*, *String lmoName*, *String lmoNamespace*)	Retrieves the preferred integration module that implements the logical management operation on the specified operational management product.
`psdi.iface.omp.OmpImLmoRelInfo` **getPreferredIM** (*String ompHostname*, *String ompProductname*, *String ompManufacturer*, *String lmoName*, *String lmoNamespace*)	Retrieves the preferred integration module that implements the logical management operation on the specified operational management product.

Integration module invocation

Integration modules can be implemented either as Java classes or as invocation channels. Integration module instances are called service invokers because they implement the Java interface `psdi.iface.mic.ServiceInvoker`.

The service invoker interface hides the underlying implementation from the caller. The invocation of the integration module by the caller is the same, regardless of the underlying implementation.

Service invoker property map:

The get service invoker utility methods return a map of name and value pairs.

The `psdi.iface.omp.IMConstants` Java interface defines the names of the properties that are returned in the map. The property `IMConstants.IM` contains the instance of the integration module that the process management product invokes.

The property `IMConstants.ENDPOINTNAME` contains the name of the endpoint that is associated with in the operational management product, integration module, and logical management operation relationship. In most cases, the endpoint property is the value that the process management product passes to the integration module. However, in unusual cases, the action class overwrites the configured endpoint.

If a credential mapper is configured, the get service invoker utility methods call the credential mapper to retrieve the USERNAME and PASSWORD that is used for endpoint communication. These properties are returned by the utility methods in a map that is identified by the property `IMConstants.ENDPOINTPROPS`. The caller can overwrite any endpoint properties by adding them to this map.

Before the action class calls the integration module, it must populate the source object with the logical management operation input fields. The action class then passes the source object data to the integration module with the mapping that is

returned by the get service invoker utility method. The action class provides the logical management operation response object to the integration module when necessary.

The logical management operation response object requires attributes. The action class must ensure that the response object has the logical management operation attributes. The attributes can be persistent or nonpersistent. The response object typically contains the source object data.

Invoke methods:

The integration framework provides some invoke methods that the caller uses to invoke an integration module. Invoke methods use properties to determine what object data is returned to the caller. The properties also determine what action is taken on the returned data and how the integration framework communicates with the caller.

The service invoker interface has four invoke method signatures:

- `public byte[] invoke(Map <String,Object> metaData, MboRemote sourceMbo, MboRemote targetMbo, String endPointName)`
- `public byte[] invoke(Map <String,Object> metaData, MboRemote sourceMbo, MboSetRemote targetMboSet, int action, String endPointName)`
- `public byte[] invoke(Map <String,Object> metaData, MboSetRemote sourceMboSet, MboRemote targetMbo, String endPointName)`
- `public byte[] invoke(Map <String,Object> metaData, MboSetRemote sourceMboSet, MboSetRemote targetMboSet, int action, String endPointName)`

The action class passes the following properties when it calls one of the invoke methods on the instantiated service invoker.

Property	Description
metaData	The property map that the get service invoker utility method returns.
Source object and object set	The object with the input attributes that are defined on the logical management operation. This property can contain a null value.
Target object and object set	The object and object set that contains the return data. This property can contain a null value.
Action	This parameter indicates whether existing objects in the targetMboSet are updated, or new objects are added. The possible values for action are: • `psdi.mbo.MboSetRemote.INSERTONLY` • `psdi.mbo.MboSetRemote.UPDATEONLY`
Endpoint name	The name of the endpoint that the integration module uses for communication.

Operational management product service method:

Use the operational management product service method when the action class has a relationship to a configuration item, or when it has a configuration item globally unique identifier (CIGUID) attribute.

When the action class has an authorized configuration item, instead of an actual configuration item, you can use the configuration item globally unique identifier attribute of the authorized configuration item.

The following service method retrieves a list of the preferred integration modules (service invokers). The integration modules implement the specified logical management operation on the operational management products that manage the specified actual configuration item.

`public Collection` *`Map`* **`getServiceInvokerListForCIAndLMO`**`(`*`String actCIGUID, String lmoName, String lmoNamespace, UserInfo userInfo`*`)`

Each operational management product that has a relationship with the specified actual configuration item, returns the preferred integration module for the logical management operation.

Service invoker utility methods:

The `getServiceInvoker` utility includes methods that retrieve an instance of an integration module for a logical management operation and an operational management product.

The following table lists the utility methods provided with the `getServiceInvoker` utility.

Utility method	Function
Map<String, Object> **`getServiceInvoker`** (*`psdi.iface.omp.OmpImLmoRelInfo ompImLmoRelInfo, psdi.security.UserInfo userInfo`*)	Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.
Map<String, Object> **`getServiceInvoker`** (*`psdi.iface.app.im.OmpImLmoRelRemote ompImLmoRelRemote, psdi.security.UserInfo userInfo`*)	Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.
Map<String, Object> **`getServiceInvoker`** (*`String ompGUID, String imName, String imVersion, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo`*)	Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.
Map<String, Object> **`getServiceInvoker`** (*String ompGUID, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo*)	Retrieves the service invoker for the preferred integration module for the specified operational management product and logical management operation.
Collection<Map> **`getServiceInvokerListForCIAndLMO`** (*`String actCIGUID, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo`*)	Retrieves a list of the service invokers for the preferred integration modules that implement the specified logical management operation on the operational management products. The operational management products have a relationship with the configuration item.

Integration module response processing

If a logical management operation has a response object, the integration module updates the response object with the results from its invocation. The action class determines whether to save the results to the database. If the action class does not save the results, you can save the data when you view the response results in the user interface.

If the response object is the primary object from the application, or if the updated object is based on a relationship with the primary object of the application, the action class does not save the object. Instead, the user interface prompts you to save the object. However, if the updated object is unrelated to the primary application object, the action class saves and commits the changes.

Configuring integration modules

Integration modules receive data requests from process management products and return the response data from operational management products.

Creating integration modules

You can create an integration module to pass data between local and external applications. Depending on your needs, you can implement an integration module as an invocation channel or Java class. Both the invocation channel and Java class have access to the database and all the objects in memory at the time a process management product invokes an integration module.

Procedure

1. In the Integration Modules application, click **New Integration Module**.
2. In the **Name** and **Version** fields, enter a unique integration module name and version number combination. The version field must start with a `V` and be followed by an integer between 0 to 9, inclusive. The integer can then be followed by a decimal point (.) and up to 17 additional integers. For example, `V2.99.` or `V9.123456`.
3. Optional: Enter values in the following fields:

Option	Description
Operational Management Product Name	The operational management product that the integration module invokes. If an integration module works with multiple products, you do not specify a product name.
Operational Management Product Version	The version value of the product that the integration module invokes. If an integration module works with multiple versions of an operational management product, you do not specify a version value.
Handler Name	The protocol that the integration module uses to invoke an operational management product. If the invocation channel has an associated end point, you cannot configure a handler for the integration module.
Invocation Channel Name	The name of the invocation channel that the integration module runs. The integration module uses either an invocation channel or a Java class. Specify only one of them.

Option	Description
Class Name	The name of the Java class that the integration module runs. The integration module uses either an invocation channel or a Java class. Specify only one of them.

4. Click **Save Integration Module**.

What to do next

You can associate a logical management operation to an integration module to define the actions that process managers run. You also can associate an operational management product to an integration module to define the external applications that you can invoke from a process management product.

Selecting logical management operations for integration modules

You can use the **Select Operations** dialog box to associate one or more logical management operations with an integration module record. Logical management operations define an action that a process manager product runs from an application.

Before you begin

You first must associate the logical management operation with an operational management product before a process management product can run a logical management operation. The logical management operation records that you associate with an integration module that uses an invocation channel must have a source object name value. This value must be the same as the top source object that is registered to the invocation channel. Additionally, the logical management operation records must have a response object name value when the invocation channel that processes responses. This value must be the same as the top object registered to the invocation channel. You can define the logical management operation source and response object values in the Logical Management Operations application.

Procedure

1. In the Integration Modules application, select the integration module that you want to associate to the logical management operations.
2. On the **Logical Management Operations** tab, click **Select Operations**.
3. Select the logical management operations that you want to associate to the integration module record.
4. Click **OK**.
5. Click **Save Integration Module**.

What to do next

You can associate an operational management product to an integration module to define the external applications that you can invoke from a process management product.

Selecting logical management operations for operational management products:

You can use the **Select Operations** dialog box to associate one or more logical management operations to an operational management product. The association

that you make to the operational management product enables the logical management operation. The enablement indicates that the logical management operation is ready for use.

Before you begin

A logical management operation must be associated with an integration module before it can be enabled on an operational management product.

Procedure

1. In the Integration Modules application, select the integration module that you want to associate to the logical management operations.
2. On the **Operational Management Products** subtab, select the operational management product that you want to associate to the logical management operations.
3. In the Logical Management Operations for Operational Management Product table window, click **Select Operations**.
4. Select the logical management products that you want to enable for the operational management product.
5. Click **OK**.
6. Click **Save Integration Module**.

Associating a logical management operation with an integration module

You can associate a logical management operation with an integration module to define the actions that are taken on an operational management product. Process management products, such as Change or Deploy, call upon an integration module to run an operational management product. The operational management product then runs the logical management product, such as deploy software, and notifies the process management product of the action status.

Before you begin

You first must associate the logical management operation with an operational management product before a process management product can run a logical management operation. The logical management operation records that you associate with an integration module that uses an invocation channel must have a source object name value. This value must be the same as the top source object that is registered to the invocation channel. Additionally, the logical management operation records must have a response object name value when the invocation channel that processes responses. This value must be the same as the top object registered to the invocation channel. You can define the logical management operation source and response object values in the Logical Management Operations application.

About this task

The invocation pattern value determines whether the logical management operation is long-running. It also determines how the process management product expects a response from the operational management product.

Procedure

1. In the Integration Modules application, select the integration module that you want to associate with a logical management operation.

2. On the Logical Management Operations subtab, click **New Row**.
3. Enter values in the following fields:

Option	Description
Logical Management Operation Name	Identifies the actions that integration modules support, and the actions that the process management products request.
Namespace	Identifies the domain for the logical management operation name. An example is `com.ibm.mss`.

4. Click **Save Integration Module**.

What to do next

You can associate an operational management product to an integration module to define the external applications that you can invoke from a process management product.

Associating an operational management product with an integration module:

You can associate an operational management product to an integration module to define the external applications that you can invoke from a process management product. Process management products, such as Change or Deploy, call upon an integration module to run an operational management product. The operational management product then runs the logical management product, such as deploy software, and notifies the process management product of the action status.

Before you begin

Before a process management product can call an integration module to run a logical management operation on an operational management product, configure the following components:

- Define and configure a logical management operation
- Associate the integration module with the logical management operation and operational management product
- Enable the logical management operation for the integration module on the operational management product

About this task

You can build and configure relationships between integration modules, operational management products, and logical management operations. You can also perform the following actions:

- Associate multiple operational management products with an integration module
- Enable logical management operations on the operational management product for the integration module
- Specify a default integration module to use for a particular logical management operation on an operational management product
- Configure the end point for an integration module and operational management product combination when the invocation channel has no associated end point

Procedure

1. In the Integration Modules application, select the integration module that you want to associate with an operational management product.
2. On the Operational Management Products subtab, click **New Row**.
3. Enter a values in the following fields:
   - **Operational Management Product**
   - **End Point**
4. In the Logical Management Operations table window, click **New Row**.
5. Enter a value in the **Logical Operation Management Name** field. Default values display in the **Namespace** and **Description** fields.
6. Optional: Select the **Is Primary** check box to make the integration module the default integration module for the selected logical management operation. Select the **Is Primary** check box when you enable the selected logical management operation on more than one integration module record.
7. Click **Save Integration Module**.

What to do next

You can associate a logical management operation with an integration module to define the actions that are taken on an operational management product.

Configuring logical management operations

You can create a logical management operation to define an action that a process management product executes from the integration framework. You also can define what specific object field attributes that they logical management operation uses.

Creating logical management operations

You can define the invocation pattern on a logical management operation. The pattern value determines whether the logical management operation is long-running and how the process management product expects a response from the operational management product.

Before you begin

You must define a logical management operation relationship to an operational management product and integration module before you can invoke any processes.

Procedure

1. In the Logical Management Applications application, click **New Logical Management Operation**.
2. In the **Name** field, specify a logical management operation identifier.
3. Enter values in the following fields:

Option	Description
Source Object Name	The input object for the logical management operation.
Invocation Pattern	The pattern of a logical management operation invocation.
Name Space	The secondary identifier for the logical management operation record.

Option	Description
Response Object Name	The output of the object for the logical management operation.

4. Click **Save Logical Management Operation**.

Adding attributes to logical management operations

You can add attributes to a logical management operation to identify the specific object field attributes the logical management operation data source and target use. You can define the individual attribute entries for both input and output logical management operation invocations.

About this task

Input object attributes identify the data that the process management product passes to the integration module. Output object attributes identify the data that the integration module returns. Only the respective attributes of the selected source and response objects are available for field selections. If you have not defined either a source or response object, all object attributes are available for selection.

Procedure

1. In the Logical Management Applications application, select the logical management operation for which you want to add an attribute.
2. In the Attributes for LMO window, click **New Row**.
3. Enter a value in the **Name** field.
4. Optional: Clear the **Input** check box to indicate that the attribute is for the output of a logical management operation invocation.
5. Optional: Clear the **Required** check box to indicate the field attribute is not required in the logical management operation invocation.

Launch in Context feature

You use the Launch in Context application to create and modify launch entry records which open an external application in the same or a different browser session. You can open applications independently or as part of an application integration scenario.

The launch entry record can utilize the following options in the Application Designer to open an external application:

- Action menu items
- Hyperlinks
- Buttons

You use the Launch in Context application to create and update launch entries. A launch entry defines a URL that opens a console for an external application. The launch entry can pass data, referred to as context, from the application to the external console. You can configure a console URL for any application with a web-based console. Additionally, you can configure console URLs for consoles that use Java™ Web Start. You cannot use a launch entry to open applications that are not enabled for the web. You can configure a launch point from any application.

Preparation of the external application

Most external applications have a web-based console that you can open from a URL in a web browser. The launch-in-context feature supports the web application (servlet or JSP), portal, and Java Web Start console types.

To perform a navigation, the external application console must support the land-in-context capability that accepts data that is passed to it in a URL. The external console uses the URL to open a window with the data that was passed in the URL.

If the external application does not have a land-in-context capability, the launch-in-context feature can open a standard start page within the console, without the contextual data. If the application supports single sign-on authentication, application users are authenticated and directed to the external console. If single-sign on authentication is not implemented, the external console opens a login panel to authenticate the application user.

Launch entry URL into an external application

A launch entry URL value can contain substitution variables that use data from the related business objects that you are viewing in the application.

For example, for a launch entry that is implemented in the Person application, you can substitute the name value into the URL string by using the attribute name, `{attributename}`. The following URL is a URL for the Person application that uses the PERSONNAME attribute:

https://extsyshost:9045/tcWebUI/interactionhandler?actionId=viewPerson
&Person={PERSONNAME}

You can also provide an attribute from an object that is related to the main business object by specifying the relationship name and the attribute name `{relationshipname.attributename}`. The following URL includes a city attribute from an address. Use this URL in the People application when the ADDRESS object is related to a PERSON object with a relationship named ADDRESS:

https://extsyshost:9045/WebUI/interactionhandler?actionId=viewCity
&cityname={ADDRESS.CITY}

Use the `{sourcetoken}` and the `{reportinghostname}` values in a URL when you want to launch to an operational management product console.

Launch entry URL into a product application

A launch entry URL value can contain attribute variables and a SQL Where clause variable.

An external application can launch into the product, open a specific product application, and search for specific business objects to display.

When launching into the product, the product sign on screen is presented. After the user signs on, the specified application opens. Optionally, you can include the username and password in the URL to bypass the sign on screen but only if a Secure Socket Layer (SSL) is configured to secure HTTP access.

The URL to launch into the product has the following format:

http://<*server*>:<*port*>/<maximo>/ui/<*product.jsp*>?event=loadapp
&value=<*appID*>

The following example URL opens the Work Order Tracking application and displays the work order with an ID of 1000

http://<*server*>:<*port*>/<maximo>/ui/<maximo.jsp>?event=loadapp
&value=<wotrack>&attrname1=<WONUM>&attrvalue1=<1000>:

The search for a record has similar behavior to the List tab in an application, where the filter row has multiple values filled in across the columns. The names of the attributes are the database names of the primary MBO for the application being launched.

If advanced searching is required, you can add a SQL WHERE clause to the URL, in the format used in the following example:

http://:<*server*>:<*port*>/maximo/ui/maximo.jsp?event=loadapp&value=wotrack
&sqlwhere=WONUM%3D1000

The sender must provide valid URLs. In the previous example, an invalid "=" value was converted to "%3D".

If a URL includes both attribute-type parameters and a SQL Where clause parameter, the SQL Where clause parameter is used and the attribute-type parameters are ignored.

Enabling launch-in-context

To open a browser window in an external application that can contain contextual data, a user must be able to activate the launch in the source application. The user must also have security rights to view the menu item or toolbar button. The target application must be configured to accept the window request from a remote application, and arrangements must be in place to secure the transaction.

Creating a launch entry

You can create a launch entry to open an external application in the same or in a new browser session. Launch entry records create website-based links between applications and external operational management products or websites.

Procedure

1. In the Launch in Context application, click **New Launch Entry**.
2. In the **Launch Entry Name** field, specify a launch entry identifier.
3. In the console URL field, specify the URL for a website or the console for an operational management product that you want to open in a browser session.
4. In the **Target Browser Window** field, specify one of the following values:
   - The `_usecurrent` value (default) opens an application or website in the current browser session.
   - The `_blank` value open an application or website in a new browser session.
5. Optional: If the target of the launch entry is a console for an operational management product, specify its name and version in the **OMP Product Name** and the **OMP Version** fields.
6. In the Launch Contexts table, click **New Row**.

7. In the **Resource Object Name** field, you can specify a business object that restricts use of the launch entry record to applications that support this object. A launch entry can support single or multiple business objects. No restrictions apply to the use of the launch entry if you do not select any business objects. Launch entry classification value attributes. The classification value can restrict the displayed launch entries. Multiple business objects support the classification attribute. The classification restriction is implemented at run time when you use system conditions. Out of box conditions are available to control your launch entry behavior. You also can create your own conditions. For example, if you specify `PERSON` in this field, you can use the launch entry only in the People application. You can use the launch entry record in any application when the field value is null.
8. In the **Resource Classification** field, if classification attributes are defined for the specified resource object, you can select attributes that restrict the display of launch entries. Many business objects support classification attributes. The classification restriction is implemented at run-time when you use system conditions. Predefined conditions are available to control launch entry behavior. You also can create your own conditions.
9. Select the **Include Child Classifications** check box to include resource classifications for child objects in the launch entry record.
10. Click **Save Launch Entry**.

What to do next

You can associate launch entry records with signature options in the Application Designer application. These associations define and control the menu actions for launch entry use.

Properties specific to operational management products:

When you create a launch entry record, some properties are specifically for use with operational management products.

When you associate an operational management product to a launch entry, the framework searches for the product name on the operational management product servers in the database that manages the configuration item. If you provide the operational management product version, the framework searches for the specified version. The server data populates the `{sourcetoken}` and `{reportinghostname}` variables in the launch URL.

To determine what configuration item you are working with, the integration framework looks for a configuration item globally unique identifier (CIGUID) attribute in the business object that you are viewing. If you are viewing the configuration item business object, the globally unique identifier (GUID) attribute is used instead.

The source token is an attribute of the configuration item and operational management product server relationship. The operational management product console accepts the source token as the configuration item identifier. When you include a `{sourcetoken}` in the launch URL, the framework replaces it with the corresponding source token for the selected operational management product server.

The reporting host name is the host name of the selected operational management product server. When the `{reportinghostname}` value is in the URL for the launch

entry, the framework replaces it with the host name of the selected operational management product. The source token and host name information is loaded from the discovery engine. If the host name information is not loaded from the discovery engine, you must add the operational management product server information to your database.

Configuring a signature option for a launch point

When you create a signature option for a launch point, it becomes available to use within the user interface controls.

Procedure

1. In the Application Designer, select the application where you want to configure a launch point.
2. Select the **Add/Modify Signature Options** action.
3. In the Add/Modify Signature Options window, click **New Row**.
4. Specify values in the **Option** and **Description** fields.
5. In the Advanced Signature Options table, select the **Associate to Launch Entry To Enable The Launch in Context** option. To access the Advanced Signature Options table, scroll to the end of the Add/Modify Signature Options window, and click **Maximize** to show the available options.
6. In the **Launch Entry Name** field, specify the name for the launch entry.
7. Click **OK** to return to the Application Designer.
8. Click **Save Application Definition** to commit the modifications to the application to the database.

Adding a launch point to an application menu

After you configure a signature option for a launch point, you can add the option to an application menu. The procedure is similar for adding the launch point to an action or to a toolbar menu.

Procedure

1. In the Application Designer, choose one of the following actions:
   - **Add/Modify Select Action Menu**
   - **Add/Modify Toolbar Menu**
2. Click **New Row**.
3. In the **Element Type** field, specify the OPTION value.
4. In the **Key Value** field, specify the name of the signature option that you configured for the launch point.
5. In the **Position** field, specify a number to indicate the relative position of the item in the menu.
6. In the **Tabs** field, select one of the following values:
   - Select the MAIN value if the launch URL contains substitution variables.
   - Select the ALL value if the launch URL does not contain substitution variables.
7. Specify values in other, optional fields as appropriate.
8. Click **OK** to return to the Application Designer.
9. Click **Save Application Definition** to commit the modifications to the application to the database.

What to do next

Before users can see the new menu item or toolbar button, you must grant user and group access privileges to it. Grant the privileges in the Options section of the **Applications** tab in the Security Groups application.

Adding a button as a launch point

To add a button to an application that acts as a launch point, configure a button to use the signature option for the launch point.

Procedure

1. In the Application Designer, click **Control Palette**.
2. Drag a push button control to the application workspace.
3. Open the **Properties** window for the push button control.
4. In the **Label** field, specify the name that you want to appear on the button.
5. In the **Event** field, specify the name of the signature option.
6. Click **OK** to return to the Application Designer.
7. Click **Save Application Definition** to commit the modifications to the application to the database.

Adding a condition to a launch point

You can create a condition and then associate the condition with the signature option for the launch point. When you associate a condition with a signature option, the user interface behavior is changed based on the condition and the data that is being viewed.

Procedure

1. In the Security Groups application, select the group for which you want to apply a condition.
2. In the Application tab, select the application for which you want to apply the condition.
3. In the Options table, select the condition value that you created.
4. Click **Save Group**.

Signature option conditions:

You can apply conditions to control the user interface behavior based on the data that is being viewed.

The integration framework provides a predefined condition class, psdi.iface.app.launch.LaunchCICondition, to hide launch entry menu items when the current object classification does not match the launch entry classification value. This condition applies to any object that has a classification attribute.

Launch entry menu items can also be hidden when the operational management product that is configured on the launch entry does not manage the configuration item. This condition applies to launch entries that you associate with a configuration item object or actual configuration item object.

You can configure a condition to use the predefined condition class. You also can configure a condition to implement a custom condition using the Conditional Expression Manager application.

The launch point is available from the application, regardless of the data that is being viewed, when you do not use a condition. The data you view is restricted to the signature option security settings for a user group. You also can configure your security settings to hide a launch point, based on set group access privileges.

If you use a Java condition class, you must change the condition EXPRESSION attribute to the name of the launch entry. The Java class can identify which launch entry is executed. The expression attribute value must be an exact match to the name of the launch entry and values are case-sensitive.

Integration reference information

Reference information includes a guide to the XML structures and schemas used by the integration framework, the system properties you can set, and predefined collaboration switches provided with applications.

Integration system properties

System properties define the behavior and characteristics of the integration framework. To review or change integration framework properties, filter for the properties in the System Properties application.

General integration properties

To see a list of general integration properties, specify `mxe.int` as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 46. General integration properties

Property	Description	Default value
mxe.int.containerdeploy	Deploy web services to the application server container. When set to 0 (false), web services are deployed to the product container.	0
mxe.int.credentialmapperclassname	Credential mapper classname is a class file that can be used for mapping credential information when an integration module is implemented.	No default value
mxe.int.genboolasbool	Generate Boolean as schema Boolean.	1
mxe.int.globaldir	Specifies the location of a global directory which stores files that are related to integration	No default value
mxe.int.queueusercachesize	Number of users that are cached for inbound queue messages.	10
mxe.int.resolveschema	Resolves all schema includes to contain inline schema definition.	1
mxe.int.servicedeployer	Web services deployer class is a custom Java class for web service deployment when the default deployer class is not used.	No default value
mxe.int.uddiinqurl	Represents the integration UDDI registry inquiry URL.	No default value
mxe.int.uddiname	Represents the integration UDDI registry user ID.	No default value
mxe.int.uddipassword	Integration UDDI registry password.	No default value
mxe.int.uddipuburl	Integration UDDI registry publish URL.	No default value

Table 46. General integration properties (continued)

Property	Description	Default value
mxe.int.validatedbupdates	Validates the database updates completed by integration. When set to 1 (true), the deletion of business objects, attributes, indexes, and relationships by a user through the Database configuration application are validated against integration content. The validation ensures that the data that is deleted is not referenced by an integration component. If a reference exists, the user is not able to complete the delete action.	1
mxe.int.validatemmpackage	Validates the Migration Manager database updates by integration.	0
mxe.int.verifywebappurl	Verifies web application URL when schema files are generating.	1
mxe.int.webappurl	Represents the integration web application URL. Configure this property to contain the correct host name and port number.	http://localhost/meaweb
mxe.int.wsdlcurrentschema	Shows the current schema definition in WSDL.	1
mxe.int.wsdlincludesschema	Includes the schema directly in the WSDL.	1
mxe.int.wsdlnamespace	Represents the integration WSDL namespace.	http://www.ibm.com/maximo/wsdl
mxe.int.xmlnamespace	Represents the integration XML namespace.	http://www.ibm.com/maximo
mxe.int.binarytext	Converts a text value to base 64 encoded value.	0
mxe.int.defaultaction	The default action for flat file import.	AddChange
mxe.int.defaultoperation	The default operation for the application export.	Sync
mxe.int.dfltuser	Represents the Integration default login user.	mxintadm
mxe.int.doclink.maxfilesize	Represents the maximum file size (MB) for attachments that are included as part of an integration message.	10
mxe.int.enabledatemillis	Enables the dates with milliseconds part.	0
mxe.int.expupdatesender	Updates the **SENDERSYSID** field on the primary object during data export.	0
mxe.int.extracttrycount	The File Extract Retry Count is the number of times an error message is retried during data import when using file-based error management.	0
mxe.int.flatfiledelimiter	Integration flat file text delimiter is the default delimiter value that is used for application import enablement and for data import.	,
mxe.int.flatfilenewline	Retains new line character in flat files. For fields, such as descriptions, that can contain new line characters, the characters are retained in the integration messages when the property value is 1 (true).	0
mxe.int.interactiveimport	Performs the application import as interactive.	0
mxe.int.keyresponse	Provides response content for inbound integration messages for all operations. When set to 1 (true), response content, that includes the primary object key values, is provided for all service operations. When set to 0 (false), response content is provided for Query and Create operations only.	1
mxe.int.maxextractdocs	Represents the number of error documents that are written to each temporary file when an extract file is building .	1000
mxe.int.mdbdelay	Represents the wait time in milliseconds before a message from the error queue is processed.	-1

Table 46. General integration properties (continued)

Property	Description	Default value
mxe.int.propagateuser	Propagate authenticated user through the inbound queue. When set to 1 (true), the user of the integration message is saved with the queue message and used during the processing of the message because it is processed from the queue to the business objects.	0
mxe.int.savemessage	Indicates the save JMS message.	0
mxe.int.setclobasaln	Controls the truncation of characters that are sent to interface tables.	0
mxe.int.textqualifier	The flat file text qualifier is the default text qualifier value in application import enablement and in data import.	"
mxe.int.updatecoafromglcomp	Updates the Chart of Accounts that contain an identified component. When set to 1 (true), processing of inbound GL component data initiates related updates to any chart of account data that references the GL component.	1
mxe.int.usescientific	Uses scientific notation for double values.	1
mxe.int.validatexmltext	Validates XML element value for invalid XML characters. When set to 1 (true), an outbound message is validated to ensure that all data in the message uses valid XML characters. If messages contains invalid characters, the operation stops and no outbound message is delivered.	0
mxe.int.whereclausepolicy	Sets the where-clause policy for an integration query.	parse
mxe.int.adminfromemail	The email address FROM integration administration, which is used as the From email address when integration initiates an email. Must be a valid email address format, such as from@example.com.	No default value
mxe.int.admintoemail	The email address TO integration administration, which is used as the To email address when integration initiates an email. Must be a valid email address format, such as to@example.com. You can provide more than one email address in a comma-separated list.	No default value
mxe.int.usedbinfoforifacetb	Sets the processing for interface tables. When set to 1, the interface process refers to the existing table to determine what to insert into the table rather than the object structure definition.	0
mxe.int.enableosauth	Enables the authorization configuration of object structures.	1

REST integration properties

To see a list of REST API integration properties, specify `mxe.rest` as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 47. REST API integration properties

Property	Description	Default value
mxe.rest.format.json.mimetypes	The REST supported mime types for JSON.	application/json
mxe.rest.format.xml.mimetypes	The REST supported mime types for JSON.	application/xml,text/xml
mxe.rest.handler.mbo	The REST MBO resource handler.	com.ibm.tivoli.maximo.rest. MboResourceRequestHandler

Table 47. REST API integration properties (continued)

Property	Description	Default value
mxe.rest.handler.os	The REST object structure resource handler.	com.ibm.tivoli.maximo.rest.OSResourceRequestHandler
mxe.rest.handler.ss	The REST standard service resource handler.	com.ibm.tivoli.maximo.rest.MaxServiceResourceRequestHandler
mxe.rest.serializer.mbo.imglib.image	The REST serializer for the imagelib MBO for image format.	com.ibm.tivoli.maximo.rest.ImageLibSerializer
mxe.rest.serializer.mbo.json	The REST serializer for MBO for JSON format.	com.ibm.tivoli.maximo.rest.MboJSONSerializer
mxe.rest.serializer.mbo.xml	The REST serializer for MBO for xml format.	com.ibm.tivoli.maximo.rest.MboXMLSerializer
mxe.rest.serializer.os.json	The REST serializer for object structures for JSON format.	com.ibm.tivoli.maximo.rest.OSJSONSerializer
mxe.rest.serializer.os.xml	The REST serializer for object structures for xml formats.	com.ibm.tivoli.maximo.rest.OSXMLSerializer
mxe.rest.serializer.ss.json	The REST serializer for standard services for JSON format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseJSONSerializer
mxe.rest.serializer.ss.xml	The REST serializer for standard services for xml format.	com.ibm.tivoli.maximo.rest.ServiceMethodResponseXMLSerializer
mxe.rest.webappurl	Token Authentication on Web Application URL.	No default value
mxe.rest.mbo.blockaccess	Blocks access to the comma-separated list of MBOs.	No default value
mxe.rest.mbo.defaultformat	The REST default format for all MBOs.	xml
mxe.rest.mbo.imglib.defaultformat	The REST default format for the MBO imglib.	image
mxe.rest.os.blockaccess	Blocks access to the separated list of object structures.	10
mxe.rest.os.defaultformat	The REST default format for all object structures.	xml
mxe.rest.ss.defaultformat	The REST default format for all standard service response	xml
mxe.rest.supportedformats	The REST supported formats for a response.	xmljsonimage
mxe.rest.whereclausepolicy	Sets the where clause policy for REST query.	parse

OSLC integration properties

To see a list of OSLC integration properties, specify mxe.oslc as a filter term in the System Properties application. For Boolean properties (true/false), a value of 0 means false, and a value of 1 means true.

Table 48. OSLC integration properties

Property	Description	Default value
mxe.oslc.dfltconsumerversion	The default OSLC version that the consumer uses.	2
mxe.oslc.dfltversion	The default OSLC version for an OSLC provider.	2
mxe.oslc.enableprovider	Enables the OSLC provider.	1
mxe.oslc.idleexpiry	Indicates the idle expiry time.	300
mxe.oslc.webappurl	The provider's public URL.	http://localhost/maximo/oslc/

Table 48. OSLC integration properties (continued)

Property	Description	Default value
mxe.oslc.collectioncount	Adds the total count in the OSLC collection.	0
mxe.oslc.defaultep	The default OSLC Endpoint.	OSLCDEFAULT
mxe.oslc.defaultformat	The default format for OSLC.	oslcjson
mxe.oslc.errorresponse	The OSLC Error Response Format.	1
mxe.oslc.preferproviderdesc	Prefers OSLC provider description for resource registry reconciled URLs	false
mxe.oslc.prefersmallpreview	Prefers small preview for OSLC consumer.	false
mxe.oslc.prettyjson	Pretty printed JSON.	0
mxe.oslc.prettyrdf	Pretty printed RDF.	0
mxe.oslc.prqueryep	The Provider Registry Query Endpoint.	PROVIDERREGISTRY
mxe.oslc.prcreateep	Represents the Provider Registry Create Endpoint.	No default value

Integration XML

Most integration XML messages are based on an object structure and an operation that a channel or service performs. Standard services, however, do not support object structures and predefined XML schemas are used to construct these XML messages.

Overview

When you configure object structures, the integration framework specifies an appropriate XML schema that defines the content and structure of integration messages.

Message content

The following channels and services can be used for XML messages based on an object structure:

- Invocation channel
- Publish channel
- Enterprise service
- Object structure service

The following operations are supported:

- Create
- Delete
- Invoke
- Publish
- Query
- Sync
- Update

Operation values are case-sensitive.

Message structure

The standard format for an integration XML message begins with a root element that contains an object structure element. The object structure element contains elements for the primary object and its fields and for any child objects that are defined for the object structure. The following example is based on the PERSON object structure, where the primary object and its fields precede child objects and their fields.

```
<?xml version="1.0" encoding="UTF-8"?>
<max:SyncMXPERSON xmlns:max="http://www.ibm.com/maximo"> (Root element)
  <max:MXPERSONSet> (Sets the object structure element)
    <max:PERSON> (Object element for primary object)
       <max:PERSONID>Value</max:PERSONID> (Object field element)
       <max:EMPLOYEETYPE>Value</max:EMPLOYEETYPE> (Object field element)
.
.
.
       <max:PHONE> (Child object element)
         <max:PHONENUM>Value</max:PHONENUM> (Child object field element)
         <max:TYPE>Value</max:TYPE> (Child object field element)
       </max:PHONE>
       <max:EMAIL> (Child object element)
         <max:EMAILADDRESS>New value to update</max:EMAILADDRESS>
         <max:TYPE>Value</max:TYPE> (Child object field element)
       </max:EMAIL>
.
.
.
    </max:PERSON>
  </max:MXPERSONSet>
</max:SyncMXPERSON>
```

The name of the root element is a concatenation of the operation specified for the channel or service and the name of the object structure. In this example, the `SyncMXPERSON` root element combines the Sync operation and the MXPERSON object structure.

Each element can contain one or more attributes. In this example XML message, the root element includes the namespace attribute.

Schema generation

To generate the schema and view the generated XML, filter for a specific record and select the Generate Schema/View XML action in any of the following applications:

- Enterprise Services
- Invocation Channels
- Object Structures
- Publish Channels
- Web Services Library

XML structure

A typical integration XML message has a root element, an object structure element, and elements for the objects that are defined for the object structure. Object elements contain elements for object fields and elements can contain one or more attributes. Attribute names and values are case-sensitive.

Root element and attributes:

The root element of an XML message is based on an object structure and an operation specified for the channel or service used for the communication. The root element can contain one or more attributes.

The following table shows the attributes that can apply to root elements. Attribute names and values are case-sensitive. All attributes are optional.

Attribute	Description	Type	Applicable to
`baselanguage`	The base language in which the content values are supplied.	string	All input and output operations
`creationDateTime`	Date and time when the content is generated.	dateTime	All input and output operations
`maximoVersion`	The major version, minor version, build, and database build that is generated for all published XML.	MaximoVersionType	All input and output operations
`messageID`	Unique identifier generated for all messages.	string	All input and output operations
`transLanguage`	The language in which the values for multilanguage-enabled fields are supplied.	string	All input and output operations
`event`	The origin of an outbound XML message. Valid values are: • 0 (false), indicates that the message is generated by the data export feature. • 1 (true), indicates that the message is generated by an outbound integration event listener (that is, data entry in an application).	eventType	All output operations
`uniqueResult`	Specifies whether a query expects one record or multiple records in a response. If the value is 0, or the attribute is not specified, the query can return multiple records. If the value is 1, the query can return a single record only and, otherwise, an error occurs.	Boolean	Queries (input)

Attribute	Description	Type	Applicable to
maxItems	If a query can return multiple records, this attribute limits the number of records to be returned at one time. If this attribute is not specified, the response contains the entire result set.	positiveInteger	Queries (input)
rsStart	Specifies the first record to return in the response. If not specified, the response starts with the first record in the result set. If the number of results in the result set is lower than the rsStart value, the response returns no records. If a maxItems value is specified, the response returns the specified number of records, starting with rsStart value, if one is set. For example, if maxItems=10 and rsStart is not specified, the response returns results 1 through 10. To receive results 11 through 20, resend the query with rsStart=11.	integer	Queries (input)
rsStart	This value matches the rsStart value in the corresponding query. If the query contains a maxItems value, the rsStart value in requests for additional records is rsStart + rsCount + 1. If this attribute is not specified, the response starts with the first record in the result set and includes the number of records specified by the rsCount attribute.	integer	Responses (output)
rsCount	The number of records returned in a message. If the original query specifies a maxItems value, the rsStart value for the subsequent request for additional records is rsStart + rsCount + 1.	integer	Responses (output)

Attribute	Description	Type	Applicable to
rsTotal	The total number of records in the result set. If the query does not specify a maxItems value, the rsTotal value is the same as the rsCount value.	integer	Responses (output)

Object structure element:

The object structure element contains an element for the primary object and elements for any child objects defined for the object structure. This element can support multiple occurrences of the primary object and its child objects.

The primary object element contains elements for each object field. Child object elements, containing elements for their object fields, are listed after the primary object.

Object elements and attributes:

An object element contains elements for the object fields. Each object element can contain one or more attributes.

The following table lists the attributes that can apply to an object element. Attribute names and values are case-sensitive. All attributes are optional.

Attribute	Description	Type	Applicable to
action	This value is derived from the action attribute of the primary object within the message. For outbound messages, this attribute is for informational purposes only. For inbound messages, the processing logic uses this value only for the Sync operation. For other operations, this value is ignored.	ProcessingActionType	All input and output operations
relationship	Identifies the relationship that the system uses to retrieve the object using the parent object.	string	All input and output operations
deleteForInsert	Identifies a child object that must be deleted before reinserting. This attribute applies only to inbound messages where the operation is Sync and the action is Change.	string	All input and output operations

Related reference:

"Action attributes" on page 350
An action attribute is an optional attribute that specifies to the receiving system the type of processing to perform on an XML message. Action attributes apply to inbound XML messages that synchronize data, using the Sync operation, and to outbound XML messages that use publish channels.

Object field elements and attributes:

If the same field is included in the key for both a parent object and a child object in an object structure, the field is contained in the parent object only. Each object field can include one or more attributes.

Changed field attribute:

A Boolean value can be set to 1 (True) in outbound XML messages to indicate that the value in the field is changed. The changed attribute is not set in XML generated by the data export feature.

The changed attribute is set in outbound XML messages only when the transaction satisfies all of the following conditions:

- An outbound, event-based transaction creates the message.
- The Change action attribute or the Replace action attribute is set on the primary object.
- The sending object structure has the same parent-child object relationship as the receiving application.

In the following example, the Replace action attribute is set for the parent object and the changed attribute is set to 1 for the ADDRESSLINE field.

```
<MXPERSON>
    <PERSON action="Replace">
        <PERSONID>123</PERSONID>
        <ADDRESSLINE1 changed="1" >1 Main Street</ADDRESSLINE1>
```

For general ledger (GL) type fields, the changed attribute is always set on the name element, which is GLDEBITACCT in the following example:

```
<GLDEBITACCT changed="1">
     <VALUE>6600-800-SAF</VALUE>
     <GLCOMP glorder="0">6600</GLCOMP>
     <GLCOMP glorder="1">800</GLCOMP>
     <GLCOMP glorder="2">SAF</GLCOMP>
</GLDEBITACCT>
```

General ledger field attribute:

The glorder attribute is set in the GLCOMP elements in fields that identify general ledger (GL) accounts, such as a GLDEBITACCT field. Each GLCOMP element contains part of the account number and the glorder attribute identifies how to combine these elements to construct the GL value.

In outbound XML messages, the value of a general ledger type field, including delimiters, is set in the VALUE child element within the field. The components of the value, based on the database definition of the components, are included in GLCOMP elements. The glorder attribute in each GLCOMP element identifies the order of the component, starting from 0 (zero), to a maximum of 20. In the following example, the account number has three GL components

```
<GLDEBITACCT>
    <VALUE>6600-800-SAF</VALUE>
    <GLCOMP glorder="0">6600</GLCOMP>
    <GLCOMP glorder="1">800</GLCOMP>
    <GLCOMP glorder="2">SAF</GLCOMP>
</GLBDEBITACCT>
```

Inbound XML messages can set a GL account number in the VALUE element or in GLCOMP elements with associated glorder attributes. If the message includes GLCOMP elements, the account number is recreated based on the delimiters defined in the GLCONFIGURE table. If both VALUE and GLCOMP elements are included in the message, the VALUE element is used and the GLCOMP elements are ignored.

Translatable field attribute:

The langenabled attribute is set to 1 (true) on every field that can be translated in outbound XML messages.

Synonym field attribute:

For outbound messages, fields that are associated with a synonym-type domain specify the corresponding internal value using the maxvalue attribute. This value is available for customization or exit processing as required. The attribute is informational only, and is not used for inbound message processing.

In the following example, a maxvalue of NOLOT is set for the LOTTYPE field.

```
<MXITEM>
<ITEM>
<ITEMNUM>560-00</ITEMNUM>
<DESCRIPTION>Tubing, Copper-1 In ID X .030 In Wall Test
</DESCRIPTION>
<LOTTYPE maxvalue="NOLOT">NOLOT</LOTTYPE>
```

Encrypted field attribute:

When the data in a field, such as a password field, is encrypted, the Boolean attribute, maxencrypted, is set to 1 (true). External systems use this value to determine whether to apply a decryption process to the information in the field.

The following example uses the MXPERSUSER object structure to provide user information, including a password, and the maxencrypted attribute is set on the PASSWORD field in the MAXUSER object.

```
<MXPERUSER>
<PERSON action="Replace">
<PERSONID>123</PERSONID>
<MAXUSER>
<MAXUSERID>10</MAXUSERID>
<PASSWORD maxencrypted="1"> dmFzdG8=</PASSWORD>
<MAXUSER>
<PERSON>
<MXPERUSER>
```

Action attributes:

An action attribute is an optional attribute that specifies to the receiving system the type of processing to perform on an XML message. Action attributes apply to inbound XML messages that synchronize data, using the Sync operation, and to outbound XML messages that use publish channels.

The following description describes the processing for inbound messages using the Sync operation. The external system must evaluate the action code that is provided with outbound messages and determine the processing that is appropriate for that external application.

Action attributes can apply to the content of the parent object and child objects in an object structure. An action attribute applied to the parent object specifies the overall processing action for parent and child records. Applied to a child object, an action indicates processing that is specific to that record. An attribute provided for a child object is evaluated only when the primary object has an action value of Change. When the action for the primary object is not Change, actions on the child object are ignored.

Business rules take precedence over action attributes. If a business rule prohibits an action that is specified on an inbound XML message, an error occurs. For example, an inbound transaction that attempts to update a closed purchase order generates an error.

If an XML message contains multiple instances of an object structure, each instance of the object structure can specify a different action attribute. In the following example, the COMPANIES record has multiple child COMPCONTACT records, and each instance has its own action attribute.

```
<MXVENDOR>
  <COMPANIES action="Change">
    <COMPANY>TEST4
      <NAME>test</NAME>
      <ADDRESS1>100 Main Str</ADDRESS1>
      <COMPCONTACT action="Add">
        <NAME>SMITH</NAME>
        <TITLE>MANAGER</TITLE>
      </COMPCONTACT>
      <COMPCONTACT action="Change">
        <NAME>JONES</NAME>
        <TITLE>ENGINEER</TITLE>
      </COMPCONTACT>
    </COMPANY>
  </COMPANIES>
</MXVENDOR>
```

The action attribute can have the following values which are case-sensitive:

- Add
- Delete
- Change
- Replace
- AddChange
- Null

Value	Description
Add	Add records to the database in the receiving system.
Delete	Delete records from the database in the receiving system.
Change	Update existing records in the database in the receiving system.

Value	Description
Replace	Add records or replace records in the receiving system, depending on whether the primary record exists in the database.
AddChange	Add or update existing records in the database in the receiving system.
Null	Add records or replace records in the receiving system, depending on whether the primary record exists in the database.

Default action processing

When an inbound XML messages does not contain an action attribute, the message is processed as follows:

- If the primary record does not exist in the database, the Add action is applied.
- If the primary record exists in the database, the Replace action is applied.

For a message sourced from a flat file or an interface table, you can provide an action code for the primary object only. There is no support for providing action codes for child objects.

Add action

The Add action adds records to the database in the receiving system.

For inbound transactions, an error occurs if the data already exists. If the Add action is set on a parent object, the action extends to child objects and it is not necessary to specify the action at the child object level. Outbound transactions contain an Add action when the insert of an object generates the transaction.

Delete action

The Delete action deletes records from the database in the receiving system.

If the Delete action is set on a parent object, the action extends to child objects and it is not necessary to specify the action at the child object level.

If the Delete action is set on the parent object in an outbound XML message that does not include the child objects, the receiving system is responsible for identifying and deleting child objects.

If the Delete action is set on the parent object in an inbound XML message, the integration framework deletes the related child objects. If the parent object does not exist in the database, no error is reported to the sending system.

Change action

The Change action updates existing records in the database in the receiving system.

A Change action on the primary object in an XML message indicates that the message contains one or more parent or child records that are added, changed, or deleted. The message always contains the parent of any child record to be updated, even if the parent is unchanged.

The Change action on the primary object is the only case where you can supply an action for a child object. When the primary object in an inbound or outbound XML message contains a Change action, each child object in the message can contain one of the following actions.

Action attribute of child object	Processing action
Add	Add the child record. If the child record already exists, an error is generated.
Delete	Delete the child record. If the child record does not exist, an error is generated.
Change	Update the child record. If the child record already exists, an error is generated.
Null or no action specified	If the child record exists, update it. If the child record does not exist, add it.

Replace action

The Replace action updates existing records in the database in the receiving system.

A Replace action on the primary object in an XML message indicates that the message contains a complete set of objects that represents the result of additions, changes, and deletions to the object structure. These objects replace the existing database records and any record that is not referenced in the XML message is deleted.

For outbound processing, the Replace action is always used, not the Change action.

For inbound processing, any existing child record that is not explicitly mentioned in the message is deleted. External systems must also delete any child records that are not included in the XML message.

The Replace action can apply only to the primary object in an XML message. If a child record in an inbound XML message contains a Replace action when the primary object contains a Change action, the message is not processed. If a child level record contains a Replace action when the parent contains any action other than Change, the action on the child record is ignored.

AddChange action

The AddChange action adds or updates existing records in the database in the receiving system. The AddChange action is like the Replace action except that the AddChange action does not apply to child objects.

An AddChange action on the primary object adds the primary record and all the sub-records that are specfied in the message, if the primary record does not exist in the database. If the primary record does exist, it is updated and any child record that is included in the message. Existing child records that are not specified in the inbound message are not deleted.

The AddChange action is useful when an object structure includes elements that do not exist in the external system. For example, an external system can maintain vendor information but contact information is stored only in the database. An inbound message with a vendor record that has an action value of NULL deletes

the contact information in the database. If the action is set to AddChange for this transaction, the vendor information is updated and the contact information remains the same.

Comparison of Change, Replace, and AddChange actions

The Change, Replace, and AddChange actions differ in the information that they include in the XML message and the processing that they require of the receiving system.

Action attribute combinations

The following table shows the combinations of action attributes you can include on primary and child records.

Child Record	Add	Delete	Change	Replace	AddChange	No Value	Remarks
Primary Record							
Add	N/A	N/A	N/A	N/A	N/A	N/A	All child values ignored
Delete	N/A	N/A	N/A	N/A	N/A	N/A	All child values ignored
Change	Yes	Yes	Yes	No	No	Yes (insert, update)	Replace and AddChange not allowed at child level
Replace	N/A	N/A	N/A	N/A	N/A	N/A	All child values ignored
AddChange	N/A	N/A	N/A	N/A	N/A	N/A	All child values ignored
No value	N/A	N/A	N/A	N/A	N/A	N/A	All child values ignored

Related reference:

"Object elements and attributes" on page 348
An object element contains elements for the object fields. Each object element can contain one or more attributes.

Field value types:

The integration framework specifies the processing behavior for the different data types used to format field values.

Boolean columns

In inbound transactions, an element that represents a Boolean field must contain a value of 0 (false) or 1 (true). If the element does not contain a 0 or a 1, an error is generated. If the XML does not include an element for a Boolean field, the

corresponding database value is updated with the default value (0 or 1) that is defined for that column.

Encrypted fields

In inbound transactions, the attribute that represents the maxencrypted field must contain a value of 0 (false) or 1 (true). When the attribute value is 1, a decryption process is applied to the received data. When the attribute value is 0, the received data is not decrypted.

If the attribute does not contain a 0 or a 1, the received data is not decrypted. If the XML field does not include an element for an encrypted field, the received data is not decrypted.

Character encoding

The integration XML uses UTF-8 encoding. If an inbound transaction specifies any other encoding, the entire message must use that encoding. If an error is encountered during the processing of an inbound transaction that uses encoding other than UTF-8, the entire error XML that is written is encoded as UTF-8.

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

Date format

The integration XML supports the following ISO 8601 date format:

```
2011-04-06T10:11:58-05:00
```

Null columns

If an element in an inbound transaction contains no value, the corresponding database column is updated with a null value. If the XML does not include an element for a particular field, that field is not updated in the database.

For inbound data that is sourced from a flat file or an interface table, you can set a field to null by providing ~NULL~ for the field value in the flat file or interface table. Integration processing creates an empty tag in the corresponding XML message for any ~NULL~ values that are provided. This feature does not support numeric or date fields from an interface table source.

Number format

Regardless of the locale setting of the application server or the database, all decimal fields must use a period (.) as the decimal placeholder. Do not format numbers to the left of the placeholder. This format applies to inbound and outbound data.

$1,738,593.64 must be in the following format: 1738593.64

Integration XML schemas

XML schemas for channels and services, excluding standard services, are based on the configuration of the objects that are included in the object structure. You can generate schemas in the integration applications.

XML schemas overview:

An integration framework schema includes a predefined schema component for metadata as well as schema components based on configured objects, object structures, and standard services

Schemas can be generated to support XML-based integration scenarios where the following components are used:

- Object structure service
- Publish channel
- Invocation channel
- Enterprise service
- Standard service

Key fields

XML schema annotation identifies the key fields for a service or channel from the object structure definition and the data dictionary definitions for the corresponding object.

In the following example, the ITEMNUM element is identified as a key field in the schema for the MXITEM object structure.

```
<xsd:element name="ITEMNUM" minOccurs="0" type="MXString">
  <xsd:annotation>
    <xsd:documentation>ITEMNUM is a key field</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

XML validation

Inbound and outbound XML transactions are not validated against the corresponding XML schema. Integration business rules apply to inbound data regardless of schema validation. For object structure and enterprise services, the integration must comply with the schema format at the point where the message is processed by the object structure layer.

Namespace property

You can change the designation of the XML namespace by updating the `mxe.int.xmlnamespace` property in the System Properties application. The default namespace property value is `http://www.ibm.com/maximo` If you change the namespace property, the `MXMeta.xsd` file is regenerated. This file contains the metadata schema information that is used to build all other schemas. The integration framework validates the namespace provided with inbound XML messages.

Schema generation

Whenever you change an object structure or the data dictionary, you must regenerate the affected schemas. You must also regenerate the schemas after the following changes:

- Making a database field required or optional
- Changing the data type of a database field
- Adding or removing fields from an object structure

- Changing the structure of an object structure

Schemas are automatically generated when you deploy a web service.

You can manually generate schemas by selecting a record and selecting the Generate Schema/View XML action in the following applications:

- Enterprise Services
- Invocation Channels
- Object Structures
- Publish Channels
- Web Services Library

To access a generated schema, you can use a URL.

Enterprise service and object structure schemas

Use a URL that is similar to the following example in which MXSR is the object structure name that is associated with the service:

`http://localhost:port/meaweb/schema/service/MXSR`

For compatibility with version 7.5, URLs that are similar to the following example are still supported:

`http://localhost:port/meaweb/schema/service/MXSRService.xsd`

Standard service schemas

Use a URL that is similar to the following example in which ITEM is the name of the service.

`http://localhost:port/meaweb/schema/service/ss/ITEM`

For compatibility with version 7.5, URLs that are similar to the following example are still supported:

`http://localhost:port/meaweb/schema/service/ss/ITEMService.xsd`

Schema structures:

Schema structures include the predefined Metadata schema component and components for objects, object structures, services and standard services, all of which are generated at the time of schema creation

Metadata schema:

The `MXMeta.xsd` file contains the metadata information that is used to build all other schemas. Do not modify the `MXMeta.xsd` file because changes can result in incorrect schemas and issues with web services. The system regenerates this schema when someone changes the Namespace property.

In addition to data content, the metadata schema file includes the following additional information about messages:

- Attribute groups
- Content types
- Query data types
- Supporting data types

Attribute groups

The following table lists attributes by group.

Group	Attributes	Applies to
CommonContentGroup	• baseLanguage • creationDateTime • maximoversion • messageID • transLanguage	The root element of all input and output schema types.
ObjectStructurePropertyGroup	• action • relationship • deleteForInsert	The object element of all input and output schema types.
PublishingContentGroup	• event	The root element of all output schema types.
QueryContentGroup	• maxItems • rsStart • uniqueResult	The root element of all query input schema types.
ResponseContentGroup	• rsCount • rsStart • rsTotal	The root element of all response output schema types.

Content types

The following table lists the content types in the metadata schema.

Type	Description	Attributes
MaximoVersionType	Concatenated string that identifies: • major version • minor version • build • database build Identifies software version	
MXBooleanType	Extension of integer	changed
MXDateTimeType	Extension of dateTime	changed
MXDomainType	Extension of string; identifies the corresponding Maxvalue for a domain value.	changed maxvalue
MXDoubleType	Extension of double	changed
MXFloatType	Extension of float	changed
MXGLAccountType	Complex type with 2 values, VALUE and GLCOMP; identifies individual GL components and their sequential order for an account.	changed
MXGLComponentType	Extension of string; identifies GL component sequential order within the chart of accounts structure.	glorder

Type	Description	Attributes
MXIntType	Extension of integer	changed
MLLangStringType	Extension of MXString	changed languageEnabled
MXLongType	Extension of long	changed
MXStringType	Extension of string	changed

Query types

The following table lists the query types in the metadata schema.

Type	Description	Attribute
MXBooleanQueryType	Extension of integer	operator
MXDateTimeQueryType	Extension of dateTime	operator
MXDomainQueryType	Extension of string	operator maxvalue
MXDoubleQueryType	Extension of double	operator
MXFloatQueryType	Extension of float	operator
MXGLAccountQueryType	Complex type with a value of VALUE	operator
MXGLComponentQueryType	Extension of string	operator
MXIntQueryType	Extension of integer	operator
MXLongQueryType	Extension of long	operator
MXStringQueryType	Extension of string	operator

Data types

The following table lists the data types in the metadata schema.

Type	Description	Restricted Values
BooleanType	Indicates whether the result of a logical test is true or false.	• 0 (false) • 1 (true)
ChangeIndicatorType	Indicates whether a field has a new value. Applies only to object structures generated by an event.	1 (true)
EventType	Indicates whether a published object structure is the result of an event. If the value is 1, the published structure is the result of an event. EventType is an extension of BooleanType.	• 0 (false) • 1 (true)

ProcessingActionType	Processing actions that the integration services support.	• Add • Change • Replace • Delete • AddChange
QueryOperatorType	Identifies the Query by Example action to be performed on the corresponding field.	• = • != • &lt • <= • &gt • >= • SW • EW

Object structure schemas:

Object structure schemas define the content of an object structure. Each object structure has its own schema that includes all of the persistent and nonpersistent fields that are defined for each object in the structure. Object structure schemas are used to define input and output types and are not used directly as input or output for any service.

Schema generation

When you generate the following components, the object structure schemas are regenerated:

- The object structure schema by using an action in the Object Structures application
- The enterprise service schema, where the object structure is referenced by the service
- The publish channel schema, where the object structure is referenced by the channel

To access a schema, use a URL that is similar to the following example in which MXPERSON is the object structure name.

`http://localhost:port/meaweb/schema/service/MXPERSON`

For compatibility with version 7.5, URLs that are similar to the following example are still supported:

`http://localhost:port/meaweb/schema/service/MXPERSONService.xsd`

Schema object structure content

An object structure schema has the following elements:

- Object
- Object set
- Object query

For example, the schema for the MXPERSON object structure has the following elements:

Element	Element name	Type
Object	MXPERSON	MXPERSONType
Object set	MXPERSONSet	MXPERSONSetType
Object query	MXPERSONQuery	MXPERSONQueryType

Schema object content

The object element has the following content:

- The MXPERSON element has a type of MXPERSONType.
- The MXPERSONType is a complex type and has the PERSON element, which has a type of MXPERSON_PERSONType.
- The complex type MXPERSON_PERSONType has elements for all the configured attributes of the PERSON object and elements for the child objects in the object structure.
- Additional objects in the schema have a corresponding complex type, such as the MXPERSON_PERSONType, that defines the PERSON object.

Element	Type
PERSON (primary object)	MXPERSON_PERSONType
PHONE (child object)	MXPERSON_PHONEType
EMAIL (child object)	MXPERSON_EMAILType
SMS (child object)	MXPERSON_SMSType

The following example shows the structure of the corresponding XML:

```
<MXPERSON>
  <PERSON>
    <PHONE>
    </PHONE>
.
.
.
    <EMAIL>
    </EMAIL>
.
.
.
    <SMS>
    </SMS>
.
.
.
  </PERSON>
</MXPERSON>
```

Object set content

For the MXPERSON example, the MXPERSONSet element replaces the MXPERSON element and MXPERSONSetType replaces the complex type MXPERSONType. Everything else remains the same. The set element acts as a wrapper for multiple occurrences of the primary object (MXPERSON) and its child objects.

Object query content

You can use query elements only within the context of a service level schema.

The following content format is in the object query element:

- The MXPERSONQuery element is of type MXPERSONQueryType.
- The MXPERSONQueryType is a complex type and has elements for all the configured attributes of the primary object (PERSON) of the object structure.

The object query and the object set differ in the following ways:

- The query element includes only the primary object of the object structure.
- The query element does not include nonpersistent columns.
- The query element can include two occurrences of the elements to support a query on a range, such as a date range.

Because the query includes only the top object of the structure, it is not possible, for example, to query for a person by phone number. The phone number exists in the child PHONE object.

Object schemas:

Object schemas define the content of objects. Each object has a distinct schema, which includes all the persistent fields for a persistent object and all nonpersistent fields defined for a nonpersistent object. Object schemas are used to define input and output types and are not used directly as input or output for any service.

File names and location

The naming convention for object schemas is the object name, for example, `PERSON.xsd` and the files are located in the `schema/common/mbo` directory.

Schema generation

Object schema files are generated when you generate the following components:

- An object structure schema that contains the object
- An object structure service or enterprise service schema that references an object structure that includes the object
- A standard service schema that contains the object

All object schemas include the `MXMeta.xsd` schema file.

Schema content

The schema has the following elements (using MXPERSON as an example)

Element	Type	Comments
PERSONMbo	PERSONMboType	This type contains one instance of the PERSONMbo element.
PERSONMboSet	PERSONMboSetType	This type contains multiple instances of the PERSONMbo element.

Element	Type	Comments
PERSONMboKey	PERSONMboKeyType	This type contains a single instance of the PERSON element that is PERSONKeyType. The PERSONKeyType contains the attribute that is the primary key of the PERSON object, PERSONID. If multiple attributes make up the primary key of an object, the *object*KeyType contains the attributes that make up the primary key. This element is included in the response to a Create operation.
PERSONMboKeySet	PERSONMboKeySetType	This type contains multiple instances of the PERSON element that is PERSONKeyType. The PERSONKeyType contains the attribute that is the primary key of the PERSON object, PERSONID. If multiple attributes make up the primary key of an object, the *object*KeyType contains the attributes that make up the primary key. This element is included in the response of a Create operation.
PERSONMboQuery	PERSONMboQueryType	This type contains two instances of the PERSON element. Two instances allow a query to specify a range, for example, From Date and To Date.

Multi-noun messages:

An XML message can contain multiple nouns.

```
<MXPERSON>
  <MXPERSONSET>
    <PERSON>
    .
    .
    <SMS>
    </SMS>
    </PERSON>
    <PERSON>
    .
    .
    <SMS>
```

```
    </SMS>
    </PERSON>
  </MXPERSONSET>
</MXPERSON>
```

Service level schemas:

Service level schemas apply to enterprise services, object structure services, and standard services. The same schema describes enterprise services and object structure services. A different schema describes standard services.

Enterprise services, object structure services, and standard services use objects and object structures as input and output for the operations they support. Multiple services can perform the same operation using the same input and output.

Predefined input and output for publish channels and invocation channels are used to implement these channels to invoke external services or to map to other output formats.

File names and location

The naming convention for object structure schemas and enterprise services schemas is *application service name* + Service, for example, `PERSONService.xsd` where PERSON is the application service name.

The naming convention for service level schemas is *object structure* + Service, for example, `MXPERSONService.xsd`.

The schema files are located in the `/schema/service` directory.

The schema files are located in the `<root>/schema/common/service` directory.

Schema generation

Predefined service-level schema files are not provided. When you deploy a web service, a service-level schema file is generated for the associated object structure, if one does not exist. You can also generate a service-level schema by using an action in the Web Services Library application

All service level schemas include the metadata schema file and the applicable object structure and object schema files.

Schema content

One schema file is generated for each object structure with multiple data types within each file. Each data type corresponds to each input and output operation that can be deployed or processed as a service. Different services reuse the data types in these schemas. No one service uses all the data types in a single schema.

Service-level schemas contain the following types, using the MXPERSON example:

- CreateMXPERSON
- CreateMXPERSONResponse
- DeleteMXPERSON
- InvokeMXPERSON
- InvokeMXPERSONResponse

- PublishMXPERSON
- QueryMXPERSON
- QueryMXPERSONResponse
- SyncMXPERSON
- UpdateMXPERSON

Standard service schemas input and output:

Standard services are provided by applications to perform specific operations on objects and can expose multiple methods as web services. Standard services are available only for methods that are properly annotated within the service.

The service-level schemas that are generated for standard services are used only by the corresponding actions.

Object structure and enterprise service input and output:

The object structure contains the content for both object structure service and enterprise service schemas. An object structure service supports all defined operations but an enterprise service supports just one specific operation. Multiple enterprise services must be created to support multiple operations even when they reference the same object structure and use the same schema.

Object structure and enterprise service schemas identify the following:
- The input and output types for object structure and enterprise services
- The input and output of an invocation channel
- The output provided by a publish channel

All operations support an output type:
- The output for the Create, Update, Delete and Sync operations are the primary object keys, the internal ID field and autokey values.
- The output for a Query operation is a result set of object structures.

The following examples, based on the MXPERSON object structure, describe the content of object structure and enterprise services,

CreateMXPERSON element

The CreateMXPerson element has the following definitions:
- The CreateMXPERSON element is type CreateMXPERSONType.
- The CreateMXPERSONType has element MXPERSONSet, which is type MXPERSONSetType. MXPERSONSet is derived from the object structure schema.
- The MXPERSONSetType has elements for all the configured attributes of the PERSON object and elements for the child objects (PHONE, EMAIL, and SMS) that are defined in the object structure.

The definitions of the following elements and types are comparable to the CreateMXPERSON element and type:

Element	Type
UpdateMXPERSON	UpdateMXPERSONType

Element	Type
SyncMXPERSON	SyncMXPERSONType
InvokeMXPERSON	InvokeMXPERSONType
InvokeMXPERSONResponse	InvokeMXPERSONResponseType

Create MXPERSONResponse element

The CreateMXPERSONResponse element has the following definitions:

- The CreateMXPERSONResponse element is type CreateMXPERSONResponseType.
- The CreateMXPERSONResponseType has element PERSONMboKeySet, which is type PERSONMboKeySetType.
- The PERSONMboKeySetType has element PERSON, which is type PERSONKeyType.
- The PERSONKeyType contains only the PERSONID attribute of the PERSON object, which is the primary key of the PERSON object.

PublishMXPERSON element

The PublishMXPERSON element has the following definitions:

- The PublishMXPERSON element is type PublishMXPERSONType.
- The PublishMXPERSONType has element MXPERSONSet, which is type MXPERSONSetType. MXPERSONSet is derived from the object structure schema.
- MXPERSONSetType has elements for all the configured attributes of the PERSON object and for the child objects ((PHONE, EMAIL, and SMS) defined in the object structure.

The Publish element is like the Create element but uses an additional attribute, event, that is defined in the PublishingContentGroup in the metadata schema.

DeleteMXPERSON element

The DeleteMXPERSON element has the following definitions:

- The DeleteMXPERSON element is type DeleteMXPERSONType.
- The DeleteMXPERSONType has element MXPERSONDelete, which is type MXPERSONDeleteType.
- The MXPERSONDeleteType has elements for all the configured attributes of the PERSON object. Delete supports only the top (primary) object of the object structure.

QueryMXPERSON element

The QueryMXPERSON element has the following definitions:

- The QueryMXPERSON element is type QueryMXPERSONType.
- The QueryMXPERSONType has element MXPERSONQuery, which is type MXPERSONQueryType.
- The MXPERSONQueryType has elements for all the configured attributes of the PERSON object. Query supports only querying against the top (primary) object of the object structure.

The Query element uses the additional attributes uniqueResults, maxItems, and rsStart, which are defined in the QueryContentGroup in the metadata schema.

QueryMXPERSONResponse element

The QueryMXPERSONResponse element has the following definitions:

- The QueryMXPERSONResponse element is type QueryMXPERSONResponseType.
- The QueryMXPERSONResponseType has element MXPERSONSet, which is type MXPERSONSetType.
- The MXPERSONSetType has elements for all the configured attributes of the PERSON object and elements for child objects defined in the object structure (PHONE, EMAIL, and SMS).
- Although the QueryMXPERSON element restricts queries to the top object of the object structure (PERSON), the response can contain child objects (PHONE, EMAIL, and SMS).

The Query Response element uses the additional attributes rsStart, rsCount, and Total, which are defined in the ResponseContentGroup in the metadata schema.

Collaboration switches

Collaboration switches use a concept of ownership to help users manage the synchronization of inbound data between the integration framework and an external system. The switches provide the ability to control specific subprocesses within an application, based on the ownership of different data objects within a transaction.

The primary object of the object structure for most master data and document integration objects include an OWNERSYSID attribute. By default, inbound integration processing does not specify any value in the OWNERSYSID field and enterprise services are processed in a standard manner.

Format of collaboration switches

Collaboration switches provide a flexible, user-defined way to control the processing of some inbound transactions through bypassing the default processing for certain types of transactions. Collaboration switches are located in the MXCOLLAB table.

Switch elements

Each collaboration switch contains four elements, three of which combine to create a unique key. The following table lists these elements and the elements that comprise the unique key are marked with an asterisk (*).

Element	Corresponding MXCOLLAB field
Process control ID*	PCID
System ID 1*	OWNER1SYSID
System ID 2*	OWNER2SYSID
Process control value	PCVALUE

Process control ID

The process control ID identifies a business process in an application, such as the validation of an invoice match, the creation of a blanket PO release, and the update of a physical inventory count.

The following table shows the prefix of the process control ID and indicates the application to which it applies.

Prefix of Process Control ID	Corresponding Application
INV	Invoice
ITM	Item
IV	Inventory
LT	Labor
PO	Purchase order
PR	Purchase requisition
WO	Work order

For example, the IVRC, IVRCY, and IVWO collaboration switches are all related to inventory processing.

System ID values

System ID 1 and System ID 2 identify the internal and external systems. The values in these fields vary, depending on the transaction and the objects in the transaction. In general, System ID 1 identifies the system that creates an object, and System ID 2 identifies the system that creates the record that is being referenced or updated.

Process control value

The process control value specifies whether the business components bypass default processing for the type of transaction indicated by the process control ID, System ID 1, and the System ID 2. The process control value can be 0 (false) or 1 (true) and can have the following meanings:

Value of process control	Description
0	Performs default processing
1	Bypasses default processing

Retrieving a collaboration switch

Each process control ID is associated with at least three collaboration switches, the default switches and any switches that the user adds. The integration framework provides the logic that determines which system ID values to set when retrieving a collaboration switch from the MXCOLLAB table.

Procedure

1. The integration framework determines the values for System ID 1 and System ID 2 based on the process control ID in an inbound transaction. For example, if the process control ID is PRDEL, System ID 1 is THISMX, and System ID 2 is the system that owns the PR.

2. If the value in System ID 1 is blank, null, or equal to the value in the MXSYSID row of the MAXVARS table, System ID 1 is set to THISMX .
3. If the value in System ID 2 is blank, null, or is equal to the value in the MXSYSID row of the MAXVARS table, and the process control ID is not PRPAB, System ID 2 is set to THISMX.
4. If the process control ID is PRPAB, the value in System ID 2 is set to null after step 1, and System ID 2 is set to EXT (if a blanket PO does not exist).
5. If both System ID 1 and System ID 2 are now set to THISMX, default processing is used and the remaining logic is not applied.
6. The MXCOLLAB table is checked, to identify if it contains a record with the key. If the record exists, the process control value for the record indicates whether to use or bypass standard processing and the remaining logic is not applied.
7. If a matching record does not exist in the database, the key is modified in the following manner:
   - If System ID 1 now equals THISMX and System ID 2 does not equal THISMX, EXT is set as the value for System ID 2.
   - If System ID 1 value does not equal THISMX and System ID 2 equals THISMX, EXT is set as the value for System ID 1.
8. The MXCOLLAB table is checked again, to identify if it contains a record with the modified key. If the record exists, the process control value for the record indicates whether to use or bypass standard processing and the remaining logic is not applied.
9. EXT is set as the value for both System ID 1 and System ID 2.
10. The record with the modified key is retrieved from the MXCOLLAB table. This record always exists, because every process control value has a default collaboration switch with both system IDs set to EXT.
11. The process control value for the record indicates whether to use or bypass standard processing.

Configuring collaboration switches

You can use predefined collaboration switches and you can configure new ones if required.

Viewing collaboration switches:

Use any database tool to execute a SQL query to see the values in the MXCOLLAB table, or generate a report to view its content.

Procedure

1. To view the collaboration switches for a single process control ID, use the following SQL query:

```
select   pcid, owner1sysid, owner2sysid, pcvalue
from     mxcollab
where    pcid = 'PCID'
order by pcid, owner1sysid, owner2sysid;
```

2. To view all collaboration switches, use the following SQL query:

```
select   pcid, owner1sysid, owner2sysid, pcvalue
from     mxcollab
order by pcid, owner1sysid, owner2sysid;
```

3. To view a short description of the process control IDs, use the following SQL query:

```
select * from mxcollabref order by pcid;
```

Modifying a collaboration switch:

Authorized users can modify the process control value of a collaboration switch. Do not change the value of PCID, OWNER1SYSID, or OWNER2SYSID values on existing collaboration switches.

About this task

The values in the MXCOLLAB table are case-sensitive.

Procedure

1. Use a database tool to connect to the Maximo database.
2. Use the following SQL statement to change the process control value in a collaboration switch :

```
update mxcollab
set pcvalue     =  PCVALUE
where pcid      = 'PCID'
and owner1sysid = 'OWNER1SYSID'
and owner2sysid = 'OWNER2SYSID';
```

3. Restart the application server to refresh the cached values for collaboration switches.

Adding a collaboration switch to the database:

Authorized users can add collaboration switches to the MXCOLLAB table. New switches must use an existing process control ID, but they can use new system IDs.

About this task

Only the default collaboration switches can use the values THISMX and EXT in the system ID fields.

Procedure

1. Use a database tool to connect to the Maximo database.
2. Use the following SQL statement to add a collaboration switch:

```
insert into mxcollab
(pcid, owner1sysid, owner2sysid, pcvalue)
values ('PCID', 'OWNER1SYSID', 'OWNER2SYSID', PCVALUE);
```

Example

For example, you can configure the integration framework to integrate with an Oracle Financials system and other applications. When Oracle Financials issues system-owned inventory, the integration framework accepts the transaction and updates inventory balances and costs in the Maximo database. When other applications issue system-owned inventory, the integration framework accepts the transaction, but does not update inventory balances or costs in the Maximo database.

Before you modify the MXCOLLAB table to reflect these conditions, the INV collaboration switches have the following values:

Process Control ID	System ID 1	System ID 2	Process Control Value
INV	THISMX	EXT	1

Process Control ID	System ID 1	System ID 2	Process Control Value
INV	EXT	THISMX	0
INV	EXT	EXT	1

To modify the MXCOLLAB table, to support this scenario:

1. To update the collaboration switch, use the following SQL statement:

```
update mxcollab
set pcvalue     =  1
where pcid      = 'INV'
and owner1sysid = 'EXT'
and owner2sysid = 'THISMX';
```

   This statement changes the value of the INV/ EXT/ THISMX collaboration switches to 1 to bypass normal update processing.

2. Use the following SQL statement to add a new collaboration switch for transactions from the Oracle Financials system:

```
insert into mxcollab
(pcid, owner1sysid, owner2sysid, pcvalue)
values ('INV', 'ORC', 'THISMX', 0);
```

   The new collaboration switch sets ORC as the system ID, and sets the process control value to 0 so that normal processing is applied to issue transactions received from Oracle Financials. If the OWNERSYSID is blank, the value in the DEFEXTSYS integration control is used.

After you perform this procedure, the INV collaboration switches have the following values.

Process Control ID	System ID 1	System ID 2	Process Control Value
INV	THISMX	EXT	1
INV	EXT	THISMX	1
INV	EXT	EXT	1
INV	ORC	THISMX	0

For example, when you set the value of the ISUIN integration control to 1, issue transactions are accepted from an external system. The INV collaboration switch controls the update of inventory balance and cost related to issues. You can adjust the setting of this switch, if necessary, to bypass that update process.

The INV/EXT/THISMX collaboration switch controls the processing of inventory (where the Process Control ID = INV) that is issued in the external system (System ID 1 = EXT) and owned by the system (System ID 2 = THISMX).

If the value of the INV/EXT/THISMX collaboration switch is 0, default processing applies and inventory balance and cost values are updated.

If the value of the INV/EXT/THISMX collaboration switch is 1, default processing is bypassed and inventory balance and cost values are not updated.

In the example, ISUIN accepts any issues into the system. The INV/EXT/THISMX collaboration switch determines how the inventory business component processes a specific type of issue.

Predefined collaboration switches

You can use the default collaboration switches and the predefined collaboration switches that are provided.

Predefined collaboration switches are available for the following applications:

- Inventory management
- Invoicing
- Labor transactions
- Purchase orders and purchase requisitions
- Receipts
- Work orders

A detailed description of predefined collaboration switches is available in the product documentation on IBM Knowledge Center.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample

programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user’s name, user name, password, or other personally identifiable information for purposes of session management, authentication, single sign-on configuration or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also likely eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies” and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Printed in USA