Integration Guide
Note

Before using this information and the product it supports, read the information in “Notices” on page 307.

This edition applies to version 7, release 1, modification 0 of IBM Maximo Asset Management, IBM Tivoli Asset Management for IT, and IBM Tivoli Service Request Manager, and version 7, release 1, modification 1 of IBM Tivoli Change and Configuration Management Database and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2007, 2008. All rights reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

## Section I: Using the Integration Framework

### Chapter 1: What is the Integration Framework?

- Intended Audience ........................................... xiii

### Chapter 2: Integration Framework Architecture

- Integration Framework Overview .......................... 6
- Integration Framework for Data Exchange ................. 8
  - Object Structures ........................................ 8
  - Publish Channels ........................................ 9
  - Invocation Channels ..................................... 10
  - Enterprise Services ..................................... 11
  - External Systems ........................................ 11
  - Web Services ............................................ 12
  - Data Import and Data Export ............................ 13
    - Content .................................................. 13
- Integration Framework for Operational Management Product Integration ........................................ 15
  - Process Management Products .......................... 15
    - Action ................................................... 14
    - Logical Management Operations ...................... 14
    - Integration Modules .................................. 14
    - Operational Management Products .................. 15
- Integration Framework for User Interface Integration ....................................................... 16
  - Launch Entries .......................................... 16
  - Land in Context .......................................... 16

### Chapter 3: Inbound Integration Processing

- Asynchronous Inbound Integration Processing .......... 19
  - Enterprise Services ..................................... 20
    - Inbound Integration Process Initiation ............... 21
    - Messages in the Inbound Queues ...................... 21
    - Inbound Queue Message Retrieval .................... 24
    - Object Structure Identification ...................... 24
    - User Exit Preprocessing ............................... 25
    - Enterprise Service Class Processing ............... 26
    - User Exit Postprocessing ............................. 27
    - XSL Map ................................................ 27
    - Object Structure Multiplication ..................... 28
    - Object Structure Processing Rules .................. 29
    - Object Creation ....................................... 29
    - Object Processing Rules ............................... 29
    - Object Structure Processing Class .................. 30
    - User Exit Object Processing ......................... 30
    - Object Processing ..................................... 31
- Synchronous Inbound Integration Processing ............ 32
  - Enterprise Services ..................................... 32
    - Inbound Integration Process Initiation ............... 32
Chapter 4: Outbound Integration Processing .............................................................. 47
  Asynchronous Outbound Integration Processing .............................................. 48
    Publish Channels ......................................................................................... 48
      Outbound Integration Process Initiation .................................................... 49
      Object Structure Identification .................................................................. 50
      Object Structure Processing Rules ............................................................ 50
      Object Structure Multiplication .................................................................. 51
      User Exit Preprocessing ............................................................................. 52
    Publish Channel Class Processing ............................................................... 52
      User Exit Postprocessing ........................................................................... 53
      XSL Map .................................................................................................... 54
      Send the Publish Channel Data to the External System ............................ 54
  Synchronous Outbound Integration Processing .............................................. 55
    Invocation Channels. ..................................................................................... 55
      Outbound Integration Process Initiation .................................................... 56
      Request Object Structure Definition ....................................................... 56
      Request Class Processing .......................................................................... 57
      Request User Exit Preprocessing ............................................................... 57
      Request XSL Map ...................................................................................... 58
      Send the Invocation Channel Data to the Endpoint .................................... 58
      Response Class Processing ....................................................................... 59
      Response User Exit Preprocessing ............................................................ 59
      Response XSL Mapping ............................................................................. 60
      Object Processing ........................................................................................ 60

Chapter 5: Integration XML and Schemas ............................................................... 61
  Integration XML Structure ................................................................................ 62
  Integration XML Content ................................................................................ 62
    Root Element ............................................................................................... 63
  Object Structure Element ................................................................................. 67
    Key Fields .................................................................................................... 67
    Field Attributes ............................................................................................. 68
## Section II: Advanced Topics

### Chapter 10: Endpoints and Handlers

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints</td>
<td>144</td>
</tr>
<tr>
<td>Configuring Endpoints</td>
<td>144</td>
</tr>
<tr>
<td>Handlers</td>
<td>145</td>
</tr>
<tr>
<td>Enterprise Bean Handler</td>
<td>145</td>
</tr>
<tr>
<td>FLATFILE Handler</td>
<td>147</td>
</tr>
<tr>
<td>Flat File Naming Convention</td>
<td>147</td>
</tr>
<tr>
<td>Flat File Formatting</td>
<td>148</td>
</tr>
<tr>
<td>Flat File Properties</td>
<td>148</td>
</tr>
<tr>
<td>HTTP Handler</td>
<td>148</td>
</tr>
<tr>
<td>I FACETABLE Handler</td>
<td>150</td>
</tr>
<tr>
<td>JMS Handler</td>
<td>151</td>
</tr>
<tr>
<td>WEBSERVICE Handler</td>
<td>153</td>
</tr>
<tr>
<td>XMLFILE Handler</td>
<td>154</td>
</tr>
<tr>
<td>CMDLINE Handler</td>
<td>155</td>
</tr>
<tr>
<td>Writing Custom Handlers</td>
<td>156</td>
</tr>
</tbody>
</table>

### Chapter 11: Advanced Interface Table Polling

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cron Tasks</td>
<td>160</td>
</tr>
<tr>
<td>Configuring Cron Tasks</td>
<td>160</td>
</tr>
<tr>
<td>Selectors</td>
<td>160</td>
</tr>
<tr>
<td>Defining Selectors</td>
<td>160</td>
</tr>
<tr>
<td>Queue Tables</td>
<td>161</td>
</tr>
</tbody>
</table>

### Chapter 12: JMS Queue Configuration

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating and Configuring a Queue</td>
<td>164</td>
</tr>
<tr>
<td>Sequential Queues</td>
<td>165</td>
</tr>
<tr>
<td>Continuous Queue</td>
<td>165</td>
</tr>
<tr>
<td>Enabling Message Beans</td>
<td>166</td>
</tr>
<tr>
<td>Continuous Queue Performance</td>
<td>167</td>
</tr>
<tr>
<td>Configuring Message Beans</td>
<td>167</td>
</tr>
<tr>
<td>WebSphere Application Server</td>
<td>167</td>
</tr>
<tr>
<td>WebLogic Server</td>
<td>168</td>
</tr>
<tr>
<td>Message Caching</td>
<td>168</td>
</tr>
<tr>
<td>Continuous Queue Errors</td>
<td>169</td>
</tr>
<tr>
<td>WebSphere Application Server</td>
<td>169</td>
</tr>
<tr>
<td>WebLogic Server</td>
<td>170</td>
</tr>
<tr>
<td>Queue Message Format</td>
<td>171</td>
</tr>
<tr>
<td>Message Header</td>
<td>171</td>
</tr>
<tr>
<td>Message Properties</td>
<td>171</td>
</tr>
<tr>
<td>Message Body</td>
<td>172</td>
</tr>
<tr>
<td>Queue Selectors</td>
<td>172</td>
</tr>
<tr>
<td>Queue Utilities</td>
<td>173</td>
</tr>
<tr>
<td>IBM WebSphere MQ</td>
<td>174</td>
</tr>
<tr>
<td>Configuring JMS Endpoints and Handlers</td>
<td>174</td>
</tr>
<tr>
<td>Configuring the Integration Queues and WebSphere MQ Provider</td>
<td>175</td>
</tr>
</tbody>
</table>

### Chapter 13: Security

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Queue</td>
<td>178</td>
</tr>
<tr>
<td>Configuring J2EE Restrictions</td>
<td>178</td>
</tr>
<tr>
<td>Enterprise Bean Access</td>
<td>179</td>
</tr>
</tbody>
</table>
Default Collaboration Switches ................................................................. 290
Collaboration Switch Retrieval ................................................................. 291
View Collaboration Switches ..................................................................... 292
Collaboration Switch Modification ......................................................... 293
Collaboration Switch Additions ............................................................... 293
    Modifying the MXCOLLAB Table ......................................................... 294
Inventory Collaboration Switches ............................................................ 296
Invoice Collaboration Switches ............................................................... 298
Labor Transaction Collaboration Switches............................................... 300
Purchase Order Collaboration Switches.................................................. 300
Purchase Requisition Collaboration Switches .......................................... 302
Receipt Collaboration Switches ............................................................... 304
Work Order Collaboration Switches ........................................................ 305

Notices ........................................................................................................ 307

Index ........................................................................................................... 311
About This Publication

This guide describes the integration framework and how to configure, integrate, and create business flows between the system and your framework applications.

Intended Audience

The guide is intended for the following people:

- Developers
- Implementation analysts
- Support personnel
- System administrators
Using the Integration Framework

Chapter 1: What is the Integration Framework?
Chapter 2: Integration Framework Architecture
Chapter 3: Inbound Integration Processing
Chapter 4: Outbound Integration Processing
Chapter 5: Integration XML and Schemas
Chapter 6: Interface Tables
Chapter 7: Message Tracking
Chapter 8: Error Management
Chapter 9: Basic Configuration
What is the Integration Framework?

The integration framework is a set of applications that help you to integrate the system to your external applications. You also can create business flows between the system and your external applications.

The key features of the integration framework include:

> Predefined content to assist in implementing integration requirements in a timely manner. This content is a comprehensive set of outbound (channels) and inbound (services) integration components that are available to use immediately.

> Applications to configure, redefine, and to create new integration definitions.

> Applications to facilitate the customization of predefined content by using a processing rule engine, Java™, and Extensible Stylesheet Language Transformations (XSLT).

> Support for multiple communication modes, including:
  - Web services
  - HTTP
  - Java™ Message Service (JMS) messaging
  - Database interface tables
  - XML and flat files

> Event-based, batch, program-initiated, and user-initiated processing of outbound and inbound messages.

> Data load and performance scalability by using JMS queues.

> Support for clustered environments that reduce system downtime, increase system availability, and improve system performance.

> Support for user interface-based integration that includes context-based external application launches.

> Support for the integration to operational management products (OMPs).

> Support for the bulk export of data by using a user-defined SQL query.

> Support for the bulk imports of XML or flat files.
▼ Dynamic XML schema generation for all integration interfaces.

▼ Dynamic generation of Web Services Interoperability (WS-I) compliant Web services, including Web Service Definition Language (WSDL).

▼ Provides the concept of an adapter that is used to group related integration framework components. You can configure and deploy adapters for enterprise connectivity with various external systems. Each adapter can have its own interface and delivery mode. Predefined adapters for Oracle® and SAP® are available as add-ons.
Anyone involved in the implementation or day-to-day administration of the integration framework applications must be familiarized with the integration framework architecture. Familiarity with the architectural concepts is essential for using the application and implementing framework components.
Integration Framework Overview

The integration framework facilitates bidirectional data exchange between the system and external applications in real time or batch mode. Through the integration framework, you can exchange data synchronously and asynchronously by using a variety of communication protocols.

The integration framework also provides features that support the integration with operational management products, such as IBM® Tivoli® Provisioning Manager. You also can use a system application user interface to launch an external application user interface.

Through the integration framework, you can send and receive XML messages between the system and external applications. The integration framework provides the following capabilities:

- Build, transform, and customize message content
- Send and receive messages using multiple protocols, including:
  - Web services
  - Hypertext Transfer Protocol (HTTP)
  - Java Message Service (JMS)
- Exchange data synchronously and asynchronously
- Exchange event-based messages
- Import and export messages in batch mode

The following integration framework components support data integration:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object structures</td>
<td>Define message content.</td>
</tr>
<tr>
<td>Services</td>
<td>Receive data into the system.</td>
</tr>
<tr>
<td>Channels</td>
<td>Send data out of the system.</td>
</tr>
<tr>
<td>External systems</td>
<td>Define external applications and services that integrate with the system.</td>
</tr>
<tr>
<td>Endpoint</td>
<td>Modes that you use to communicate with external applications. Modes include Web services, HTTP, Enterprise JavaBean™ (EJB), and flat files.</td>
</tr>
<tr>
<td>Events</td>
<td>The business object events that you use to initiate data exchange. Events include data import, data export, and record status changes.</td>
</tr>
</tbody>
</table>
Operational management product integration facilitates the automation of information technology services, such as software deployment. A process management product calls an integration module which in turn communicates with the operational management product to perform a logical management operation.

With this framework, you can automate logical management operation actions, such as software deployment. The process management product initiates the integration module to invoke the operational management product to perform automated actions.

By using the integration framework, you can configure integration modules to support specific logical management operations and operational management products. You configure an endpoint and handler to identify the communication protocol (HTTP, Web service) that the integration module uses to invoke the operational management product.

The integration module can map the service response so that it is returned to the process management product. The service response then can be processed in multiple ways. The service can open a response in a user interface application, or save the response data to the application database.

You use the integration framework to integrate operational management products by using an assisted approach.

The following integration framework components use operational management product integration:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>Query message and receive data by the integration framework.</td>
</tr>
<tr>
<td>Content</td>
<td>System content that is configured to enable various integration components.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical management operations</td>
<td>An application that you use to define the actions that the integration module supports for an operational management product, such as software deployment.</td>
</tr>
<tr>
<td>Integration modules</td>
<td>An application that you use to define the configurations and the relationships to integration modules, logical management products, and operational management products.</td>
</tr>
</tbody>
</table>
The integration framework provides a mechanism for you to navigate from a system application user interface to an external application user interface. You can define the context to facilitate the navigation into the external application interface. The integration framework supports URL substitutions of any values of any system business object. For example, you can configure the system to display a specific part number in an external application.

You can use operational management product-specific features when you launch to an operational management product application user interface. Features include retrieving the registered host name of the operational management product and a configured item source token for the operational management product.

Through the Launch in Context application, you can navigate to any external application other than the operational management product. You also can navigate to a system user interface from an external application (Land in Context).

### Object Structures

An object structure is the common data layer that the integration framework uses for all outbound and inbound application data processing. An object structure consists of one or more business objects that make up the content of an XML message. You can use the message content of a single object structure to support inbound and outbound messages.

When you define multiple objects in the object structure, the objects must have a reference to a valid parent-child relationship within the system.

The object structure has a Java definition class that you can code to perform logic such as filtering for outbound messages. For inbound messages, you can use an object structure Java processing class to invoke specific business object logic that is beyond the normal integration framework insert, update, and delete processing.
The object structure is the building block of the integration framework that lets framework applications to perform the following functions:

- Publish and query application data
- Add, update, and delete application data
- Import and export application data

You also can use the object structure as a service to support inbound message processing. You can invoke the object structure service as a Web service, as an Enterprise JavaBean, or by using HTTP. The object structure service supports system data updates, and queries that are started outside of the system.

**Publish Channels**

A publish channel is the pipeline for sending data asynchronously from the system to an external system. Events that initiate publish channel processing are object events (insert, update, and delete), application-initiated calls, and data export.

The content of a publish channel XML message is based on the associated object structure. When you trigger publish channel processing, the integration framework builds the XML message based on the object structure. The system then moves the message through multiple processing layers before placing the message into a queue and releasing the initiator of the transaction.

The publish channel can use the following processing layers:

- Processing rules – The integration framework provides a rule engine where you can filter and transform the XML message. You can implement rules in the Publish Channel application.
- User exit – Represents a Java class that you can use to filter data, transform data, and implement business logic. You can use this class as part of an installation-customization.
- Data processing class – Represents a Java class that you can use to filter, transform data, and implement business logic. Adapters for Oracle and SAP provide processing classes to support integration to these products.
- XSL map – Represents an XSLT style sheet that you can use to transform data and perform mapping of the XML message to another format.

After the system places the message into the queue, a polling thread (the system cron task) picks up the message and sends it to an external system through a configured endpoint. The endpoint identifies the protocol that the system uses to send data, such as HTTP or Web services. The endpoint also identifies the property values that are specific to that endpoint, such as URL, user name, and password.
Invocation Channels

The Service Oriented Architecture (SOA) enables the use of external services to process data from multiple sources. Invocation channels support a generic service-oriented architecture capability by enabling the system to call an external service synchronously. The invocation channel returns the response of the service back to the caller for subsequent processing.

For example, you might want to use an external system to calculate the tax amount for a product that you want to purchase. You can configure an invocation channel to call the external tax service. The invocation channel then can save the value of the external tax amount in the system database.

The initiation of an invocation channel is implemented by using an action class, which then calls an invocation channel. You can implement an action by using the following means:

- A user interface control (within an application)
- Workflow routing
- Escalation

The system execution of an invocation channel is synchronous, and a response can be returned from the external service to the caller.

The content of an invocation channel data structure is based on the associated object structure. When the invocation channel processing starts, the integration framework builds the XML message based on the object structure. The message then moves through multiple processing layers before calling the external service.

The invocation channel can use the following processing layers:

- User exit – Represents a Java class that you can use to filter data, transform data, and implement business logic. You can use this class as part of an installation-customization.
- Data processing class – Represents a Java class that you can use to filter, transform data, and implement business logic. Adapters for Oracle and SAP provide processing classes to support integration to these products.
- XSL map – Represents an XSLT style sheet that you can use to transform data and perform mapping of the XML message to another format.

After the message goes through the processing layers, the integration framework uses the configured endpoint to call the external service. The endpoint identifies the protocol the system uses to send data, such as HTTP or a Web service. The endpoint also identifies the property values that are specific to that endpoint, such as URL, user name, and password.

When the message is delivered to the endpoint, the response of the service is returned to the invocation channel. The response message can use similar processing layers that are available on the response portion of the invocation channel: user exit, process class, and XSL map. The response XML is mapped back to the response object structure, which can be the same or different from the object structure that initiated the message. The data mapped in the object structure is returned to the caller of the channel for subsequent processing. The
invocation channel can be configured to ignore the response that is returned from the external service and return no data to the caller of the channel.

Enterprise Services

The enterprise service is a pipeline for querying system data and importing data into the system from an external system. You can configure enterprise services to process data synchronously (without a queue) or asynchronously (with a queue). Enterprise services can use multiple protocols, such as Web services and HTTP.

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

The enterprise service can use the following processing layers:

- **Processing rules** – The integration framework provides a rule engine where you can filter and transform the XML message.
- **User exit** – Represents a Java class that you can use to filter, transform data, and implement business logic. You can use this class as part of an installation-customization.
- **Data processing class** – Represents a Java class that you can use to filter, transform data, and implement business logic. Adapters for Oracle and SAP provide processing classes to support integration to these products.
- **XSL Map** – Represents an XSLT style sheet that you can use to transform data and perform mapping of the XML message to another format.

External Systems

Any business application that sends data to the system or receives data from the system is an external system. External systems are an integral part of enterprise service and publish channel processing. You use and configure enterprise services and publish channels to exchange data with one or more external systems.

Object structure services and invocation channels do not use external systems.

You can use the External Systems application to perform the following functions:

- **Name the external applications or systems that exchange data with the integration framework**
- **Specify the protocol that the integration framework uses to send data to the external system**
- **Identify the publish channels and enterprise services that each system implements**
- **Create interface tables**
To create an external system, you specify an endpoint, the queues, and the integration control values in the External System application.

You can also define the following properties on the external system:

- The endpoint that identifies how and where the integration framework exchanges data with the system
- The Java Message Service (JMS) queues that the system uses
- Whether the external system is ready to begin integration processing

**Web Services**

External applications, Enterprise Service Bus, and Business Process Execution Language processes can use Web services to query or send transactions to the integration framework.

The integration framework provides three types of services that you can deploy as a Web service:

- Object structure service
- Enterprise service
- Standard service

When you deploy Web services, the system generates a schema and Web Services Description Language (WSDL) file that you can access with a URL. Optionally, a Universal Description Discovery and Integration (UDDI) registry can be updated for each deployed service.

The integration framework supports the following Web services:

- **Object structure Web service** - Object structure Web services are created from an object structure and do not provide a processing layer which is available to enterprise services. An object structure Web service supports five operations: create, delete, query, sync, and update.

- **Enterprise Web service** - Enterprise Web services are created from an enterprise service and provide exit processing and optional JMS support. The integration framework creates individual enterprise Web services for the operation that is defined in an enterprise service (one operation per service).

  The operations that are supported in an object structure service are also supported in an enterprise Web service. You can deploy an enterprise Web service to use a JMS queue (asynchronous process) or to bypass the JMS queue (synchronous process).

- **Standard Web service** - Standard Web services are created from methods that are defined in application services. The methods must be annotated in the application service to be available for Web service implementation. The integration framework links input and output parameters of the methods to the Web Services Description Language operation parameters.
Data Import and Data Export

With the integration framework, you can load data from either flat files, such as comma separated, or XML files. You can initiate the data load through an application user interface. You also can start a data load from a scheduled background process by using a system cron task.

With the integration framework, you can export data in a batch mode. You can start a data export of the content that is associated with the publish channel from the application user interface. You can filter the content to limit the amount of data that is being exported. You can export data to a destination by using any of the available endpoints and handlers that the integration framework provides.

Content

The integration framework provides predefined content that facilitates your integration to external applications and services. The predefined content available for your use includes:

- Over 40 predefined object structures that use many of the primary business objects within the system. Object structures have one or more business objects and contain the relationships that are needed between business objects.

- Corresponding enterprise services and publish channels for the predefined object structures.

- One external system that is configured to use all the predefined enterprise services and publish channels.

- Eight predefined handlers that support different communication protocols, such as Web services and HTTP.

Integration Framework for Operational Management Product Integration

The integration framework provides components and features that support the integration between the system and the operational management products.

Process Management Products

Process management products, or system built applications, escalations, and workflows, use the integration framework to make calls to operational management products by using defined logical management operations and integration modules. Process management products can integrate with operational management products in an automated mode using integration modules. Process management products also can integrate with operational management products in an assisted mode by using launch in context.

Most process management products have mechanisms to automate tasks such as software deployment by using the system supported logical management
operations and integration modules. The process management product provides an action 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. Process management product processing can involve saving a value to the system database or displaying the response to you from a user interface, or both.

**Action**

A process management product can implement a custom action Java class to call an integration module. An action can be associated with a system application, a system workflow, or a system escalation. When you initiate the action, the system runs the registered Java class, which can be coded to call an integration module. The integration module then calls the operational management product.

**Logical Management Operations**

A logical management operation, such as a software deployment, defines the action that the process management product takes on the operational management product. Typically, a process management product takes action against a configuration item, such as a server.

Logical management product definitions act as the interface between the process management product and the integration module. The logical management operation allows the integration module and the process management product to be designed and implemented, independent of each other.

A logical management product created by an integration module would identify the actions that the integration module supports for an operational management product. A logical management product created by a process management product identifies the actions the process management product needs the integration module to support.

The logical management product record identifies the following properties:

- The name and description of the action that it supports
- Whether operational management processing is synchronous or asynchronous
- The input (source) and output (target) objects and fields that are required for the logical management product

**Integration Modules**

The integration module provides a mechanism for a process management product, such as change or release, to invoke an external operational management product.

The integration module provides the capability for a process management product to communicate with an operational management product for specific logical management operation actions. The integration module is the integration component that resides between the process management product and the operational management product.
When invoked by a process management product, the integration module uses data that is passed by the process management product to assist in the invocation of the operational management product service. The integration module may also return the operational management product response data to the process management product.

When installed, integration modules include the logical management operations that they support for an operational management product. Depending upon the level of complexity, you can implement an integration module as a Java class or an invocation channel.

**Operational Management Products**

Operational management products are external products that you can use to perform information technology services. IBM® Tivoli® Application Dependency Discovery Manager (TADDM), Tivoli Provisioning Manager (TPM), and IBM® Tivoli® Configuration Manager (TCM) are examples of operational management products. Operational management products provide services that external applications (integration modules) can invoke to initiate operational management product actions.

Operational management product definitions are registered in the system and can be loaded from the discovery engine using the Integration Composer. You can also load operational management product definitions by using the features of the integration framework, such as object structure services.

The operational management product definitions include properties of the operational management product, such as a host name. The definitions also include configuration item relationships for those configuration items that are managed by the operational management product.

**Integration Framework for User Interface Integration**

The integration framework provides components and features that support user interface-based integration between the system applications and the external applications.

**Launch Entries**

You create a launch entry record in the Launch in Context application. A launch entry defines the URL of an external application that you use to open an external application.

Launch entries can have the following properties:

- Specific business objects or multiple objects to identify the objects that can restrict the use of a launch entry to certain applications
- Context by substituting object field values into the URL string
- An object classification value that controls the launch entry visibility in a user interface (only show the launch entry on the user interface based on
the classification value of current data being processed in the user interface

▼ Operational management product-specific features including the automatic substitution of operational management product host name, and the configuration item source token into the URL based on configuration item the system processes in the user interface

**Land in Context**

You can use the land in context to have an external application open a system application user interface and to pass context information as part of the URL string.
Data that the integration framework receives from an external application is identified as inbound data. The integration framework can facilitate inbound data exchange asynchronously or synchronously.

Asynchronous messages are processed through the Java Message Service (JMS) queues. JMS queues can process messages in order of priority, or in a multi-threaded manner.

Synchronous messages are not processed through the JMS queues. A direct connection is established between the integration framework and the external application. The integration framework returns a response to the external application that confirms success or failure of the processed message.

The integration framework expects all XML messages to be encoded in a Unicode format.
Asynchronous Inbound Integration Processing

The following prerequisites apply to asynchronous inbound processing:

- All applicable object structures, services, and external systems must be defined.
- The following entities must be enabled:
  - External system
  - Applicable enterprise services with operations other than QUERY
  - If applicable, the cron task that polls the inbound sequential queue
  - If applicable, the message-driven beans that process messages from the continuous queue

Query type enterprise services cannot be processed asynchronously in the integration framework.

Enterprise Services

The following diagram gives an overview of the asynchronous enterprise service processing activities. Not every activity applies to every inbound message.
Inbound Integration Process Initiation

Summary

An external system can use the following methods to send asynchronous messages to the system:

<table>
<thead>
<tr>
<th>Method</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML that uses the integration framework that uses HTTP Post, Enterprise JavaBean (EJB) Invocation, Web service invocation</td>
<td>All external systems with enabled enterprise services. Enterprise services must support an operation other than QUERY.</td>
</tr>
<tr>
<td>Data import feature, that uses XML or flat files</td>
<td>External systems with enabled enterprise services. The enterprise service object structure must support flat structures for flat file processing.</td>
</tr>
<tr>
<td>Interface tables</td>
<td>External systems with enabled enterprise services. Enterprise service object structure must support flat structures.</td>
</tr>
<tr>
<td>Data import cron task</td>
<td>External systems with enabled enterprise services. Enterprise service object structure must support flat structures.</td>
</tr>
</tbody>
</table>

Messages in the Inbound Queues

HTTP Post and Enterprise Beans

The following steps describe the initiation of inbound processing using the integration framework:

1. A servlet provides the ability for external systems to post messages to the system using HTTP or HTTPS. You can use the following URL to post the messages to enterprise services that process messages asynchronously:

   \[
   \text{hostname:port/meaweb/esqueue/extsysname/entservname}
   \]

   - \text{extsysname} is the name of the external system.
   - \text{entservname} is the name of the enterprise service.

2. J2EE™ clients can invoke Enterprise JavaBeans that are written according to the Sun Microsystems®, Inc., J2EE client specification.

   You can use the following code to invoke the Enterprise JavaBean to process the enterprise service message asynchronously:

   \[
   \text{public byte[] processExternalDataAsync(byte[] extData, String serviceName, String sender)}
   \]
Asynchronous Inbound Integration Processing

A client can use the Java Naming and Directory Interface (JNDI) name of the Enterprise JavaBean (ejb/maximo/remote/enterpriseservice) to look up the Enterprise JavaBean reference and to invoke the method. The client also needs the following settings:

- Access to the Home and Remote class files
- Access to the J2EE jar files for the server
- The URL of the server that hosts the Enterprise JavaBean
- The context factory class name

The client code must instantiate the default InitialContext object. The context derives the provider URL and the context factory from the environment.

3 The integration framework checks that the external system and the enterprise service are valid and enabled.

If the verification fails, the integration framework notifies the sender of the error and does not process the message.

**Tip** To see if an external system is enabled, go to the System tab in the External Systems application.

**Tip** To see if an enterprise service is valid and enabled, go to the Enterprise Services tab in the External Systems application.

4 If the verification is successful, the integration framework identifies the inbound JMS queue that is assigned to the selected enterprise service and external system.

**Tip** To find the location of that queue, go to the System tab in the External Systems application.

5 The integration framework writes the message to the inbound queue. If the message contains multiple instances of a document, for example, if a single message contains ten person records, the application writes the single message, not ten individual messages, to the queue.

If one of the referred records has a processing error, a complete message processing exception is identified and none of the other records are committed to the database.

6 The integration framework updates the JMS message header with the external system and enterprise service names.

**Data Import**

The following steps describe the initiation of inbound processing that uses the data import feature.

1 The integration framework checks that the external system and the enterprise service are valid and enabled.

If you are importing flat files, the integration framework also checks that the enterprise service object structure supports flat structures.
If the verification fails, the integration framework issues an error and does not process the message.

2 If the verification is successful, the integration framework identifies the inbound JMS queue that is assigned to the enterprise service and the external system.

3 The integration framework writes the message to the inbound queue. If the message contains multiple instances of a document, for example, if a single message contains ten person records, the application writes ten messages to the queue.

   If one of the messages has a processing error, a single message processing exception is identified and none of the other messages that can be successfully processed are committed to the database.

4 The integration framework updates the JMS message header with the external system and enterprise service names.

You can preview the data that you want to import to check the format and the data validity of a source file. It is a synchronous validation mechanism that presents all the source file processing errors without committing data to the database.

The preview data process checks that the data structure of the selected file complies with the integration XML or flat file definitions.

Interface Tables

The following steps describe the initiation of inbound processing that uses interface tables.

1 The external system writes message data to the appropriate interface tables. It also updates the MXIN_INTER_TRANS queue table with information about the sequence in which the interface table records must be processed.

2 A system cron task regularly polls the MXIN_INTER_TRANS queue table for records to be processed.

3 If any records are ready to be processed, the integration framework identifies the sender as the external system and the enterprise service name from the MXIN_INTER_TRANS queue table.

4 The integration framework checks that the external system and enterprise service are valid and enabled.

   The integration framework checks that the enterprise service object structure supports flat structures.

   If the verification fails, the integration framework sends an error notification to the system administrator and stops processing the inbound message.

5 If the verification is successful, the integration framework identifies the inbound JMS queue that is assigned to the enterprise service and external system and writes the message to that queue.
Asynchronous Inbound Integration Processing

**Data Import Cron Task**

The following steps describe the initiation of inbound processing that uses a data import cron task.

1. The XMLFILECONSUMER cron task or the FLATFILECONSUMER cron task regularly polls the source directory that is specified in its corresponding configuration.

2. The files in the source directory are processed according to the cron task configurations.

3. The integration framework checks that the external system and enterprise service are valid and enabled.
   - If you are importing flat files, the integration framework also checks that the enterprise service object structure supports flat structures.
   - If the verification fails, the integration framework sends an error notification to the system administrator and stops processing the inbound message.

4. If the verification is successful, the integration framework identifies the inbound JMS queue that is assigned to the enterprise service and the external system, and writes the message to that queue.

**Inbound Queue Message Retrieval**

**Summary**

You can choose to process inbound messages through the sequential queue, the continuous queue, or a combination of the two. The sequential queue processes messages on a strict first-in-first-out basis. The continuous queue supports multi-threaded processing of messages.

When an error occurs in the sequential queue, the application sends an e-mail message to the system administrator. The sequential queue also flags the message as an error and continues to reprocess the message in error. Until the error is corrected or removed, the queue stops processing all unflagged messages.

When an error occurs in the continuous queue, the application sends an e-mail message to the system administrator. The continuous queue also flags the message as an error, then continues processing subsequent messages in the queue.

**Tip**

To see which inbound queue (sequential or continuous) that an external system uses, go to the System tab in the External Systems application.

The inbound messages that are received into the queue from the interface tables or flat files are converted to an XML format.

A message remains in the inbound queue until the message processing completes. The transaction is then deleted from the queue.

**Object Structure Identification**

**Summary**

The integration framework identifies the object structures that are associated with the enterprise service and creates a copy of the message for each object structure.
The following framework activities identify the object structures that are associated with the message:

1. The integration framework retrieves the record from the inbound queue.

2. The integration framework identifies the enterprise service from the retrieved record.

3. The enterprise service definition lists the object structures that the processing service uses.

4. The integration framework creates one copy of the enterprise service message for every object structure that is associated with the enterprise service.

   The processing sequence that is associated with the object structure specifies the sequence in which the subsequent object structure processing is to be performed on the enterprise service.

   The integration framework inbound integration can process numbers in either standard or scientific notation.

   **Tip** To see the processing sequence of multiple object structures associated with an enterprise service, look in the Add/Modify Additional Object Structures dialog box in the Enterprise Services application. Look for the processing order field on the Object Structure Sub-Records table window.

   The Enterprise Service main object structure is always processed after all the additional object structures.

   If the integration framework creates copies of the enterprise service message, the remaining inbound processing actions apply to each copy of the enterprise service.

**Output**

   The output of this activity is one copy of the enterprise service message per object structure.

**User Exit Preprocessing**

**Summary**

   If you specify custom processing logic in the user exit class, the integration framework applies custom processing logic to the enterprise service.

   You can use a preprocessing method to manipulate the enterprise service before any predefined processing takes place.

   The predefined services do not provide any user exit classes.

   **Tip** To see if preprocessing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration and Enterprise Service Response Configuration table windows show user exit classes, check each class file for a preprocessing method value.

   The same processing class contains the user exit preprocessing, user exit postprocessing, and user exit object processing methods.
Asynchronous Inbound Integration Processing

**Output**

The following table shows the possible user exit preprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit class exists.</td>
<td>The unchanged enterprise service.</td>
</tr>
<tr>
<td>The user exit class skips the record (due to non-applicable data).</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td>The user exit class stops the record (because of an error).</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The updated enterprise service.</td>
</tr>
</tbody>
</table>

**Enterprise Service Class Processing**

**Summary**

The MAXIMO adapter does not provide any predefined enterprise service processing classes. It implements all integration rules by using the object structure processing and processing rules.

An enterprise service class can be implemented when your integration framework uses an ERP adapter.

Adapters use the processing class for the following purposes:

- To apply enterprise service-specific business rules that cannot be specified by using the processing rules
- To convert input data from enterprise service format to object structure format

**Tip**

To see if a processing class exists for the enterprise service, go to the Enterprise Service tab in the Enterprise Services application. Look in the Enterprise Service Configuration and Enterprise Service Response Configuration table windows for processing class values.

**Output**

The following table shows the possible enterprise service class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No processing class exists.</td>
<td>The unchanged enterprise service.</td>
</tr>
<tr>
<td>The processing class skips the record.</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td>The processing class stops the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
</tbody>
</table>
Asynchronous Inbound Integration Processing

User Exit Postprocessing

Summary

The integration framework applies predefined processing logic in the user exit class to the output from the preceding activity.

A user exit processing class typically customizes the object structure after the execution of any predefined enterprise service processing logic. Both the object structure and the enterprise service are available for use.

The MAXIMO adapter does not provide predefined user exit classes.

**Tip**

To see if any postprocessing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration and Enterprise Service Response Configuration table windows show user exit classes, check these class files for a postprocessing method values.

The same processing class contains the user exit preprocessing, user exit postprocessing, and user exit object processing methods.

Output

The following table shows the possible user exit postprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>The processing class completes successfully, with mapping.</td>
<td>The object structure.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output is</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit class exists.</td>
<td>The unchanged enterprise service object.</td>
</tr>
<tr>
<td>The user exit class skips the record.</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The user exit class stops the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The updated object structure.</td>
</tr>
</tbody>
</table>

XSL Map

Summary

The integration framework applies any mapping to the enterprise service to convert it to an object structure format.

The MAXIMO adapter does not provide predefined XSL maps.

**Tip**

To see if an XSL map exists for an enterprise service, go to the Enterprise Service tab in the Enterprise Services application. Look on the Enterprise Service Configuration and Enterprise Service Response Configuration table windows for XSL map values.
Asynchronous Inbound Integration Processing

Output

The following table shows the possible XSL map processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No XSL map exists.</td>
<td>The unchanged enterprise service.</td>
</tr>
<tr>
<td>An XSL map exists.</td>
<td>The updated enterprise service.</td>
</tr>
</tbody>
</table>

Object Structure Multiplication

Summary

If the transaction applies to multiple organizations or sites, the integration framework creates a copy of the object structure for each organization or site.

In most cases, a transaction applies to a single organization or site. A cross-reference control creates a copy of the object structure for each applicable organization or site.

For example, you want to insert a vendor from an external system into the system, under multiple organizations. You can send a single vendor message once and apply a cross-reference control. A copy of each vendor is created for each applicable organization.

The MAXIMO adapter does not provide any predefined cross-reference controls.

If you must skip a duplicated object structure for a specific organization or the site must be skipped because of inapplicable data, apply the skip action to that object structure only. If you must stop a duplicated object structure because of an error in the data, apply the stop action to all of the object structures that are created (every copy of the original object structure).

Tip

To see if a cross-reference control exists, go to the Enterprise Service tab in the Enterprise Services application.

The following steps describe the steps the integration framework takes when it multiplies an object structure:

1. The integration framework checks for a cross-reference control for the object structure and enterprise service.

2. If a control exists, the integration framework copies the object structure once for each value in the control.

3. The integration framework determines whether the value in the cross-reference control is an organization value or site value. The integration framework then replaces the organization or site in the object structure with the organization or site from the cross-reference control.

Output

The output of this activity is an object structure for each organization or site that is specified in the cross-reference control.
Object Structure Processing Rules

Summary
The integration framework applies processing rules to the object structures before it builds the objects.

Object structure processing rules define conditions under which the system can skip or stop a message, or change data in the object structure before the objects are created. You can manipulate the primary or unique keys that are defined for an object.

Tip
To see if inbound processing rules exist for the object structure, go to the Object Structure Processing Rules tab of the Enterprise Service tab in the Enterprise Services application.

The following table shows the possible object structure rule processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing rules exist.</td>
<td>The unchanged object structure.</td>
</tr>
<tr>
<td>The processing rules skip the record.</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td>The processing rules stop the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>The processing rules complete successfully.</td>
<td>The updated object structure.</td>
</tr>
</tbody>
</table>

Object Creation

Summary
The integration framework builds the objects by using the information in the object structure.

Output
The output of this activity is the objects.

Object Processing Rules

Summary
The integration framework applies processing rules to the objects that it builds before saving the objects.

You can use object processing rules to manipulate data in the object, before the objects are saved. You can also use processing rules to access and retrieve pertinent data from objects that you do not include in the object structure.

Tip
To see if processing rules exist for the object, go to the Object Processing Rules tab of the Enterprise Service tab in the Enterprise Services application.
Asynchronous Inbound Integration Processing

Output

The following table shows the possible object rule processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object processing rules exist.</td>
<td>The object.</td>
</tr>
<tr>
<td>The processing rules skip the record.</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The processing rules stop the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>The processing rules complete successfully.</td>
<td>The updated object.</td>
</tr>
</tbody>
</table>

Object Structure Processing Class

Summary

The integration framework applies any predefined logic to the object structure.

Output

The following table shows the possible object structure class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing exists.</td>
<td>The unchanged object.</td>
</tr>
<tr>
<td>The object structure class skips the record.</td>
<td>None; processing ends and the message is deleted from the queue.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The object structure class stops the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td></td>
<td>The predefined enterprise services do not implement the stop action.</td>
</tr>
<tr>
<td>The object structure processing completes successfully.</td>
<td>The updated object.</td>
</tr>
</tbody>
</table>

User Exit Object Processing

Summary

The integration framework applies any final custom logic to the object.

Tip

To see if object processing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration table window shows a user exit class, check that class file for an object processing method.
Asynchronous Inbound Integration Processing

The same processing class contains the enterprise service user exit preprocessing, enterprise service user exit postprocessing, and user exit object processing methods.

Output

The following table shows the possible user exit object processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit processing.</td>
<td>The unchanged object.</td>
</tr>
<tr>
<td>The user exit class stops the record.</td>
<td>None; processing ends and the message in error remains in the queue.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The updated object.</td>
</tr>
</tbody>
</table>

Object Processing

Summary

The objects are passed to the system, and standard system processing is applied.

The integration framework treats a single message from the inbound queue as a single system transaction. All objects that a message creates must be successfully processed before the database commit transaction can occur. An error in any one of the multiple object structures causes the entire transaction to fail.
Enterprise services that support query and create operations, return XML content as part of the synchronous response. The response processing layer supports Java and XSL customization, but the response does not support processing rules. Any operation can be processed synchronously, but only query and create operations return data to the caller.

Enterprise Services

The following diagram illustrates an overview of the synchronous enterprise Web service processing activities. Not every activity applies to every inbound message.

**Inbound Integration Process Initiation**

**Summary**

An enterprise service can use HTTP Post, Enterprise JavaBean invocation, and Web service invocation methods for enterprise service synchronous message responses and receipts:

**HTTP Post and Enterprise Beans**

The following steps describe the initiation of inbound processing in the integration framework:

1. A servlet enables the external systems to post messages to the system. You can use the following URL to post the transactions to enterprise services that process synchronously:

   \[\text{hostname:port/meaweb/es/extsysname/entservname}\]

   - *extsysname* is the name of the external system.
   - *entservname* is the name of your enterprise service.
2 You can use the following code to invoke the Enterprise JavaBean to process the enterprise service message synchronously:

```java
public byte[] processExternalDataSync(byte[] extData, String serviceName, String sender)
```

A client can use the Java Naming and Directory Interface name of the Enterprise JavaBean (ejb/maximo/remote/enterpriseservice) to look up the Enterprise JavaBean reference and to invoke the method. The client also needs the following settings:

- Access to the Home and Remote class files
- Access to the J2EE jar files for the server
- The URL of the server that hosts the Enterprise JavaBean
- The context factory class name

The client code must instantiate the default InitialContext object. The context derives the provider URL and the context factory from the environment.

3 The integration framework checks that the external system and enterprise service are valid and enabled.

If the verification fails, the integration framework notifies the sender of the error and does not process the message.

**Tip**

To see if an external system is enabled, go to the System tab in the External Systems application.

To see if an enterprise service is valid and enabled, go to the Enterprise Services tab in the External Systems application.

### Object Structure Identification

**Summary**

The integration framework identifies the object structures that are associated with the enterprise service and creates a copy of the message for each object structure.

The following activities identify the object structures that are associated with the message:

1. The integration framework retrieves the record from the inbound queue.
2. The integration framework identifies the enterprise service from the retrieved record.
3. The enterprise service definition lists the object structures used by the processing service.
4. The integration framework creates one copy of the enterprise service for every object structure that is associated with the enterprise service.

The processing sequence that is associated with the object structure specifies the sequence in which the subsequent object structure processing is to be performed on the enterprise service.
Synchronous Inbound Integration Processing

The integration framework inbound integration can process numbers in either standard or scientific notation.

**Tip**
To see the processing sequence of multiple object structures associated with an enterprise service, go to the Add/Modify Additional Object Structures dialog box in the Enterprise Services application. Look for the processing order field on the Object Structure Sub-Records table window.

The Enterprise Service primary object structure is always processed after all the additional object structures.

If the integration framework creates copies of the enterprise service, the remaining inbound processing actions apply to each copy of the enterprise service.

**Output**
The output of this activity is one copy of the enterprise service per object structure.

**User Exit Preprocessing**

**Summary**
The integration framework applies the custom processing logic in the user exit class to the enterprise service.

You can use a preprocessing method to manipulate the enterprise service before any predefined processing takes place.

The predefined services do not provide any user exit classes.

**Tip**
To see if preprocessing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration and Enterprise Service Response Configuration table windows show user exit classes, check each class file for a preprocessing method value.

The same processing class contains the user exit preprocessing, user exit postprocessing, and user exit object processing methods.

**Output**
The following table shows the possible user exit preprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit class exists.</td>
<td>The existing enterprise service, unchanged.</td>
</tr>
<tr>
<td>The user exit class skips the record (due to non-applicable data).</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The user exit class stops the record (due to an error).</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The existing enterprise service, updated.</td>
</tr>
</tbody>
</table>
Enterprise Service Class Processing

Summary
The integration framework applies predefined enterprise service processing logic in the processing class to the enterprise service.

The MAXIMO adapter does not provide any predefined enterprise service processing classes. It implements all its integration rules through the object structure processing and processing rules.

Adapters use the processing class to apply enterprise service-specific business rules that cannot be specified using the processing rules, and to convert input data from enterprise service format to object structure format.

Tip
To see if a processing class exists for the enterprise service, go to the Enterprise Service tab in the Enterprise Services application. Look on the Enterprise Service Configuration and Enterprise Service Response Configuration table windows for processing class values.

Output
The following table shows the possible enterprise service class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No processing class exists.</td>
<td>The existing enterprise service, unchanged.</td>
</tr>
<tr>
<td>The processing class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The processing class stops the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The processing class completes successfully, with mapping.</td>
<td>The object structure.</td>
</tr>
</tbody>
</table>

User Exit Postprocessing

Summary
If specified, the integration framework applies predefined processing logic in the user exit class to the output from the preceding activity.

A user exit processing class typically customizes the object structure after the execution of any predefined enterprise service processing logic. Both the object structure and the enterprise service are available for your use.

The MAXIMO adapter does not provide predefined user exit classes.

Tip
To see if any postprocessing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration and Enterprise Service Response Configuration table windows show user exit classes, check these class files for a postprocessing method values.

The same processing class contains the user exit preprocessing, user exit postprocessing, and user exit object processing methods.
Synchronous Inbound Integration Processing

Output

The following table shows the possible user exit postprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit class exists.</td>
<td>The unchanged enterprise service object.</td>
</tr>
<tr>
<td>The user exit class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The user exit class stops the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The updated enterprise service.</td>
</tr>
</tbody>
</table>

XSL Map

Summary

The integration framework applies any mapping to the enterprise service to convert it to object structure format.

The MAXIMO adapter does not provide predefined XSL maps.

Tip

To see if an XSL map exists for an enterprise service, go to the Enterprise Service tab in the Enterprise Services application. Look on the Enterprise Service Configuration and Enterprise Service Response Configuration table windows for XSL map values.

Output

The following table shows the possible XSL map processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No XSL map exists.</td>
<td>The enterprise service, unchanged.</td>
</tr>
<tr>
<td>An XSL map exists.</td>
<td>The object structure.</td>
</tr>
</tbody>
</table>

Object Structure Multiplication

Summary

If the transaction applies to multiple organizations or sites, the integration framework creates a copy of the object structure for each organization or site.

In most cases, a transaction applies to a single organization or site. A cross-reference control directs the system to create a copy of the object structure for each applicable organization or site.

For example, you want to insert a vendor from an external system into the system, under multiple organizations. You can insert a vendor once and apply a cross-reference control. The system creates a copy of each vendor for each applicable organization.

The MAXIMO adapter does not provide any predefined cross-reference controls.

If you must skip a duplicated object structure for a specific organization or the site must be skipped because of inapplicable data, apply the skip action to that object structure only. If you must stop a duplicated object structure because of an error
in the data, apply the stop action to all of the object structures that are created (every copy of the original object structure).

**Tip**
To see if a cross-reference control exists, go to the Enterprise Service tab in the Enterprise Services application.

The following steps describe the steps the integration framework takes when it multiplies an object structure:

1. The integration framework checks for a cross-reference control for the object and enterprise service.

2. If a control exists, the integration framework copies the object structure once for each value in the control.

3. The integration framework determines whether the value in the cross-reference control is an organization or site value. The integration framework then replaces the organization or site in the object structure with the organization or site from the cross-reference control.

**Output**
The output of this activity is an object structure for each organization or site that is specified in the cross-reference control.

**Object Structure Processing Rules**

**Summary**
The integration framework applies any processing rules to the object structures, before it builds the objects.

Object structure processing rules define conditions under which the system can skip or stop a message, or change data in the object structure before the objects are created. You can manipulate the primary or unique keys that are defined for an object.

**Tip**
To see if inbound processing rules exist for the object structure, go to the Object Structure Processing Rules tab of the Enterprise Service tab in the Enterprise Services application.

The following table shows the possible object structure rule processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing rules exist.</td>
<td>The unchanged object structure.</td>
</tr>
<tr>
<td>The processing rules skip the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The processing rules stop the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The processing rules complete successfully.</td>
<td>The updated object structure.</td>
</tr>
<tr>
<td></td>
<td>The primary keys of the objects.</td>
</tr>
</tbody>
</table>
Object Creation

Summary: The integration framework builds the objects by using the information in the object structure.

Output: The output of this activity is the objects.

Object Processing Rules

Summary: If specified, the integration framework applies processing rules to the objects it has built, before saving the objects.

Object processing rules let you manipulate data in the object, before the objects are saved. You can also use processing rules to access and retrieve pertinent data from objects that you do not include in the object structure.

Tip: To see if processing rules exist for the object, go to the Object Processing Rules tab of the Enterprise Service tab in the Enterprise Services application.

Output: The following table shows the possible object rule processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object processing rules exist</td>
<td>The object.</td>
</tr>
<tr>
<td>The processing rules skip the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The processing rules stop the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The processing rules complete successfully.</td>
<td>The object, possibly updated.</td>
</tr>
</tbody>
</table>

Object Structure Processing Class

Summary: The integration framework applies any predefined logic to the object structure.

Output: The following table shows the possible object structure processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing exists.</td>
<td>The unchanged object.</td>
</tr>
<tr>
<td>The object structure class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
</tbody>
</table>
User Exit Object Processing

Summary

The integration framework applies any final custom logic to the object.

Tip

To see if object processing customization exists, go to the Enterprise Service tab in the Enterprise Services application. If the Enterprise Service Configuration table window shows a user exit class, check that class file for an object processing method.

The same processing class contains the enterprise service user exit preprocessing, enterprise service user exit postprocessing, and user exit object processing methods.

Output

The following table shows the possible user exit object processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No user exit processing.</td>
<td>The object, unchanged.</td>
</tr>
<tr>
<td>The user exit class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The user exit class stops the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The user exit class completes successfully.</td>
<td>The object, possibly updated.</td>
</tr>
</tbody>
</table>

Object Processing

Summary

The objects are passed to the system and standard system processing is applied.

The integration framework treats a single message from the inbound queue as a single system transaction. All objects that a message creates must be successfully processed before the database commit transaction can occur. An error in any one of the multiple object structures causes the entire transaction to fail.

Enterprise Service Response Processing

A synchronous enterprise service provides a response to the invoking application. The service indicates whether the message was processed successfully. For the query and create operations, XML content is returned to the invoking application.
For the create operation, the keys of the primary object of the object structure are returned.

The entire object structure is returned based on the results of the Query operation. When you use an enterprise service you can implement customizations on the response of the query by using Java and XSL.

**Enterprise Service Web Services**

You can access enterprise services by using HTTP or an enterprise bean. You can deploy enterprise services as Web services. External applications can use Web services to query or to send messages to the integration framework.
Object Structure Services

The following diagram illustrates an overview of the synchronous object structure service processing activities.

Inbound Integration Process Initiation

Summary

An object structure service can use HTTP Post, Enterprise JavaBean invocation, and Web service invocation methods to receive synchronous object structure service messages.

HTTP Post and Enterprise Beans

1. A servlet enables the external systems to post messages to the system. You can use the following URL to post the transactions to object structure services that process synchronously:

   \[ \text{hostname:port/meaweb/os/osname} \]

   - \text{oasename} is the name of the object structure.

2. J2EE™ clients can invoke Enterprise JavaBeans that are written according to the Sun Microsystems®, Inc., J2EE client specification.

   To invoke the Enterprise JavaBean to process the message synchronously:

   \[
   \text{public byte[] processMOS(byte[] reqmosData, String mosName)}
   \]

   A client can use the Java Naming and Directory Interface name of the Enterprise JavaBean (ejb/maximo/remote/mosservice) to look up the Enterprise JavaBeans reference and to invoke the method. The client also needs the following settings:
Synchronous Inbound Integration Processing

- Access to the Home and Remote class files
- Access to the J2EE jar files for the server
- The URL of the server that hosts the Enterprise JavaBean
- The context factory class name

The client code must instantiate the default InitialContext object. The context derives the provider URL and the context factory from the environment.

Object Structure Identification

Summary

The integration framework identifies the object structure that are associated with the service and creates a copy of the message.

The following activities identify the object structure that are associated with the message:

1. The object structure definition identifies the operations to which the object structure is mapped.
2. The integration framework identifies the object structure name and operation from the root element in the inbound XML transaction.

The integration framework inbound integration can process numbers that are represented in standard or scientific notation.

Object Creation

Summary

The integration framework builds the objects by using the information in the object structure.

Output

The output of this activity is the objects.

Object Structure Processing Class

Summary

The integration framework applies any predefined logic to the object structure.

Output

The following table shows the possible object structure processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing exists.</td>
<td>The object, unchanged.</td>
</tr>
<tr>
<td>The object structure class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple records in one XML file, the class skips the current record and processes the next record.</td>
</tr>
<tr>
<td>The object structure class stops the record.</td>
<td>None; processing ends and an error is sent to the caller.</td>
</tr>
<tr>
<td>The object structure processing completes successfully.</td>
<td>The object, possibly updated.</td>
</tr>
</tbody>
</table>
Object Processing

Summary

The objects are passed to the system, and standard system processing is applied. If the system identifies an error in processing, it sends the object structure to the caller of the service.

Object Structure Web Services

You can access object structure services by using HTTP or an enterprise bean. You can deploy object structure services as Web services. External applications can use Web services to query or to send messages to the integration framework.
Standard Services

The following diagram illustrates an overview of the synchronous standard service processing activities.

Inbound Integration Process Initiation

Summary
A standard service can be accessed by using an HTTP Post, an Enterprise JavaBean invocation, and a Web service invocation to receive synchronous standard service messages.

HTTP Post and Enterprise Beans
The following steps describe the initiation of inbound processing in the integration framework:

1. A servlet provides the ability for external systems to post transactions to the system. You can use the following URL to post the transactions to standard services that process synchronously:

   $hostname:port/meaweb/ss/maximo service name$

   - *maximo service name* is the name of the application service.

2. J2EE™ clients can invoke Enterprise JavaBeans that are written according to the Sun Microsystems®, Inc., J2EE client specification.

   To invoke the Enterprise JavaBean to process the message synchronously:

   ```java
   public byte[] action(byte[] actionData, String maxServiceName)
   ```

   A client can use the Java Naming and Directory Interface name of the Enterprise JavaBean (ejb/maximo/remote/actionservice) to look up the Enterprise JavaBean reference and invoke the method. The client also needs the following settings:

   - Access to the Home and Remote class files
Synchronous Inbound Integration Processing

- Access to the J2EE jar files for the server
- The URL of the server that hosts the Enterprise JavaBean
- The context factory class name

The client code must run the default InitialContext object. The context derives the provider URL and the context factory from the environment.

Object Processing

Summary
The objects are passed to the system, and standard system processing is applied.

Standard Web Services

You can access standard services by using HTTP or an enterprise bean. You can deploy standard services as Web services. External applications can use Web services to query or to send messages to the integration framework.
Data that the integration framework sends to an external application is outbound data. The integration framework can facilitate outbound data exchange asynchronously or synchronously.

Asynchronous messages are processed through the Java Message Service (JMS) queue. JMS queues can process messages in the queues in either first in order of sequence, or in a multi-threaded manner.

Synchronous messages are not processed through the JMS queue. A direct connection is established between the integration framework and the external application. The external application returns a response to the service invoker that confirms success or failure of message processing.
Asynchronous Outbound Integration Processing

The following prerequisites apply to all asynchronous outbound integrations:

- All applicable object structures, publish channels, and external systems must be defined.
- The external systems must be configured with an endpoint.
- The following entities must be enabled:
  - External systems
  - Applicable publish channels
  - Applicable event listeners
  - The cron task that polls the outbound queue

Publish Channels

The following diagram illustrates an overview of the asynchronous publish channel processing activities. Not every activity applies to every outbound message.
Outbound Integration Process Initiation

Summary

You initiate the outbound integration process by using event-based activities or by using the data export feature that you access in the External Systems application.

Event-based messages are sent when a record is updated using an application or a process such as workflow or escalation.

When you use the data export feature, you can export the result set of a query from the integration framework to an external system. The data export process uses different publish channels other than what is used for real-time integration.

The integration framework performs two types of outbound integration processing.

Real Time Integration Processing

1. You complete a transaction in the system. This step is the only step in the outbound integration process that is visible to you.

2. The primary object that is associated with the transaction identifies the related publish channel that has the event listener enabled. The outbound integration process is initiated for all publish channels with enabled listeners.

   TIP To see if the event listener is enabled for a particular publish channel, see the Enable Listener check box in the Publish Channels application.

Batch Integration Processing

1. In the External Systems application, select the external system to which data is to be sent.

2. On the Publish Channels tab, select a publish channel, then select the Data Export button.

3. Select the records to be exported by using a SQL query statement. The query must act on the primary object in the object structure.

   TIP To see if the name of the primary object in an object structure, go to the Object Structure tab in the Object Structures application.

4. If the query returns a valid result set, the data export process continues.
Object Structure Identification

Summary

The integration framework builds an object structure from the objects.

1. The object structure that is associated with each publish channel is determined.

2. The object structure identifies its component objects.

3. The integration framework constructs the object structure from the objects.

4. The integration framework adds the following header information to the object structure:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object structure name</td>
<td>The name of the object structure.</td>
</tr>
<tr>
<td>messageID</td>
<td>Unique message identifier.</td>
</tr>
<tr>
<td>creationDateTime</td>
<td>System datetime.</td>
</tr>
</tbody>
</table>

Output

The output of this activity is an object structure with the following format and header data:

```xml
<PublishObjectStructureName creationDateTime="system"
<ObjectStructureNameSet>
   <ObjectName>
      .
      .
   </ObjectName>
</ObjectStructureNameSet>
</PublishObjectStructureName>
```

Object Structure Processing Rules

Summary

Object structure processing rules define conditions under which the system can skip or stop a message. Rules also can change in an object structure. You have a final opportunity to manipulate the primary and unique keys that are defined for an object.

Tip

To see if outbound processing rules exist for the object structure, go to the Processing Rules for Sub-Records table window of the Publish Channel tab in the Publish Channels application.
Asynchronous Outbound Integration Processing

Outbound Integration Processing

Output

The following table shows the possible object structure processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object structure processing rules exist.</td>
<td>Object structure, unchanged.</td>
</tr>
<tr>
<td>The processing rules skip the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The processing rules stop the record.</td>
<td>None; processing ends, and an exception opens on the user interface screen.</td>
</tr>
<tr>
<td>The processing rules complete successfully.</td>
<td>Object structure may be updated.</td>
</tr>
</tbody>
</table>

Object Structure Multiplication

Summary

If the publish channel is associated with multiple external systems, the integration framework creates copies of the object structure.

1. The integration framework creates a copy of the object structure for every publish channel and external system combination.

2. The integration framework does not change the object structure XML unless the processing class or rules modify the XML.

Output

The outcome of this activity is one object structure per combination of publish channel and external system. The following format and header data appears:

```xml
<PublishChannelName creationDateTime="system" messageID="MessageID per external system - publish channel" event="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/maximo">
    <ObjectStructureNameSet>
        <ObjectName>...
        ...
        </ObjectName>
    </ObjectStructureNameSet>
</PublishChannelName>
```

The actions listed in each stage apply to each copy of the object structure that is created for a specific external system and publish channel combination.

The skip action applies only to the copy of the object structure when the outbound processing for an external system-publish channel combination determines that a duplicated object structure is skipped due to inapplicable data.

The stop action applies to all the copies of the object structure created in this multiplication step when the processing determines that a copied object structure is stopped due to an error in the data.
Asynchronous Outbound Integration Processing

User Exit Preprocessing

Summary

The integration framework applies the custom processing logic in the user exit class to the object structure.

You can use a preprocessing method in a user exit class to manipulate the object structure before any predefined processing takes place on the publish channel.

The predefined publish channels do not provide any user exit classes.

Tip

To see if preprocessing customization exists, go to the Publish Channel tab in the Publish Channels application. If the Publish Channel tab shows a user exit class, check that class file for a preprocessing method.

The same processing class contains the user exit preprocessing and user exit postprocessing methods.

Output

The following table shows the possible user exit processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No preprocessing method exists.</td>
<td>The existing object structure, unchanged.</td>
</tr>
<tr>
<td>The preprocessing method skips the record.</td>
<td>None; processing ends</td>
</tr>
<tr>
<td>The preprocessing method stops the record.</td>
<td>None; processing ends, and an exception opens on the user interface screen.</td>
</tr>
<tr>
<td>The preprocessing method completes successfully.</td>
<td>The existing object structure, possibly updated.</td>
</tr>
</tbody>
</table>

Publish Channel Class Processing

Summary

The integration framework applies predefined publish channel processing logic in the processing class to the object structure.

A publish channel processing class typically implements additional processing logic and converts data from an object structure format to a publish channel format.

The MAXIMO adapter does not provide any predefined publish channel processing classes. The XML format for the publish channels that are defined within the MAXIMO adapter is comparable to the format of the object structure, as no XML mapping takes place.

Tip

To see if a processing class exists for a publish channel, go to the Publish Channel tab in the Publish Channels application. On the Publish Channel tab look for a processing class value.
Output

The following table shows the possible publish channel class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No processing class exists.</td>
<td>The existing object structure, unchanged.</td>
</tr>
<tr>
<td>The processing class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The processing class stops the record.</td>
<td>None; processing ends, and an exception displays on the user interface screen.</td>
</tr>
<tr>
<td>The processing class completes successfully, without mapping.</td>
<td>The existing object structure, possibly updated.</td>
</tr>
<tr>
<td>The processing class completes successfully, with mapping.</td>
<td>The XML file in an external system format.</td>
</tr>
</tbody>
</table>

User Exit Postprocessing

Summary

The integration framework applies custom processing logic in the user exit class to the publish channel XML that is created in the preceding activity.

This option is typically used to customize a predefined publish channel after publish channel processing takes place. Both the input object structure and the publish channel that are created from that object structure are available.

The predefined publish channels do not provide any user exit classes.

Tip

To see if postprocessing customization exists, go to the Publish Channel tab in the Publish Channels application. If the Publish Channel tab shows a user exit class, check that class file for a postprocessing method.

The same processing class contains the user exit preprocessing and user exit postprocessing methods.

Output

The following table shows the possible user exit postprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No postprocessing method exists.</td>
<td>The existing object structure or publish channel, unchanged.</td>
</tr>
<tr>
<td>The postprocessing method skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The postprocessing method stops the record.</td>
<td>None; processing ends, and an exception displays on the user interface screen.</td>
</tr>
<tr>
<td>The postprocessing method completes successfully.</td>
<td>The existing publish channel, possibly updated.</td>
</tr>
</tbody>
</table>
Asynchronous Outbound Integration Processing

XSL Map

Summary

The integration framework applies any custom mapping to the publish channel.

XSL mapping lets you map user-defined publish channels or customize the mapping done by predefined publish channel processing classes. Both the object structure and publish channel are available.

**Tip**  
To see if an XSL map exists for a publish channel, go to the Publish Channel tab in the Publish Channel application. Look on the Publish Channel tab for an XSL map.

Output

The output of this activity must be an XML message in publish channel format.

The notation used by the integration framework to represent numbers greater than seven digits is controlled by the `mxe.int.usesscientific` variable. By default, numbers greater than seven digits use a scientific notation.

Send the Publish Channel Data to the External System

Summary

The following activities take place when the integration framework writes and sends the publish channel data to the external system:

1. The integration framework writes the XML message to the outbound queue specified for the external system.

   **Tip** To see which outbound queue an external system uses, go to the System tab in the External Systems application.

2. The cron task that polls the outbound queue picks up the message.

3. The cron task passes the message to a message router. The router uses the endpoint that is associated with the external system, or the publish channel, to identify and to invoke the correct handler. If the publish channel does not have an associated endpoint, the integration framework uses the external system endpoint.

   **Tip** To see which endpoint a system uses, go to the System tab in the External Systems application.

   To see which endpoint a publish channel uses, go to the Publish Channels tab in the External Systems application.

   To see which handler an endpoint uses, select Add/Modify End Points from the Select Action menu in the End Points application.

4. The processing class that is associated with the handler sends the data to the external system.
Synchronous Outbound Integration Processing

As a prerequisite, you must define all applicable object structures and endpoints on the invocation channel.

Invocation Channels

The following diagram illustrates an overview of the synchronous invocation channel processing activities. Not every activity applies to every outbound message.
Outbound Integration Process Initiation

Summary
A Service Oriented Architecture (SOA) environment enables the use of external services for the purposes of data processing from multiple data sources. Invocation channels support this generic service-oriented architecture capability by enabling the integration framework to call external applications and process their response.

You initiate the outbound integration process. This process can be implemented by using an action class which calls an invocation channel. You can implement an action by using the following means:

- A user interface control (within an application)
- Workflow routing
- Escalation

Request Object Structure Definition

Summary
The integration framework performs the following activities when it builds a request object structure from the objects:

1. The request object structure that is associated with each invocation channel is determined.
2. The request object structure identifies its component objects.
3. The integration framework constructs the request object structure from the objects.
4. The integration framework adds the following header information to the request object structure:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object structure name</td>
<td>The name of the object structure.</td>
</tr>
<tr>
<td>messageID</td>
<td>Unique message ID.</td>
</tr>
<tr>
<td>creationDateTime</td>
<td>System datetime.</td>
</tr>
</tbody>
</table>

Output
The output of this activity is an object structure with the following format and header data:

```xml
<InvokeObjectStructureName creationDateTime="system"
messageID="1234444" event="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/maximo">
  <ObjectStructureNameSet>
    <ObjectName>
      .
      .
      .
    </ObjectName>
  </ObjectStructureNameSet>
</InvokeObjectStructureName>
```
**Request Class Processing**

**Summary**

The integration framework applies predefined invocation channel request class processing logic in the processing class to the request object structure.

An invocation channel request processing class typically implements additional processing logic and converts data from an object structure format to an invocation channel format.

The MAXIMO adapter does not provide any predefined invocation channel processing classes. The XML format for invocation channels, defined within the MAXIMO adapter, is comparable to the format of the object structure, as no XML mapping takes place.

**Tip**

To see if request class processing exists for an invocation channel, go to the Invocation Channel tab in the Invocation Channels application. On the Service Request Configuration table window look for a request processing class value.

**Output**

The following table shows the possible request class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No processing class exists.</td>
<td>The existing object structure, unchanged.</td>
</tr>
<tr>
<td>The processing class skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The processing class stops the record.</td>
<td>None; processing ends, and an exception is returned to the caller of the invocation channel.</td>
</tr>
<tr>
<td>The processing class completes successfully, without mapping.</td>
<td>The existing object structure, possibly updated.</td>
</tr>
<tr>
<td>The processing class completes successfully, with mapping.</td>
<td>The invocation channel.</td>
</tr>
</tbody>
</table>

**Request User Exit Preprocessing**

**Summary**

The integration framework applies custom request processing logic in the user exit class to the invocation channel.

This option is typically used to customize a predefined invocation channel before invocation channel request processing takes place. Both the input object structure and the invocation channel that are created from that object structure are available.

The MAXIMO adapter does not provide any predefined invocation channels.

**Tip**

To see if preprocessing customization exists, go to the Invocation Channel tab in the Invocation Channels application. If the Service Request Configuration table window shows a request user exit class, check that class file for a preprocessing method.
The following table shows the possible request user exit preprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No preprocessing method exists.</td>
<td>The existing object structure or invocation channel, unchanged.</td>
</tr>
<tr>
<td>The preprocessing method skips the record.</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The preprocessing method stops the record.</td>
<td>None; processing ends, and an exception is returned to the caller of</td>
</tr>
<tr>
<td></td>
<td>the invocation channel.</td>
</tr>
<tr>
<td>The preprocessing method completes successfully.</td>
<td>The existing invocation channel, possibly updated.</td>
</tr>
</tbody>
</table>

**Request XSL Map**

**Summary**

The integration framework applies any custom request mapping to the invocation channel.

You can use XSL mapping to map user-defined invocation channels or customize the mapping done by predefined invocation channel processing classes. Both the object structure and invocation channel are available.

**Tip**

To see if request XSL mapping exists for an invocation channel, go to the Invocation Channel tab in the Invocation Channel application. Look on the Service Request Configuration table window for a request XSL map.

**Output**

The output of this activity must be an XML message in invocation channel format.

The notation used by the integration framework to represent numbers greater than seven digits is controlled by the `mxe.int.useScientific` variable. By default, numbers greater than seven digits use a scientific notation.

**Send the Invocation Channel Data to the Endpoint**

**Summary**

The integration framework writes the invocation channel data to the location that you specify on the endpoint. The endpoint determines the data transport mechanism of an invocation channel.

1. The integration framework writes the XML message to the endpoint location.

**Tip**

To see what location, and the transport mechanism the endpoint uses, go to the End Point tab in the End Points application.

To see which handler an endpoint uses, select Add/Modify End Points from the Select Action menu in the End Points application.

2. The handler processing class invokes an external application or system.
Response Class Processing

Summary
The integration framework applies predefined invocation channel response class processing logic in the processing class to the response object structure.

An invocation channel response processing class typically implements additional processing logic and converts data from an invocation channel to an object structure format.

The MAXIMO adapter does not provide any predefined invocation channel processing classes. The XML format for invocation channels that are defined within the MAXIMO adapter is comparable to the format of the object structure, as no XML mapping takes place.

The integration framework can process numbers represented in either standard or scientific notation.

Tip
To see if response class processing exists for an invocation channel, go to the Invocation Channel tab in the Invocation Channels application. On the Service Response Configuration table window look for a response processing class value.

Output
The following table shows the possible response class processing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No processing class exists.</td>
<td>The existing object structure, unchanged.</td>
</tr>
<tr>
<td>The processing class skips the record.</td>
<td>None; processing ends</td>
</tr>
<tr>
<td>The processing class stops the record.</td>
<td>None; processing ends, and an exception is returned to the caller of the invocation channel.</td>
</tr>
<tr>
<td>The processing class completes successfully, without mapping.</td>
<td>The existing response object structure, possibly updated.</td>
</tr>
<tr>
<td>The processing class completes successfully, with mapping.</td>
<td>The updated response object structure.</td>
</tr>
</tbody>
</table>

Response User Exit Preprocessing

Summary
The integration framework applies custom response processing logic in the user exit class to the invocation channel.

This option is typically used to customize a predefined invocation channel before invocation channel response processing takes place. Both the invocation channel and the response object structure are available.

The predefined invocation channels do not provide any user exit classes.

Tip
To see if preprocessing customization exists, go to the Invocation Channel tab in the Invocation Channels application. If the Service Response Configuration table window shows a response user exit class, check that class file for a preprocessing method.
Synchronous Outbound Integration Processing

Output

The following table shows the possible response user exit preprocessing outcomes.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>No preprocessing method exists.</td>
<td>The existing object structure or invocation channel, unchanged.</td>
</tr>
<tr>
<td>The preprocessing method skips the record</td>
<td>None; processing ends.</td>
</tr>
<tr>
<td>The preprocessing method stops the record.</td>
<td>None; processing ends, and an exception is returned to the caller of the invocation channel.</td>
</tr>
<tr>
<td>The preprocessing method completes successfully.</td>
<td>The existing response object structure.</td>
</tr>
</tbody>
</table>

Response XSL Mapping

Summary

The integration framework applies any custom response mapping to the invocation channel.

You can use XSL mapping to map user-defined invocation channels or customize the mapping done by predefined invocation channel processing classes. Both the response object structure and invocation channel are available.

Tip

To see if response XSL mapping exists for an invocation channel, go to the Invocation Channel tab in the Invocation Channel application. Look on the Service Response Configuration table window for a response XSL map.

Output

The output of this activity must be an XML message in an invocation channel format.

Object Processing

Summary

The objects are build from response object structure data and are passed to the system where standard system processing is applied.

The system must successfully process all the objects that are created from the response object structure that is created from the invocation channel response before it performs the database commit.
The integration framework creates various structures, elements, and attributes for XML messages. The content of an XML message is based on the object structure that you associate with a service or a channel. Additionally, the elements are formed based on the service or the channel operation behavior.
Integration XML Structure

The standard format for all integration XML messages that are exchanged through the integration framework is an object structure within a root element. Standard services do not support the use of object structures.

For example, the XML has the following format which is based on the PERSON object structure:

```xml
<syncMXPERSON>  (root element)
       <MXPERSON> (object structure element)
              <PERSON> (object element)
                     <PHONE> (object element)
                     </PHONE> (object element)
                     <EMAIL> (object element)
                     </EMAIL> (object element)
                     <SMS> (object element)
                     </SMS> (object element)
              </PERSON>
       </MXPERSON>
</syncMXPERSON>
```

Integration XML Content

The content of any XML document is based on the associated object structure and the operation that the channel or service performs.

The following channels and services are available.

- Invocation channel
- Publish channel
- Enterprise service
- Object structure service
- Standard service
The following operations are available.

- Create
- Delete
-Invoke
- Publish
- Query
- Sync
- Update

**Root Element**

The root element of an XML message contains one or more object structures. The name of the root element is the concatenation of the operation and the name of the associated object structure.

The following table shows the attributes that can apply to root elements. All attributes in this table are optional.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Type</th>
<th>Applicable to</th>
</tr>
</thead>
<tbody>
<tr>
<td>baselanguage</td>
<td>Base language in which the content values are supplied.</td>
<td>string</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>creationDateTime</td>
<td>Date and time the content was generated.</td>
<td>dateTime</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>maximoVersion</td>
<td>A concatenated string that identifies the major version, minor version, build, and database build.</td>
<td>MaximoVersionType</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>messageID</td>
<td>The system generates this value for all messages.</td>
<td>string</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>transLanguage</td>
<td>Language in which the values for multilanguage-enabled fields are supplied.</td>
<td>string</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Type</td>
<td>Applicable to</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>event</td>
<td>The origin of an outbound XML message.</td>
<td>eventType</td>
<td>All output operations</td>
</tr>
<tr>
<td></td>
<td>Valid values are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 (false)—Generated by the Data Export feature.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 (true)—Generated by an outbound integration event listener (that is, data entry in an application).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uniqueResult</td>
<td>Specifies whether the query expects one record or multiple records in a response.</td>
<td>Boolean</td>
<td>Queries (input)</td>
</tr>
<tr>
<td></td>
<td>If the value is 0 or the attribute is not specified, the query can return multiple records.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the value is 1, the query can return a single record; otherwise, an error occurs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxItems</td>
<td>If the query can return multiple records, this attribute limits the number of records to be returned at one time.</td>
<td>positiveInteger</td>
<td>Queries (input)</td>
</tr>
<tr>
<td></td>
<td>If this attribute is not specified, the response contains the entire result set.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rsStart</td>
<td>The first record in the response.</td>
<td>integer</td>
<td>Queries (input) and responses (output)</td>
</tr>
<tr>
<td></td>
<td>In queries:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use with maxItems to specify the first record to be returned in a response.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Type</td>
<td>Applicable to</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>rsStart</td>
<td>In responses to queries: This value matches the rsStart value in the corresponding query. If the corresponding 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.</td>
<td>integer</td>
<td>Responses (output)</td>
</tr>
<tr>
<td>rsCount</td>
<td>The number of records returned in the message. If the original query specifies a maxItems value, the rsStart value for the subsequent request for additional records is rsStart + rsCount + 1.</td>
<td>integer</td>
<td>Responses (output)</td>
</tr>
<tr>
<td>rsTotal</td>
<td>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.</td>
<td>integer</td>
<td>Responses (output)</td>
</tr>
</tbody>
</table>
The following examples illustrate a root element with attributes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
creationDateTime="2008-09-28T21:49:45" baseLanguage="EN"
transLanguage="EN" messageID="11876346770938768" maximoVersion="7
1 Harrier 060 HARRIER-066" uniqueResult="0" maxItems="201"
rsStart="100">
  <max:MXPERSONQuery>
    <max:PERSON>
      .
      .
      .
    </max:PERSON>
  </max:MXPERSONQuery>
</max:QueryMXPERSON>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
creationDateTime="2008-09-28T21:49:45" baseLanguage="EN"
transLanguage="EN" messageID="11876346770938768" maximoVersion="7
1 Harrier 060 HARRIER-066" rsStart="100" rsCount="100" rsTotal="100">
  <max:MXPERSONSet>
    <max:PERSON>
      .
      .
      .
    </max:PERSON>
  </max:MXPERSONSet>
</max:QueryMXPERSONResponse>
```
Object Structure Element

Every XML document contains one or more object structure elements.

The following table lists the attributes that can apply to the object structure element. All attributes in this table are optional.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Type</th>
<th>Applicable to</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The value is derived from action attribute of the primary object within the message.</td>
<td>ProcessingActionType</td>
<td>All input and output operations</td>
</tr>
<tr>
<td></td>
<td>For outbound messages, this attribute is for informational purposes only.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For inbound messages, the processing logic uses this value only for the sync operation; otherwise the system ignores it.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relationship</td>
<td>Identifies the relationship that the system uses to retrieve the object using the parent object.</td>
<td>string</td>
<td>All input and output operations</td>
</tr>
<tr>
<td>deleteForInsert</td>
<td>Identifies a child object that the system must delete before reinserting.</td>
<td>string</td>
<td>All input and output operations</td>
</tr>
<tr>
<td></td>
<td>This attribute applies only to inbound messages where the operation is sync and the action is change.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following example illustrates an object structure element with attributes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<max:SyncMXPERSON>
  <max:MXPERSONSet>
    <max:PERSON action="Delete" relationship=EMAIL" deleteForInsert="EMAIL" transLanguage="EN">
      ...
    </max:PERSON>
  </max:MXPERSONSet>
</max:SyncMXPERSON>
```

Key Fields

Any field that is part of a parent key and child key, appears only in the parent object in the object structure. For example, PERSONID is part of the key of both the PERSON and PHONE objects, but in the MXPERSON object structure it appears only in the PERSON object.
Field Attributes

The following attributes apply at the field level on outbound XML messages:

- changed
- glorder
- langenabled
- maxvalue
- maxencrypted

These attributes apply to messages that are generated by an outbound integration event. The changed attribute does not apply to messages that are generated by the Data Export feature.

The changed Attribute

The changed attribute is a boolean field attribute that contains the value 1 when you modify the value in the corresponding field. This information lets you identify the field-level change that triggers a publish channel. This attribute does not appear in XML that the data export feature creates.

An outbound XML contains the changed attribute only when the transaction meets all the following conditions:

- An outbound, event-based transaction creates the message.
- The action attribute on the primary object is Change or Replace.
- The object structure uses the same parent-child object relationship as the corresponding application. For example, the MXPERSON object structure and the Person application must use the same relationships between PERSON, PHONE, EMAIL, and SMS.

```
<MXPERSON>
  <PERSON action="Replace">
    <PERSONID>123</PERSONID>
    <ADDRESSLINE1 changed="1" >1 Main Street</ADDRESSLINE1>
  </PERSON>
</MXPERSON>
```

If there are GL type fields, the changed attribute appears on the field name.

```
<GLDEBITACCT changed="1">
  <VALUE>6600-800-SAF</VALUE>
  <GLCOMP glorder="0">6600</GLCOMP>
  <GLCOMP glorder="1">800</GLCOMP>
  <GLCOMP glorder="2">SAF</GLCOMP>
</GLDEBITACCT>
```

The glorder Attribute

Fields that identify general ledger accounts have the following XML structure.
In outbound XML the value of a GL type field, including delimiters, appears in the VALUE child element within the field. The outbound XML also places the GL type field components, based on the database definition of the components, in the GLCOMP element. The glorder attribute in the GLCOMP element identifies the level of the component (GLORDER1 through GLORDER20).

```
<GLDEBITACCT>
    <VALUE>6600-800-SAF</VALUE>
    <GLCOMP glorder="0">6600</GLCOMP>
    <GLCOMP glorder="1">800</GLCOMP>
    <GLCOMP glorder="2">SAF</GLCOMP>
</GLDEBITACCT>
```

Inbound XML messages can contain GL account numbers in one of the following formats:

- The external system can provide the individual components, as in the following example:

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

  In this case, the system validates the components and uses the segment delimiter defined in the GLCONFIGURE table to recreate the account number.

- The external system can provide the entire account number, separated by the defined delimiter, as in the following example:

  ```xml
  <GLDEBITACCT>
    <VALUE>6400-2-10</VALUE>
  </GLDEBITACCT>
  ```

  In this case, the delimiter is used to identify each component, and the account number is validated accordingly.

If the external system provides both an account number and its individual components, the system ignores the components.

**The langenabled Attribute**

For information about the langenabled attribute, see Chapter 20, "Multiple Language Support," on page 259.
Object Structure Element

The maxvalue Attribute

Fields that are associated with a synonym type domain can specify the corresponding maxvalue. This value is available for customization or exit processing. It is informational only and is not used for processing.

```xml
<MXITEM>
  <ITEM>
    <ITEMNUM>560-00</ITEMNUM>
    <DESCRIPTION>Tubing, Copper-1 In ID X .030 In Wall Test</DESCRIPTION>
    <LOTTYPE maxvalue="NOLOT">NOLOT</LOTTYPE>
  </ITEM>
</MXITEM>
```

The maxencrypted Attribute

The maxencrypted attribute is a boolean attribute. The attribute contains the value of 1 when its associated field includes data that has been encrypted by the system for the purposes of privacy and security. External systems can identify whether a decryption process must be applied at the field level.

```xml
<MXPERUSER>
  <PERSON action="Replace">
    <PERSONID>123</PERSONID>
    <MAXUSER>
      <MAXUSERID>10</MAXUSERID>
      <PASSWORD mxencrypted="1">dmFzdG8=</PASSWORD>
    </MAXUSER>
  </PERSON>
</MXPERUSER>
```

The action Attribute

The action attribute is an optional attribute that applies to the content of primary and child objects in an object structure. It specifies the type of processing that the receiving system is to perform on the XML message.

This attribute applies to inbound XML documents that perform data synchronization (operation = sync). It also applies to publish channels that an outbound integration event generates. The Data Export feature does not update records and therefore it does not include an action attribute in the messages that it generates.

The following table lists the valid values for the action attribute. Action values are case-sensitive.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Add records to the database in the receiving system.</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete records from the database in the receiving system.</td>
</tr>
<tr>
<td>Change</td>
<td>Update existing records in the database in the receiving system.</td>
</tr>
<tr>
<td>Replace</td>
<td>Update existing records in the database in the receiving system.</td>
</tr>
<tr>
<td>AddChange</td>
<td>Add or update existing records in the database in the receiving system.</td>
</tr>
</tbody>
</table>
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.

An action attribute at the primary object level specifies the overall processing action that applies to the parent and child records. At the child object level, it indicates processing that is specific to that record.

Business rules always take precedence over the action attribute. If business rules prohibit the action that is specified on an inbound XML message, an error occurs. For example, an inbound transaction that attempts to update a closed PO will result in an error.

If a single XML document 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, each with its own action attribute.

```xml
<MXVENDOR>
  <COMPANIES action="Change">
    <COMPANY>TEST4</COMPANY>
    <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>
  </COMPANIES>
</MXVENDOR>
```

Add action

An add action indicates that the corresponding data is added to a database. For inbound transactions, an error occurs if the data exists. The add action on a primary object extends to its child objects, so it is not necessary to specify the add action at the child object level. Outbound transactions contain an add action when the insert of an object generates the transaction.

Delete action

A delete action indicates that the corresponding object and its child objects are deleted from the database. The delete action on a primary object extends to its child objects, so it is not necessary to specify the delete action at the child object level.

An outbound XML message with a delete action on the primary object may not include the child objects, but the receiving system is responsible for identifying and deleting them.

When a primary object in an inbound XML message contains the delete action, the integration framework deletes the child objects that are related to the primary
Object Structure Element

object. If the parent object does not exist in the database, no error is reported to the sending system.

Update actions

The change and replace actions indicate that the existing database records are updated. The AddChange action can result in an update when the record that is to be processed exists in the receiving system.

The change and replace actions are interchangeable in the case of updates to single-level standard object structures. For hierarchical object structures, the two actions have different XML content and require different processing by the receiving system.

Change action

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. It always contains the parent of any child record to be updated, even if the parent is unchanged.

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 action codes.

<table>
<thead>
<tr>
<th>Action Code of Child object</th>
<th>Processing Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Add the child record; if it exists, an error results.</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete the child record; if it does not exist, an error results.</td>
</tr>
<tr>
<td>Change</td>
<td>Update the child record; if it does not exist, an error results.</td>
</tr>
<tr>
<td>Null or no action specified</td>
<td>If the child record exists, update it; if child record does not exist, add it.</td>
</tr>
</tbody>
</table>

If an action code is not listed in the preceding table, an error occurs in the processing of the XML document.

Replace action

A replace action on the primary object in an XML message indicates that the message contains a complete set of objects that represent the net result of additions, changes, and deletions to the object structure. These objects replace the existing database records, and any database record that is not referenced in the XML message is deleted.

For outbound processing, the system always uses the replace action, not the change action.

In inbound processing, the system deletes any existing child record that is not explicitly mentioned in the message. External systems must process replace actions in the same manner; that is, delete any child records that are not part of the XML document.

The replace action can appear only on 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 integration framework does not process the message. If a child level record contains a replace action when the parent contains any action other than change, the system ignores the action on the child record.
AddChange action

The AddChange action is like the replace action, except that any existing child record that is not specified in the message is not deleted. An AddChange action on the primary object adds the primary record and all the sub-records that are provided in the message, if the primary record does not exist in the database. If the primary record does exist, it is updated along with any child record that is provided. Existing child records that are not provided in the inbound message are not deleted. The AddChange action does not apply to child objects.

The AddChange action is useful in cases where the object structure definition contains elements that are not available in the external system.

For example, the MXVENDORInterface enterprise service contains both vendor and contact information. If the database maintains the contacts for a vendor, and external systems maintain the vendor definition itself, sending an inbound vendor record with the action value equal to null results in the deletion of contact information in the database. However, sending a vendor record with the action value equal to AddChange results in the update of vendor information; the contact information remains.

Default action attributes

The system processes the message in the following ways when an inbound XML message does not contain an action attribute:

- If the primary record does not exist in the database, the system performs add action processing.
- If the primary record exists in the database, the system performs replace action processing.

Comparison of the change, replace, and add change actions

The following diagram contrasts the Change, Replace, and AddChange actions applied to a hypothetical purchase order.

The Original PO diagram shows the initial purchase order. It contains three line items, each with two cost lines.

The Changes to the PO diagram shows the changes to the purchase order that originated in the Purchase Orders application.

- A change to the POCOST2 record associated with POLINE1
- Deletion of the POCOST3 record associated with POLINE1
- A change to POLINE2
Deletion of POLINE3 and, by default, its child POCOST records

In the diagram, the PO with the action value equal to Change or AddChange has records that accompany a change action on the primary PO object.

For the sample PO, the following records are sent from sending system:

- A change action on the PO record (PO 101), due to the changes in its child POLINE and POCOST records
- A change action on POLINE 1, due to changes in its child PCOST records
- A change action on the POCOST 2 record within POLINE 1, due to changes in that record
- A delete action on the POCOST 3 record within POLINE 2
- A change action on the POLINE 2 record, due to changes in that record
- A delete action on the POLINE 3 record (which deletes all its child records)

In the diagram, the PO with the action value equal to Replace has records that accompany a replace action on the primary object.
For the sample PO, the sending system sends the following records. Some may contain updated information; some may contain the same data that are in the database.

- The PO header
- POLINE 1 and POCOST lines 1 and 2
- POLINE 2 and POCOST lines 1 and 2

The XML message does not include the POCOST 3 record for POLINE 1, or the POLINE 3 record, so the receiving system deletes these records and POLINE 3 child records.

Records accompanying the add change action

In this example, the same records that accompany the change action would accompany the add change action.

Valid action attribute combinations

The following table shows the combinations of action attributes you can include on primary and child records. Find the action attribute for the parent record in the column on the left, then read across the table to find out if the action attribute for the child is valid.

<table>
<thead>
<tr>
<th>Child Record</th>
<th>Add</th>
<th>Delete</th>
<th>Change</th>
<th>Replace</th>
<th>AddChange</th>
<th>No Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>All child values ignored</td>
</tr>
<tr>
<td>Delete</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>All child values ignored</td>
</tr>
<tr>
<td>Change</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Replace and AddChange not allowed at child level</td>
</tr>
<tr>
<td>Replace</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>All child values ignored</td>
</tr>
<tr>
<td>AddChange</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>All child values ignored</td>
</tr>
<tr>
<td>No value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>All child values ignored</td>
</tr>
</tbody>
</table>
Additional Considerations

Boolean columns

In inbound transactions, a tag that represents a boolean field must contain a value of 0 (false) or 1 (true). If the tag does not contain a 0 or a 1, the object generates an error. If the XML does not include a tag for a boolean field, the object updates the corresponding database value 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 the system applies a decryption process to the received data. When the attribute value is 0 the system does not decrypt the received data.

If the attribute does not contain a 0 or a 1, the system does not decrypt the received data. If the XML field does not include a tag for an encrypted field, the system does not decrypt the received data.

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.

2004-12-06T10:11:58-05:00

Null columns

If a tag in an inbound transaction contains no value, the processing logic updates the corresponding database column with a null value. If the XML does not include a tag for a particular field, the processing logic does not update that field in the database.

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
Viewing XML

To view XML, select the Generate Schema/View XML action from the Select Action menu in the following applications:

- Enterprise Services
- Invocation Channels
- Object Structures
- Publish Channels

You also can view XML on the Web Service Library tab in the Web Services Library application.

Integration Schemas

The following circumstances dictate whether the system generates XML schemas:

- Publish channels publish an object structure
- Invocation channels facilitate the invocation of an external service
- Enterprise services process an object structure as input or output
- Object structure services process an object structure as input or output
- Standard services process an object, MboSet, MboList or specific individual typed parameters as input or output

The input and output types for all services, except standard services, are reusable, as they are based on the object structure for the service.

Key Fields

The generated schemas identify the key fields within a service or channel from the object structure definition and data dictionary definitions for the corresponding object. XML schema annotation identifies these fields.

For example, the ITEMNUM is a key field in the MXITEM object structure schema. It is indicated as follows:

```xml
<xsd:element name="ITEMNUM" minOccurs="0" type="MXString">
  <xsd:annotation>
    <xsd:documentation>ITEMNUM is a key field</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

You can use the schema to determine the fields that are required to process an inbound XML message.
XML Validation

Outbound and inbound XML transactions are not validated against the corresponding XML schema. Integration business rules apply to inbound data regardless of schema validation.

Namespace Property

To change the name of the XML name space, update the `mxe.int.xmlnamespace` property in the System Properties application.

The default name space property value is:

http://www.ibm.com/maximo

If you change the name space property, the system regenerates the MXMeta.xsd file, which contains the core reusable types that the system uses to build all the other schemas.

Generating a Schema

Whenever you change an object structure or the data dictionary, you must regenerate the affected schemas. The following changes also warrant a regeneration of the schema:

- Making a database field required or optional
- Changing the data type of a database field
- Adding or removing fields from the object structure
- Changing the structure of an object structure

The system generates or regenerates schemas when a Web service is deployed.

You can manually generate or regenerate schemas in the following applications:

- Enterprise Services
- Invocation Channels
- Object Structures
- Publish Channels
- Web Services Library
Schema Directory and Files

The generated schema files are in the subdirectories in the schema directory on the server.

<table>
<thead>
<tr>
<th>Group</th>
<th>Subdirectory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>schema/common/meta</td>
<td>Metadata schema file (MXMeta.xsd)</td>
</tr>
<tr>
<td>Object Structure</td>
<td>schema/common/mos</td>
<td>Object structure schema files</td>
</tr>
<tr>
<td>Object</td>
<td>schema/common/mbo</td>
<td>Object schema files</td>
</tr>
<tr>
<td>Service</td>
<td>schema/service</td>
<td>Service level schema files (for enterprise services, object structure services, and standard services)</td>
</tr>
</tbody>
</table>

Metadata Schema

The MXMeta.xsd file contains the core reusable types that the system uses to build all other schemas. The system regenerates this schema when someone changes the Namespace property.

Do not modify the MXMeta.xsd file. Changing this file can result in incorrect schemas and problems with Web services.

Schema Attributes

Besides actual data content, the metadata schema file includes the following properties that provide additional information about messages.

- Attribute groups
- Content types
- Query data types
- Supporting data types

Attribute Groups

The following table lists attributes by group.

<table>
<thead>
<tr>
<th>Group</th>
<th>Attributes</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonContentGroup</td>
<td>▼ baseLanguage</td>
<td>The root element of all input and output schema types.</td>
</tr>
<tr>
<td></td>
<td>▼ creationDateTime</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ maximoversion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ messageID</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ transLanguage</td>
<td></td>
</tr>
<tr>
<td>ObjectStructurePropertyGroup</td>
<td>▼ action</td>
<td>The object element of all input and output schema types.</td>
</tr>
<tr>
<td></td>
<td>▼ relationship</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ deleteForInsert</td>
<td></td>
</tr>
<tr>
<td>PublishingContentGroup</td>
<td>▼ event</td>
<td>The root element of all output schema types.</td>
</tr>
</tbody>
</table>
### Content, Query Data, and Supporting Data Types

The following tables list the content, query data, and supporting data types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaximoVersionType</td>
<td>Concatenated string that identifies:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ major version</td>
<td>changed</td>
</tr>
<tr>
<td></td>
<td>▼ minor version</td>
<td>changed</td>
</tr>
<tr>
<td></td>
<td>▼ build</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▼ database build</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifies software version</td>
<td></td>
</tr>
<tr>
<td>MXBooleanType</td>
<td>Extension of integer</td>
<td>changed</td>
</tr>
<tr>
<td>MXDateTimeType</td>
<td>Extension of dateTime</td>
<td>changed</td>
</tr>
<tr>
<td>MXDomainType</td>
<td>Extension of string; identifies the corresponding Maxvalue for a domain value.</td>
<td>changed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxvalue</td>
</tr>
<tr>
<td>MXDoubleType</td>
<td>Extension of double</td>
<td>changed</td>
</tr>
<tr>
<td>MXFloatType</td>
<td>Extension of float</td>
<td>changed</td>
</tr>
<tr>
<td>MXGLAccountType</td>
<td>Complex type with 2 values, “VALUE” and “GLCOMP”; identifies individual GL components and their sequential order for an account.</td>
<td>changed</td>
</tr>
<tr>
<td>MXGLComponentType</td>
<td>Extension of string; identifies GL component sequential order within the chart of accounts structure.</td>
<td>glorder</td>
</tr>
<tr>
<td>MXIntType</td>
<td>Extension of integer</td>
<td>changed</td>
</tr>
<tr>
<td>MLLangStringType</td>
<td>Extension of MXString</td>
<td>changed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>languageEnabled</td>
</tr>
<tr>
<td>MXLongType</td>
<td>Extension of long</td>
<td>changed</td>
</tr>
<tr>
<td>MXStringType</td>
<td>Extension of string</td>
<td>changed</td>
</tr>
</tbody>
</table>
## Integration Schemas

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXBooleanQueryType</td>
<td>Extension of integer</td>
<td>operator</td>
</tr>
<tr>
<td>MXDateTimeQueryType</td>
<td>Extension of dateTime</td>
<td>operator</td>
</tr>
<tr>
<td>MXDomainQueryType</td>
<td>Extension of string</td>
<td>operator</td>
</tr>
<tr>
<td>MXDomainMaxvalue</td>
<td></td>
<td>maxvalue</td>
</tr>
<tr>
<td>MXDoubleQueryType</td>
<td>Extension of double</td>
<td>operator</td>
</tr>
<tr>
<td>MXFloatQueryType</td>
<td>Extension of float</td>
<td>operator</td>
</tr>
<tr>
<td>MXGLAccountQueryType</td>
<td>Complex type with value “VALUE”</td>
<td>operator</td>
</tr>
<tr>
<td>MXGLComponentQueryType</td>
<td>Extension of string</td>
<td>operator</td>
</tr>
<tr>
<td>MXIntQueryType</td>
<td>Extension of integer</td>
<td>operator</td>
</tr>
<tr>
<td>MXLongQueryType</td>
<td>Extension of long</td>
<td>operator</td>
</tr>
<tr>
<td>MXStringQueryType</td>
<td>Extension of string</td>
<td>operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Restricted Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BooleanType</td>
<td>Indicates whether the result of a logical test is true or false.</td>
<td>▼ 0 (false)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ 1 (true)</td>
</tr>
<tr>
<td>ChangeIndicatorType</td>
<td>Indicates whether a field has a new value.</td>
<td>1 (true)</td>
</tr>
<tr>
<td></td>
<td>Applies only to object structures generated by an event.</td>
<td></td>
</tr>
<tr>
<td>EventType</td>
<td>Indicates whether a published object structure is the result of an event.</td>
<td>▼ 0 (false)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ 1 (true)</td>
</tr>
<tr>
<td></td>
<td>Event_Type is an extension of BooleanType.</td>
<td></td>
</tr>
<tr>
<td>ProcessingActionType</td>
<td>Processing actions that the integration services support.</td>
<td>▼ Add</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ Change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ Replace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ Delete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ AddChange</td>
</tr>
<tr>
<td>QueryOperatorType</td>
<td>Identifies the Query by Example action to be performed on the corresponding field.</td>
<td>▼ =</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ !=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ &amp;lt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ &lt;=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ &amp;gt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ &gt;=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▼ EW</td>
</tr>
</tbody>
</table>
Object Structure Schemas

Object structure schemas define the content of an object structure. Each object structure has a distinct schema that includes all the configured persistent and nonpersistent fields that are defined for each object in the structure.

Object structure schemas are not used directly as input or output for any service. They are used by system services while defining input or output types.

Schema Name

The naming convention for object structure schemas is the object structure name, for example, MXPERSON.xsd.

Schema Generation

When you generate following components, the object structure schemas are regenerated:

- The object structure schema (by using the select action menu 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

All object structure schemas include the MXMeta.xsd schema file. The generated schemas are under the schema/common/mos directory.

Schema Content

An object is configured only once within an object structure. All instances of usage of an object within an object structure have the same content. If an object is included as a child to two different parents, its content is the same in both positions.

The following elements are in the object structure schemas:

- Object
- Object set
- Object query
- Object delete

For example, the MXPERSON object structure has the following elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>MXPERSON</td>
</tr>
<tr>
<td>Object set</td>
<td>MXPERSONSet</td>
</tr>
<tr>
<td>Object query</td>
<td>MXPERSONQuery</td>
</tr>
<tr>
<td>Object delete</td>
<td>MXPERSONDelete</td>
</tr>
</tbody>
</table>
These elements are referenced in service level schemas and only the MXPERSON and MXPERSONSet can be outside the context of a service level schema. For example, internal applications such as Deployment Manager use schemas comparable to MXPERSON and MXPERSONSet. Integration users must use object structure services within the context of a service level schema.

### Object content

The following content is in the object element:

- Element MXPERSON has type MXPERSONType
- Complex type MXPERSONType has element PERSON (object name), which has type MXPERSON_PERSONType.
- 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 (PHONE, EMAIL, and SMS).
- Additional objects in the schema have a corresponding complex type like MXPERSON_PERSONType, that defines the PERSON object.

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHONE</td>
<td>MXPERSON_PHONEType</td>
</tr>
<tr>
<td>EMAIL</td>
<td>MXPERSON_EMAILType</td>
</tr>
<tr>
<td>SMS</td>
<td>MXPERSON_SMSType</td>
</tr>
</tbody>
</table>

The following example shows the structure of the corresponding XML:

```xml
<MXPERSON>
  <PERSON>
    <PHONE>
      ...
    </PHONE>
  ...
  <EMAIL>
    ...
  </EMAIL>
  ...
  <SMS>
    ...
  </SMS>
  ...
  </PERSON>
</MXPERSON>
```

### Object set content

For the preceding MXPERSON example, MXPERSONSet replaces the MXPERSON element and MXPERSONSetType replaces the complex type MXPERSONType. Everything else remains the same.
Integration Schemas

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:

- Element MXPERSONQuery is type MXPERSONQueryType.
- Complex type MXPERSONQueryType has elements for all the configured attributes of the top object (PERSON) of the object structure.

The following differences exist between the content of the query and the object set:

- The query element includes only the top object of the object structure.
- The query element does not include nonpersistent columns.
- The query element allows two occurrences of the elements to support a query on a range, such as a date range.

Since the query includes only the top object of the structure, you cannot have, for example, a query for a person by phone number. The phone number exists in the child PHONE object.

Object delete content

You can use delete elements only within the context of service level schemas.

The following content format is in the object delete element:

- Element MXPERSONDelete is type MXPERSONDeleteType.
- Complex type MXPERSONDeleteType has elements for all the configured attributes of the PERSON object.

The following differences exist between the content of the delete and the object set:

- The delete element includes only the top object of the object structure.
- The delete element does not include nonpersistent columns.

Object Schemas

Object schemas define the content of objects. These schemas are not used directly as input and output for any system service. They are used by system services while defining input and output types.

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.

Schema Name

The naming convention for object schemas is the object name, for example, PERSON.xsd.
Schema Generation

The system generates the files when you generate the following components:

- **Object structure schema** that contains the object
- **Object structure service or enterprise service schema** where the object structure referenced by the service includes the object
- **Standard service schema** that contains the object

All object schemas include the MXMeta.xsd schema file. The generated object schema files are under the schema/common/mbo directory.

Schema Content

The schema has the following elements (using MXPERSON as an example):

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSONMbo</td>
<td>PERSONMboType</td>
<td>This type contains one instance of the PERSONMbo element.</td>
</tr>
<tr>
<td>PERSONMboSet</td>
<td>PERSONMboSetType</td>
<td>This type contains multiple instances of the PERSONMbo element.</td>
</tr>
<tr>
<td>PERSONMboKey</td>
<td>PERSONMboKeyType</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If multiple attributes make up the primary key of an object, objectKeyType contains the attributes that make up the primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This element is included in the response to a Create operation.</td>
</tr>
<tr>
<td>PERSONMboKeySet</td>
<td>PERSONMboKeySetType</td>
<td>This type contains multiple instances of the PERSON element that is PERSONKeyType.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The PERSONKeyType contains the attribute that is the primary key of the PERSON object, PERSONID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If multiple attributes make up the primary key of an object, objectKeyType contains the attributes that make up the primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This Element is included in the Response of a Create operation.</td>
</tr>
<tr>
<td>PERSONMboQuery</td>
<td>PERSONMboQueryType</td>
<td>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.</td>
</tr>
</tbody>
</table>
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. For example, two AddMXPERSON services can have different processing rules but the same input (the MXPERSON object structure). The input and output schema types for enterprise services and object structure services are generally reusable.

The system defines the input and output for publish channels and invocation channels. This information is available to anyone who must implement these channels to invoke external services or to map to other output formats.

Schema Name

The naming convention for object structure schemas and enterprise services schemas is application service name + Service, for example, PERSONService.xsd.

The naming convention for service level schemas is object structure + Service, for example, MXPERSONService.xsd.

Schema Generation

Predefined service schema files are not provided. When you deploy a service, a service schema file is generated for the associated object structure, if one does not exist already.

The following conditions initiate a system file generation:

- You generate the service-level schema by using the Select Action menu in the Web Services Library application
- A Web service is deployed

The generated schema files are under the root/schema/common/service directory.

All service level schemas include the metadata schema file and the applicable object structure and object schema files.

Service Level Schema Elements

One schema file is generated per 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.

The service level schemas contain the following types, which use MXPERSON as an example.

- CreateMXPERSON
- CreateMXPERSONResponse
- DeleteMXPERSON
Integration Schemas

- InvokeMXPERSON
- InvokeMXPERSONResponse
- PublishMXPERSON
- QueryMXPERSON
- QueryMXPERSONResponse
- SyncMXPERSON
- UpdateMXPERSON

Standard Services

Standard services are the services that applications provide for performing specific operations on objects. Each service might have multiple methods that can be exposed as Web services. 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. Service schemas that are generated for objects and object structures are potentially reusable across different services.

Standard service input

The following diagram shows the pochangeStatus (type = pochangeStatusType) service input. It has the following parameters:

- PO object
- Status
- Date
- Memo
Standard Service Output

The following diagram shows the pochangeStatusResponse (type = pochangeStatusResponseType) output. For this example, the response of the method is the PO object.

Object Structure and Enterprise Services

The object structure definition contains the content for both object structure and enterprise services. Since an enterprise service provides additional capability outside of the data content, a single schema can support both services. However, the object structure service supports all the defined operations (Create, Delete, Update, and so on), while an enterprise service is created for a specific operation. Therefore, multiple enterprise services must be created to support multiple operations even though they reference the same object structure and thus use the same schema.

Two operations (Add and Query) support an output type. The output for the Add operation is a set of keys corresponding to the object that was added, and the output for the query operation is a result set of object structures. In addition to identifying the input and output types for object structure and enterprise services, these schemas also identify the output provided by a publish channel and the input and output of an invocation channel.

Object Structure and Enterprise Services Schema Content

The following examples describe the content of the object structure and enterprise services. All examples use the MXPERSON object structure.
CreateMXPERSON Element

The following diagram represents the content of the CreateMXPERSON element:

- The CreateMXPERSON element is type CreateMXPERSONType.
- The CreateMXPERSONType 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 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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateMXPERSON</td>
<td>UpdateMXPERSONType</td>
</tr>
<tr>
<td>SyncMXPERSON</td>
<td>SyncMXPERSONType</td>
</tr>
<tr>
<td>InvokeMXPERSON</td>
<td>InvokeMXPERSONType</td>
</tr>
<tr>
<td>InvokeMXPERSONResponse</td>
<td>InvokeMXPERSONResponseType</td>
</tr>
</tbody>
</table>
The following diagram represents the content of the CreateMXPERSONResponse element:

- Element CreateMXPERSONResponse is type CreateMXPERSONResponseType.
- The CreateMXPERSONResponseType has element PERSONMboKeySet, which is type of PERSONMboKeySetType.
- PERSONMboKeySetType has element PERSON, which is type PERSONKeyType.
- The PERSONKeyType contains only the PERSONID attribute of the PERSON object (the Primary Key of the PERSON).
DeleteMXPERSON Element

The following diagram represents the content of the DeleteMXPERSON element:

Below are the details:

- **Element** DeleteMXPERSON is type DeleteMXPERSONType.

- The DeleteMXPERSONType has element MXPERSONDelete, which is type MXPERSONDeleteType.

- MXPERSONDeleteType has elements for all the configured attributes of the PERSON object. Delete supports only the top object of the object structure (PERSON).
The following diagram represents the content of the PublishMXPERSON element:

- Element PublishMXPERSON 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 elements for the child objects (PHONE, EMAIL, and SMS) defined in the object structure.

The Publish element is like the Create element, but it uses the additional attribute event, which is defined in the PublishingContentGroup.
QueryMXPERSON Element

The following diagram represents the content of the QueryMXPERSON element:

▼ 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. Query supports only querying against the top object of the object structure (PERSON).

The Query element uses the additional attributes uniqueResults, maxItems, and rsStart, which are defined in the QueryContentGroup.
The following diagram represents the content of the QueryMXPERSONResponse element:

- 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, and SMS).
- Although the QueryMXPERSON element restricts querying to the top object of the object structure (PERSON), its response can contain the child objects (PHONE, EMAIL, and SMS).

The Query Response element uses the additional attributes rsStart, rsCount, and Total, which are defined in the ResponseContentGroup.
Interface tables are an option for integration with systems that use database tables to exchange data. Interface tables are generated based on the definition of the object structure that is associated with an enterprise service or publish channel. Invocation channels, object structure services, standard services, and query operations do not support interface tables.

Data synchronization messages can be exchanged by using interface tables. Interface tables do not support the Query and Invoke operations.

A single external system can transfer outbound data by using either XML messages, interface table flat messages, or both. You can assign an endpoint value to an external system at the publish channel level.
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 a local database or a remote database.

The predefined interface table endpoint (MXIFACETABLE) points to the local database, and you can add endpoints for remote databases.

Names of Interface Tables

The integration framework registers interface table names to an enterprise service or a publish channel. You must configure the object structure that is associated with the enterprise service or publish channel to support flat files. The Support Flat Structure check box must be selected on the object structure.

Default names for interface table are not provided. The following guidelines apply names for 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 should process the records in the respective interface tables. Two queue tables exist, one for inbound transactions and the other for outbound transactions.

<table>
<thead>
<tr>
<th>Interface Queue Table</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXOUT_INTER_TRANS</td>
<td>Outbound</td>
</tr>
<tr>
<td>MXIN_INTER_TRANS</td>
<td>Inbound</td>
</tr>
</tbody>
</table>

Some transactions depend on the successful processing of a previous transaction. For example, a PO record for an employee must be processed before the PO receipt record. The receiving system must process the records in the same sequence in which the sending system created the records.

All inbound and outbound transactions must have a record that is inserted into the corresponding inbound or outbound queue table. This record contains a unique sequential identifier called TRANSID, which is a value 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 employees and their phone numbers are entered into the system, the TRANSID values for the PO record must be lower than the TRANSID values for the PO receipt records that reference that PO.

The primary 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 pair of interface queue tables and own 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 dialog box 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. The alias conflict must also be resolved before an interface table is created.

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 option on the Select Action menu 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, restore the data to the new table. 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.

Deletion of Interface Tables and Records

When one or more related inbound transaction records are successfully processed in an interface table, the corresponding record is deleted from the MXIN_INTER_TRANS queue table. This means that the transaction was delivered successfully to the inbound Java Message Service (JMS) queue.

Records are deleted from the MXIN_INTER_TRANS queue table, never 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. An administrator must manage the archiving of data in the interface tables.

You cannot delete interface tables by using the user interface or by deleting the corresponding object structure. If necessary, a system administrator can drop the table.
Format of Interface Tables

The format of an interface table is the same as the format of the corresponding object structure. The interface table includes the persistent and nonpersistent columns that are included in the object structure. The interface table excludes the columns that are excluded from the object structure.

The interface tables include 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 the duplicate column name problem, change the system alias for 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.

Column Name Lengths and Aliases

Most columns do not have an alias, but some columns have aliases that support the predefined enterprise services or publish channels.

A database column can have only one alias. Ensure that any alias that you assign to a column is valid for every object structure that uses that column. If multiple object structures use the object, a change to an alias affects every interface table that is associated with those object structures.

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 dialog box 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. If a duplicate does not exist, you cannot change the alias.
Format of Interface Tables

Restricted Columns

The HASLD field, which is an internal system column, 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>
<thead>
<tr>
<th>Column Name</th>
<th>Interface Queue Tables?</th>
<th>Interface Tables?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFACENAME</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TRANSID</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TRANSSEQ</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>EXTSYSNAME</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ACTION</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IMPORTMESSAGE</td>
<td>Yes (used inbound only)</td>
<td>No</td>
</tr>
<tr>
<td>TRANSLANGUAGE</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MESSAGEID</td>
<td>Yes (used outbound only)</td>
<td>No</td>
</tr>
<tr>
<td>IFACETBNNAME</td>
<td>Yes (used outbound only)</td>
<td>No</td>
</tr>
</tbody>
</table>

IFACENAME Column

The IFACENAME column in the interface queue table contains the name of the enterprise service or the publish channel that is used in a transaction. The IFACENAME column is populated in the outbound transactions. For inbound transactions, the external system must populate the column with the enterprise service name or the publish channel name that corresponds to the row that is inserted into an interface table.

TRANSID Column

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.
For example, assume that you create a purchase order with one line item. This transaction uses the predefined MXPOInterface, and it increments the TRANSID value that is associated with the MXOUT_INTER_TRANS queue table to 1065. The transaction produces the following records:

- One entry in the MXOUT_INTER_TRANS queue table, with the IFACENAME value equal to the MXPOInterface value, and the TRANSID value equal to 1065.

- One entry in the MXPO_IFACE interface table, with the TRANSID value equal to 1065.

If the same purchase order has three line items, the transaction produces the following records:

- One entry in the MXOUT_INTER_TRANS queue table, with the IFACENAME value equal to the MXPOInterface value, and the TRANSID value equal to 1065.

- Three entries in the MXPO_IFACE interface table, each with the TRANSID value equal to 1065.

  In this case, each entry with the TRANSID value equal to 1065 has a unique secondary sequence number.

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.

The following diagram shows an example of the relationship between the interface queue tables and the interface table records. The interface tables contain both inbound and outbound transactions.
The data in the MXOUT_INTER_TRANS queue table directs the external system to process the interface table records in the following sequence.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Interface Table</th>
<th>Identifier (TRANSID) of Record in Interface Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MXPR_IFACE</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>MXITEM_IFACE</td>
<td>1002</td>
</tr>
<tr>
<td>3</td>
<td>MXINVENTORY_IFACE</td>
<td>1003</td>
</tr>
<tr>
<td>4</td>
<td>MXITEM_IFACE</td>
<td>1004</td>
</tr>
</tbody>
</table>

The data in the MXIN_INTER_TRANS queue table directs the integration framework to process the interface table records in the following sequence.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Interface Table</th>
<th>Identifier (TRANSID) of Record in Interface Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MXPO_IFACE</td>
<td>2007</td>
</tr>
<tr>
<td>2</td>
<td>MXITEM_IFACE</td>
<td>2008</td>
</tr>
<tr>
<td>3</td>
<td>MXINVENTORY_IFACE</td>
<td>2009</td>
</tr>
</tbody>
</table>

**TRANSSEQ Column**

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.
For example, if a purchase order has three line items, that transaction might produce the following records:

- One entry in the MXOUT_INTER_TRANS queue table, with the IFACENAME value equal to the MXPOInterface value, and the TRANSID value equal to 1065.

- Three entries in the MXPO_INTERFACE table:
  - One entry (PO line 1) with the TRANSID value equal to 1065 and the TRANSSEQ value equal to 1.
  - One entry (PO line 2) with the TRANSID value equal to 1065 and the TRANSSEQ value equal to 2.
  - One entry (PO line 3) with the TRANSID value equal to 1065 and the TRANSSEQ value equal to 3.

The TRANSSEQ column is only in the interface tables.

**EXTSYSNAME Column**

An interface table can contain both inbound and outbound transactions. The following table shows that the EXTSYSNAME column in the interface queue tables can contain inbound or outbound data.

<table>
<thead>
<tr>
<th>Value of EXTSYSNAME</th>
<th>Direction of Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>The external system that is the destination of the</td>
<td>Outbound</td>
</tr>
<tr>
<td>transaction.</td>
<td></td>
</tr>
<tr>
<td>The value of a valid and enabled external system</td>
<td>Inbound</td>
</tr>
<tr>
<td>that is defined in the integration framework.</td>
<td></td>
</tr>
</tbody>
</table>

**ACTION Column**

The following table shows the ACTION column values in an interface queue table and the corresponding action that is applied to the interface table.

<table>
<thead>
<tr>
<th>Value in ACTION Column</th>
<th>Integration Framework or External System Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Inserts the data that is provided in the message.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the parent data, and any related child data, from the database.</td>
</tr>
<tr>
<td>Change</td>
<td>Updates parent and child data contents of the message, but does not delete existing child data that is not explicitly specified in the message.</td>
</tr>
<tr>
<td>Replace</td>
<td>Replaces the existing records with the contents of the message and deletes existing child data that is not referenced in the message.</td>
</tr>
<tr>
<td>AddChange</td>
<td>If the primary record does not exist, processes as an add action; otherwise, processes as a change action.</td>
</tr>
<tr>
<td>Null</td>
<td>If the primary record does not exist, processes as an add action; otherwise, process as a replace action.</td>
</tr>
</tbody>
</table>
The system populates the ACTION column in the outbound messages. If the external system does not populate the column in the inbound messages, the integration framework tries to retrieve and replace the corresponding database record. If the record does not exist, the integration framework tries to add it to the database.

**IMPORTMESSAGE Column**

The IMPORTMESSAGE column holds any error message that was produced when the interface table row was moved to the inbound queue.

**TRANSLANGUAGE Column**

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 Column**

The MESSAGEID column is a unique identifier that the system assigns to every outbound transaction.

**IFACETBNAME Column**

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 Oracle Databases

Long description columns are stored in a CLOB (character large object) column in an Oracle database. However, 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>
<thead>
<tr>
<th>Data Type</th>
<th>Name of Description Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOB</td>
<td>PO_DESCRIPTION_LD</td>
</tr>
<tr>
<td>ALN</td>
<td>PO_DESCRIPTION_LD2</td>
</tr>
</tbody>
</table>

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, IFACETABLECONSUMER, polls the MXIN_INTER_TRANS queue table and uses the IFACENAME, EXTSYSNAME, and TRANSID values in the queue table to place the corresponding interface table records into the appropriate inbound JMS queue. Then, individual records in the JMS queue are processed.

The interface table polling process checks that the record 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 configurable parameters. All parameters are optional.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXITCLASS</td>
<td>Java exit class that enables the manipulation of data before it is written to an inbound queue.</td>
</tr>
<tr>
<td>ENDPPOINT</td>
<td>Endpoint that is associated with the interface table. The default value is the predefined endpoint value that points to the local database.</td>
</tr>
<tr>
<td>ENTERPRISESERVICE</td>
<td>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.</td>
</tr>
<tr>
<td>TARGETENABLED</td>
<td>Optional Boolean flag that controls whether the cron task runs on a specific application server. The default value is 0 (false).</td>
</tr>
<tr>
<td>EXTSYSNAME</td>
<td>External system to be polled.</td>
</tr>
<tr>
<td>QUEUETABLE</td>
<td>Enterprise service queue table. The default value is MXIN_INTER_TRANS.</td>
</tr>
</tbody>
</table>
Configuring External Systems

To use interface tables, you must create the tables and configure the IFACETABLECONSUMER cron task.

Overall Processing

To configure the external systems to perform general interface queue table and interface table processing:

1. Define procedures to restore the backups of the interface tables.
2. Create interface tables in the user interface.
3. Manage the archiving of interface tables.

Inbound Processing

To configure the external system to perform inbound interface queue table and inbound interface table processing:

1. Create and initialize the outbound TRANSID counter.
2. Create records for each interface table that an inbound transaction writes to:
   a. Create an interface table record and populate it with the following information:
      - The transaction data
      - The incremented TRANSID value
      - If multiple records exist for the same interface table, the incremented TRANSSEQ value
   b. Create an MXIN_INTER_TRANS queue record with the following information:
      - The same TRANSID value that is contained in the interface table record
      - The name of the enterprise service that corresponds to the interface table, in the IFACENAME column
      - Optional: The ACTION value
      - The identifier of the external system, in the EXTSYSNAME column
   c. Perform a single commit, to commit all records for a transaction at one time.

Inbound Null Columns

If a column in an interface table contains a null value, the applicable objects process the column in the following ways:

- By default, the field is not updated by the transaction.
- If you add the empty tag when you are using a user exit, the object adds a null value to the field in the system database.
Outbound Processing

To configure the external system to perform outbound interface queue table and inbound interface table processing:

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:
   
   a. Read the records in the MXOUT_INTER_TRANS queue table in the TRANSID sequence.
   
   b. Enable each record in the MXOUT_INTER_TRANS queue table:
      
      i. 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.
      
      ii. Process data according to the value in the ACTION column of the interface queue table.

   c. Commit all records for a single database transaction.

   d. Delete the current record from the MXOUT_INTER_TRANS queue table.

3. Implement error management, based on your external system requirements.
The Message Tracking application tracks and displays the processing history of queue-based enterprise service messages, and queue-based publish channel 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.
When you enable message tracking, the integration framework writes all processed messages to the MAXINTMSGTRK table. The system assigns a status to each message which represents its current position in the queue-based processing cycle. Individual message events are displayed in the Message Details table window.

When you enable message tracking, pre-existing queue messages (before the function was enabled) are not identified by the message tracking logic.

When you disable message tracking, pre-existing queue messages (before the function was disabled) are identified by the message tracking logic. New messages that are received in the queue are not identified by the message tracking logic.

Messages have the following attributes, and values are assigned based on the originating enterprise service or publish channel data:

- **Integration mode** - 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: SYNC, UPDATE, QUERY, DELETE, CREATE, and PUBLISH.

- **System** - Name of the external system that is associated with either the enterprise service or publish channel.

- **Integration component** - Name of the enterprise service or publish channel.

- **Adapter** - Name of the adapter that is associated with either the enterprise service or publish channel.

- **Queue name** - 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received Datetime</td>
<td>The date and time the message was received in the queue.</td>
</tr>
<tr>
<td>Message ID</td>
<td>Unique message identifier that is assigned by the integration framework.</td>
</tr>
<tr>
<td>Search ID</td>
<td>Message identifier that is assigned by an external application and that is used in the message searches.</td>
</tr>
<tr>
<td>External Message ID</td>
<td>Unique message identifier that is assigned by an external application.</td>
</tr>
</tbody>
</table>
The following attributes have dynamic values that change based on the transaction events.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Status</td>
<td>The most current processing status for the tracked message.</td>
</tr>
<tr>
<td>Status</td>
<td>The status that is associated with the individual message event in the transaction history.</td>
</tr>
<tr>
<td>Status Date</td>
<td>The status date for the individual message event in the transaction history.</td>
</tr>
<tr>
<td>Error</td>
<td>The error message for the individual error message event in the transaction history.</td>
</tr>
</tbody>
</table>

**Message Tracking Configuration**

You can maintain a record of processing actions for messages that are sent through publish channels or that are received from enterprise services. By using the Message Tracking action in the Publish Channels and Enterprise Services applications, you can track messages and perform the following functions:

- ▼ Enable or disable transaction tracking
- ▼ Store transaction messages on the application file server
- ▼ 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.

**External Message ID**

In the Message Tracking application, you can use the External Message ID attribute to locate specific messages. The syntax that you use to identify the node values should be a fully qualified XPATH expression.

When you use the Message Tracking action, you can use an XPATH expression to specify the location of your message identifier in the XML message. To guarantee uniqueness in the messages that are tracked, the integration framework assigns an external message identifier. The system stores the external message identifier as the EXTMSGIDFIELDDATA attribute in the MAXINTMSGTRK table.
To find all messages for the MXPERSONInterface enterprise service, create the following fully qualified XPATH expression as the External Message ID:

`/{http://www.ibm.com/maximo}SyncMXPERSON/@messageID`

**Multi-noun Message and External Message ID**

When a multi-noun inbound message is received, the integration framework uses the enterprise service External Message ID XPATH to identify the message. If the path 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**

When you use the Message Tracking action, you can use an XPATH expression to specify the location of an identifier in the inbound XML message. The syntax that you use to identify the node values must be a fully qualified XPATH expression. The nodes that are identified by the XPATH expression let you perform efficient multi-noun searches. The system stores the search identifier in the SEARCHFIELDDATA field in the MAXINTMSGTRK table.

To find all messages for the MXPERSONInterface enterprise service, create the following fully qualified XPATH expression as the Search ID:


**Multi-noun Message and Search ID**

When a multi-noun inbound message is received, the integration framework uses the enterprise service Search ID XPATH to identify the message. If the path 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.

**Stored Messages**

You can save the original message that you receive from an enterprise service or a publish channel definition. The system stores files in the txndata folder in the system global directory. Define the global directory location in the mxe.int.globaldir system property, in the System Properties application.

The naming convention of the stored messages is:

```
ExternalSystemName_IntegrationComponent_UniqueId.xml
```
Message Statuses

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 Statuses

Inbound messages can have the following statuses:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>Message processing failed due to validation problems.</td>
</tr>
<tr>
<td>DELETED</td>
<td>Message was deleted from the queue.</td>
</tr>
<tr>
<td>PROCESSED</td>
<td>Message was successfully processed.</td>
</tr>
<tr>
<td>RECEIVED</td>
<td>Message was successfully written to the inbound queue.</td>
</tr>
</tbody>
</table>

Outbound Message Statuses

Outbound messages can have the following statuses:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>Message processing failed due to validation problems.</td>
</tr>
<tr>
<td>DELETED</td>
<td>Message was deleted from the queue.</td>
</tr>
<tr>
<td>PROCESSED</td>
<td>Message was successfully processed.</td>
</tr>
<tr>
<td>RECEIVED</td>
<td>Message was successfully written to the outbound queue.</td>
</tr>
</tbody>
</table>

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
- STATUDATETIME
- ERRORMSGR
## Tracked Inbound and Outbound Events

The following inbound and outbound events update the MAXINTMSGTRK table:

<table>
<thead>
<tr>
<th>Event</th>
<th>Details</th>
</tr>
</thead>
</table>
| 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. |
| Error in message processing   | The existing record in the message tracking table. When the 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 system maintains the ERROR message status. |
| End-of-queue processing       | The following transaction processing events update the existing record:  

  ▼ The 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 system sets the message record status to DELETED. The message tracking table is no longer updated. |
Error Management

The integration framework uses Java Message Service (JMS) queues as a staging mechanism. Inbound messages from external systems are first stored in a queue, and then retrieved for processing. Similarly, outbound messages are stored in a queue, from which they are retrieved for dispatch to an external system.

Queue error management is initiated when an error condition is identified. The error identification occurs when a message is processing in an outbound or inbound queue. You can view, correct, cancel, and reprocess the erroneous messages. Error messages can be managed without the need to access error files on the application server.
Queue Error Management

Outbound processing errors can occur as the message is sent from the queue to the external system. These errors are typically caused by a communication failure between the queue and external system, or a problem with database configuration when writing to interface tables or files.

Inbound processing errors can occur when the integration framework tries to process a message into the application. Inbound processing errors are typically a result of business rule validations in the objects or in 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. As a result, only one error at a time can exist in a sequential queue.

The continuous queue processes messages in a multi-threaded mode. When an error occurs in the continuous queue, the error management mechanism 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 before determining that an error requires intervention. The system also performs the following activities when encountering an error:

- Sends a notification to a specified e-mail account, informing the recipient that an error occurred.

  On an IBM WebSphere Application Server environment, the integration framework sends an additional e-mail message to the specified e-mail account each time that you successfully restart your application server.

- Creates an error file that contains the original message and information about the error.

- Creates a record in the Message Reprocessing application.
Interface Table Error Management

Errors can occur in the following two stages when inbound interface tables are processed:

- Writing data from the interface table to the queue
- Processing data from the queue into the integration framework

The integration framework does not apply any business rules to the inbound interface table records as it writes them to the inbound JMS queue. Errors that do occur can occur for one of 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 e-mail notification to the administrator e-mail address that is specified in the System Properties application.

After the cron task processes subsequent records in the MXIN_INTER_TRANS queue table, it switches to an idle state. 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. The assumption is that the person who was notified of, and is researching, 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.
Error Management Configuration

Error management requires the configuration of the following properties before use.

System Properties Configuration

The following error management properties are configurable in the System Properties application.

▼ mxe.int.adminfromemail

This property specifies the from address that appears in error notifications that are submitted by the integration framework, for example, MXINTADM@ZZZ.com. Some SMTP servers require this address to be a valid e-mail address format; some servers accept any value.

▼ mxe.int.admintoemail

This property specifies the primary addresses to which the integration framework sends notification of message processing errors. You can enter multiple addresses delimited with a comma (,).

You can optionally use the e-mail address property at the queue level to override the administrator address. Use this option to specify different e-mail addresses for each queue.

If you do not configure an e-mail address property, no e-mail notification is sent when queue processing errors occur.

▼ mxe.int.globaldir

This property specifies the root folder under which all integration configuration files are located. If this value is null, the folders are created under the directory from which the application server is started or from the current working directory of the application server.

This location must be an existing folder that is accessible by a specified application server. In a clustered environment, all application servers must have access to this location.

▼ mail.smtp.host

This property specifies the host that runs the SMTP server. This property is not unique to the integration framework, and can be configured for other applications.
External Systems Configuration Properties

The following error management properties are configurable in the Add/Modify Queues action in the External Systems application:

▼ Maximum Try Count

This property specifies the number of times that the system tries to reprocess an error. After the specified number of tries, the system administrator receives a notification, and a message is written to an error file.

If the Maximum Try Count value is set to 0, 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.

For the WebSphere Application Server, valid values are any numbers that are equal to or greater than zero

▼ E-Mail Address

This optional property can be specified if you want e-mail 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 E-mail Address property. If no value is specified, e-mail notifications are sent to the e-mail addresses that are specified for the Administrator E-mail Address property.
When an inbound or outbound transaction results in an error in a queue, the application sends an e-mail notification to the system administrator only if no other errors are awaiting correction in the same queue. The notification informs the system administrator that the queue contains one or more transactions with errors. If multiple errors exist in the queue, the system administrator must resolve all of them before notification of new errors is sent.

An e-mail error message includes a Java error stack trace and the error file name with its full folder path information, which is relative to the server location. The following example shows a typical folder path:

C:\error\jms.maximo.int.queues.cqin\EXTSYS1\EXTSYS1_MXCOAInterface_11017328950785131.xml

The notification process works the same way regardless of the type of queue (continuous or sequential) or the processing direction (inbound or outbound) in which the error occurs, or whether the system is running in a clustered or non-clustered environment.

For example, assume that the continuous inbound queue contains ten messages. The system successfully processes the first four, and then encounters an error in the fifth. (Depending on the value of the Maximum Try Count property, the message might be tried one or more times.) An e-mail notification is sent to the system administrator, and subsequent messages in the queue are processed.

If another error occurs in the seventh message, another e-mail notification is not generated if the system administrator did not resolve the original error. If the system administrator resolved the original error and no errors are pending, a new e-mail notification is sent to the system administrator.

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 error is resolved.

Multiple errors exist only in the continuous inbound queue. In a clustered environment, the system administrator might receive one e-mail error notification per application server, depending upon the timing of the transactions in error.
Message Reprocessing

When you use the Message Reprocessing application, you can manage and view integration messages that are flagged with an error. You can view the error XML file without accessing the integration server error files.

Failed messages can be managed in the following ways in the Message Reprocessing application:

- Change the message statuses to stop message processing or to reprocess a message
- Correct, process, save, or cancel the error XML file
- Delete the message from the database

The Message Reprocessing application works with the Message Tracking application when you enable message tracking in either the Publish Channels application or the Enterprise Services application. In the Message Tracking application, you can determine which tracked messages are flagged with an error and then correct the erroneous data.

If you have not enabled message tracking, you can go to the Message Reprocessing application to check for transaction errors.

Message Statuses

You use the Change Status action in the Message Reprocessing application to change the status of a message. 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:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETRY</td>
<td>The message is ready to be reprocessed by the system.</td>
</tr>
<tr>
<td>HOLD</td>
<td>The message is not ready to be reprocessed by the system.</td>
</tr>
</tbody>
</table>

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.
You can view message details and change error message contents in the Message Details dialog box in the Message Reprocessing application. You can choose to process, save, or cancel your XML changes.

You can edit only the messages that are in a HOLD status. If the message has a RETRY status, the error data content is read-only.

**Error Details**

The error XML file contains the following elements and error content:

- The error message in the ERRORMESSAGE element
- The message from the queue in the ER element
- The object structure XML in the IR element

Only the ER element can be edited.

The IR element is present only for inbound transactions, and only if enterprise service processing and user exit processing was successfully applied to the message. The IR element represents the object structure created during enterprise service and user exit processing.

The IR element is provided for information only and any change to the IR is ignored when the message is reprocessed.

The following code is an example of an error message that contains both the ER and IR elements.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ERROR>
  <ERRORMESSAGE>Error occurred while processing PO (Object Structure number 1). Error is:[system:unknownerror] An unknown error has occurred. Please contact your system administrator for assistance. null</ERRORMESSAGE>
  <ER>
      creationDateTime="2007-05-22T14:04:03-04:00"
      transLanguage="EN" baseLanguage="EN"
      messageID="11798570432187483" maximoVersion="7.1 Harrier 038a HARRIER-042" event="1" messageid="11798570432652428">
      <MXPOSet>
        <PO action="Update">
          .
          .
        </PO>
      </MXPOSet>
    </SyncMXPO>
  </ER>
</ERROR>
```

```xml
<IR>
    creationDateTime="2007-05-22T14:04:03-04:00"
    transLanguage="EN" baseLanguage="EN"
    messageID="11798570432187483" maximoVersion="7.1 Harrier 038a HARRIER-042" event="1" messageid="11798570432652428">
    <MXPOSet>
      <PO action="Update">
        .
        .
      </PO>
    </MXPOSet>
  </SyncMXPO>
</IR>
```
HARRIER-042" event="1" messageid="11798570432652428">
  <MXPOSet>
    <PO action="Update">
      .
      .
      .
    </PO>
  </MXPOSet>
</SyncMXPO>
</IR>
</ERROR>

**Process Message**

After you complete your error XML changes in the Error Data window, you can resubmit the message. The application then starts reprocessing the message.

The following actions are performed when the message is successfully processed:

- The error file is deleted
- The record in the MAXINTERRORMSG table is deleted
- The DELETEFLAG, CHANGEBY, and CHANGEDATE attributes in the MAXINTERROR table are updated
- The record in the MAXINTERROR table is deleted

The application refreshes the result set and omits the successful message listing on the main tab of the Message Reprocessing application.

The following actions are performed when message is not successfully processed:

- The MAXINTERRORMSG table is updated with the new error description
- The CHANGEBY and CHANGEDATE attributes in the MAXINTERROR table are updated
- An error message is created

The application refreshes the result set and displays the new error for the unsuccessful message listing on the main tab of the Message Reprocessing application.

**Save Message**

After you complete your error XML changes in the Error Data window, you can save the message. The system then saves the message and updates the CHANGEBY and CHANGEDATE attributes in the MAXINTERROR table.

**Cancel Message**

After you edit your error XML, you can cancel the message changes. When the message is canceled, the database is not updated. The error XML file remains in its original state.
Common Causes of Errors

Message Deletion

You can choose to delete a message from a queue. After you delete a message, the system cannot reprocess it.

When the system deletes the message, 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.

Critical Errors

Critical errors are transaction 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. When the integration framework identifies a critical error, ER and IR sections in the corresponding error file are not present.

To correct the critical error, remove the incorrect data from the error XML. You can see incorrect data that is associated with a critical error in the main tab of the Message Reprocessing application.

Common Causes of Errors

Errors that occur when a message is processed from an outbound queue are related to the problematic delivery of a message to the endpoint that is specified for the external system. Typical problems are disruptions of the communication link between the system and the external application, or 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.

Errors that occur when a message is processed from an inbound queue are typically related to a business rule validation in an object or in the inbound processing of the enterprise service.

The following table describes the most common message errors that you can encounter. Perform the suggested actions to correct your message errors.

Correcting an error in the XML file might create a mismatch in data between the sending and receiving systems.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence error</td>
<td>Caused by sequencing problems between messages. The system stops message processing when one record is referencing another record that is in a pending state.</td>
<td>No action required. The system resolves the error.</td>
</tr>
<tr>
<td>Data error</td>
<td>Occurs because the data or record does not exist in the system database, and is not part of the inbound messages in the queue.</td>
<td>Add the missing data to the system database.</td>
</tr>
</tbody>
</table>
Common Causes of Errors

When you receive an error notification, look at the XML file in the Message Reprocessing application. Depending on the type of queue (sequential or continuous) and the number of messages in the queue, zero, one, or multiple error XML files might appear in the Message Reprocessing application.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication error</td>
<td>Caused by communication problems with the external system from system failures or network issues.</td>
<td>Restore communication with the external system.</td>
</tr>
<tr>
<td>Message error</td>
<td>Caused by erroneous message data values.</td>
<td>1 Change the transaction status to HOLD.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Correct the error XML file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 Change the transaction status to RETRY to reprocess the transaction.</td>
</tr>
</tbody>
</table>

Error Research

When you receive an error notification, look at the XML file in the Message Reprocessing application. Depending on the type of queue (sequential or continuous) and the number of messages in the queue, zero, one, or multiple error XML files might appear in the Message Reprocessing application.

No XML Files Exist

If no error XML file exists in the Message Reprocessing application, the message was retried and the error was not encountered again. The system deleted the error file.

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 system successfully reprocesses the message and successfully saves the data.

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 error file is deleted.

One or More XML Files Exist

When an error occurs in a sequential queue (inbound or outbound), processing of the queue stops until the error is resolved. One XML file exists in the error folder for the queue.

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 XML files exist in the Message Reprocessing application.

No ER and IR Elements in the Error File

If the corresponding error file contains no ER or IR element, the integration framework encountered a major exception.

The following example is an error XML file that the integration framework creates when it encounters a major exception:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ERROR>
  <ERRORMESSAGE>A major exception has occurred. Error on line 14: An invalid XML character (Unicode: 0x1b) was found in the element content of the document.</ERRORMESSAGE>
</ERROR>
```
Because the error is a major exception, the integration framework cannot retry the XML file. To correct the error you must remove the incorrect data from the source XML file. You then must delete the error record by changing its message status to DELETE. After you remove the invalid data from the source XML file, you can reprocess the message.

The integration framework provides a view queue utility to help you identify the application data content that caused the error. The utility creates a view queue file and places the file in the view folder under the integration global directory. The view queue file is identified with the same file name as the error XML file, and its data content includes the message that was initially received in the inbound queue.

For example, if a view queue utility error occurs after you export person data through the MXPERUSERInterface publish channel, the system generates a MX_MXPERUSERInterface file. The file contains the following original outbound message content.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<PublishMXPERUSER creationDateTime="2007-12-04T13:21:03-05:00" transLanguage="EN" baseLanguage="EN" messageID="11967924643281249" maximoVersion="7 1 082 V7100-001" event="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/maximo">
  <MXPERUSERSet>
    <PERSON>
      <ACCEPTINGWFMAIL>0</ACCEPTINGWFMAIL>
      <ADDRESSLINE1>760 Unity Blvd.</ADDRESSLINE1>
      <ADDRESSLINE2 />
      <ADDRESSLINE3 />
      <BILLTOADDRESS />
      <BIRTHDATE xsi:nil="true" /> <CITY>Framingham</CITY>
      <COUNTRY>USA</COUNTRY>
      <COUNTY>xxx</COUNTY> <DELEGATE />
      <DELEGATEFROMDATE xsi:nil="true" />
      <DELEGATETODATE xsi:nil="true" />
      <MAXUSER>
        <DATABASEUSERID />
        <DEFSITE>BEDFORD</DEFSITE> <DEFSITEROOME />
        <EMAILPSWD>0</EMAILPSWD> <FAILEDLOGINS>0</FAILEDLOGINS> <FORCEEXPIRATION>0</FORCEEXPIRATION> <INACTIVESITES>0</INACTIVESITES> <LOGINID>jennyb</LOGINID> <MAXUSERID>37</MAXUSERID> 
      </MAXUSER> 
    </PERSON> 
  </MXPERUSERSet> 
</PublishMXPERUSER>
```

The incorrect county on line 11 caused the major exception. The integration framework does not process the message until you correct the incorrect data.
Non-queue Error Management

Synchronous integration scenarios require continuous communication between the system and external applications. These synchronous scenarios do not require a messaging queue mechanism.

Non-queue error management is necessary when an error occurs when a message is being processed by the integration framework or by the system. Instead of relying on an error queue, the integration framework responds synchronously to the caller of the process with an error message.

When the system encounters a processing error, the calling application must receive the response, correct the error, and retry the transaction. You can use the system log as an aid in troubleshooting your synchronous transaction errors. The system log contains the processing exception that the integration framework issues to the caller of the process.
Non-queue Error Management
Basic Configuration

You must complete several configuration activities before you perform basic integration processing. Basic configuration involves working with predefined components, administration, and queue properties. You can either work with pre-existing integration components, or user-defined components to meet your business needs.

Before you start the basic configuration, create and configure the Java Message service (JMS) queues on your application server.
The following integration components are predefined:

- An external system (EXTSYS1)
- An adapter (MAXIMO)
- Object structures
- Enterprise services
- Publish channels
- Endpoints

In addition, the following JMS queues are defined:

- Inbound sequential (sqin)
- Inbound continuous (cqin)
- Outbound sequential (sqout)
- Error continuous (cqinerr)

You can use the predefined queues for development and production.

Prerequisite Activities

Before you can process integration framework transactions, create the following components:

- The object structures that you use
- The publish channels and endpoints to which the systems send outbound data
- The queues that the enterprise services use to receive data
The integration framework specifies the following system properties, some of which have default values that you should consider updating:

- System user name
- Global directory
- Update SENDERSYSID setting
- Administrator e-mail address
- Sender e-mail address

The following table describes the framework properties. You can update the following properties in the System Properties application. Some property values can have a true (1) value and some properties can have a false (0) value.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxe.int.dfltuser</td>
<td>Specifies the default login ID that is used in all inbound integration transactions that process in an inbound JMS queue. The user account that is configured in this property must be a valid system user account.</td>
</tr>
<tr>
<td>mxe.int.globaldir</td>
<td>Specifies the name of the directory to be used for all generated schema, XML, and error files that are in predefined locations in the global directory. This property has no default value. The noted files are in the same directory as the application server default directory.</td>
</tr>
<tr>
<td></td>
<td>If you specify an alternate location, the location must be accessible from all system application servers.</td>
</tr>
<tr>
<td>mxe.int.expupdatesender</td>
<td>Specifies whether the integration framework writes the system identifier (the value of MAXVARS.MXSYSID) to the SENDERSYSID field when it generates an outbound transaction from the Data Export feature. The default value is 0.</td>
</tr>
<tr>
<td>mxe.int.admintoemail</td>
<td>Specifies the system administrator e-mail address.</td>
</tr>
<tr>
<td>mxe.int.adminfromemail</td>
<td>Specifies the sender e-mail address.</td>
</tr>
<tr>
<td>mxe.int.uddiinquurl</td>
<td>Specifies the integration UDDI Registry Inquiry URL.</td>
</tr>
<tr>
<td>mxe.int.uddiname</td>
<td>Specifies the integration UDDI registry user ID.</td>
</tr>
<tr>
<td>mxe.int.uddipassword</td>
<td>Specifies the integration UDDI registry password.</td>
</tr>
<tr>
<td>mxe.int.uddipuburl</td>
<td>Specifies the integration UDDI registry publish URL.</td>
</tr>
<tr>
<td>mxe.int.webappurl</td>
<td>Specifies the integration Web application URL.</td>
</tr>
<tr>
<td>mxe.credentialmapperclassname</td>
<td>Specifies a processing class where the system credential mapping policy is registered. When you define a value, the system provides this class with user and endpoint information to be used to connect to the external service.</td>
</tr>
<tr>
<td></td>
<td>If a value is not defined, the system uses the user name and password that are specified in the endpoint to connect to the external service. The default value is null.</td>
</tr>
</tbody>
</table>
The following predefined JMS queues are provided by the system.

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cqin</td>
<td>Continuous inbound</td>
</tr>
<tr>
<td>sqin</td>
<td>Sequential inbound</td>
</tr>
<tr>
<td>sqout</td>
<td>Sequential outbound</td>
</tr>
<tr>
<td>cqinerr</td>
<td>Continuous inbound error queue</td>
</tr>
</tbody>
</table>

Ensure that the JMS queues are configured so that inbound and outbound message processing is successful.

**Configuring JMS Queues**

Configure JMS queues when you are working with the integration processes, based on the enterprise services and the publish channels, including data import and data export. Queue-based integration scenarios might require one or more JMS queues.

1. Define the queue properties in the system.
2. Enable the cron task for the sequential queues.
3. Enable the message bean for the continuous queue.
Queue Properties

The maximum try count property represents the number of times the system retries to process a failed message before sending an e-mail notification. The maximum try count default value is 5; the integration framework retries a failed message five times.

The integration framework tries to process a failed message as many times as specified by the maximum try count. If the message cannot be successfully processed after the specified number of retries, the integration framework changes the failed message status from ERROR to HOLD.

The error management retry mechanism uses the maximum try count parameter. When a message status is changed from HOLD to RETRY, the Error Management application retries its processing as many times as specified by the maximum try count parameter.

The default value for the count parameter property is 0. The queue processing framework attempts to process the error message indefinitely.

Cron Task for the Sequential Queues

The default settings direct the JMSQSEQCONSUMER cron task to poll the outbound queue, sqout, and the sequential inbound queue, sqin, every 30 seconds. You must activate the applicable instances of the cron task (SEQQIN, SEQQOUT), to avoid having the unprocessed inbound and outbound messages remain in the queues.

If you do not use one or both of these queues, do not activate these cron task instances.

Asynchronous Cron Task Inbound Processing

The XMLFILECONSUMER cron task and the FLATFILECONSUMER cron tasks must first be created, configured, and enabled before cron task message processing can occur.

XMLFILECONSUMER Cron Task Parameters

The XMLFILECONSUMER cron task is the mechanism that you must use to load XML files without any application user intervention.

The XMLFILECONSUMER cron task has the following predefined parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNALSYSTEM</td>
<td>A required parameter value that identifies the external system value that the cron task data load process uses.</td>
</tr>
<tr>
<td>SOURCEDIRECTORY</td>
<td>A required parameter value that defines the directory where source files are loaded. This directory must exist on the application server.</td>
</tr>
</tbody>
</table>
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.

### XMLFILECONSUMER Cron Task Processing Properties

Keep in mind the following cron task 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 XMLFILECONSUMER 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 split tag value and XML schema (when default schema is not being used) to identify the node in which the file split occurs.

- **Queue processing** - The XMLIN cron task identifies the queue in which the XML 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 processed.
committed to the queue. Index files are available in the RECOVERY folder which is created in the cron task source directory.

The XMLFILECONSUMER 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.

**FLATFILECONSUMER Cron Task Parameters**

The FLATFILECONSUMER cron task is the mechanism you must use to load flat files without any user intervention.

The FLATFILECONSUMER cron task has the following predefined parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCEDIRECTORY</td>
<td>A required parameter value that defines the directory where source files are loaded. This directory must exist on the application server.</td>
</tr>
<tr>
<td>DELIMITER</td>
<td>A required parameter that indicates the character that is used as field delimiter in the flat file. The default value is , (comma).</td>
</tr>
<tr>
<td>USEREXITCLASS</td>
<td>A custom processing class value to enable additional cron task capability (for example file load order).</td>
</tr>
<tr>
<td>TEXTQUALIFIER</td>
<td>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).</td>
</tr>
<tr>
<td>TARGETENENABLED</td>
<td>A required Boolean flag value that determines whether the cron task runs in a specific application server. The default value is 0 (false). When set to a value of 1, the application server where the cron task was created is defined as a dedicated server for that particular cron task instance.</td>
</tr>
</tbody>
</table>

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.

**FLATFILECONSUMER Cron Task Processing Properties**

Keep in mind the following cron task 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.
Use the Select Action menu in the External Systems application to perform the following configuration activities:

1. Configure external system integration controls and assign system-level default values.
2. Create external system interface tables.
3. Create and configure a queue.

Endpoints

Your outbound transactions (publish channels or invocation channels) for your business processes need endpoint values. The endpoints specify how the transactions are delivered and the handler routes the transactions to the endpoint.

Configuring Object Structures

Use the Object Structure tab in the Object Structures application to perform the following configuration activities:

1. Create an object structure.
2. Add the system objects to the object structure.
3. Assign the processing classes (inbound or outbound) to the object structure.
4. Specify whether the object structure supports a flat file representation.
5. Restrict object structure supported operations to QUERY.

Use the Select Action menu in the Object Structures application to perform the following configuration activities:

1. Configure the column aliases to resolve alias conflicts in the object structure.
2. Include and exclude the system object columns from the object structure.

Configuring Enterprise Services

Use the Enterprise Service tab in the Enterprise Services application to perform the following configuration activities:

1. Create an enterprise service.
2. Specify the object structure.
3. Assign the supported operation.
4. Assign a multiplication control.
5. Assign an interface table name.
6 Identify whether you are using an external schema.

7 Assign a split tag that you use to process multi-noun messages.

8 Assign the processing classes (request or response).

9 Create the processing rules.

A fully qualified XPATH expression must be provided in the Split Tag field when you use an external schema. The XPATH expression indicates where the integration framework splits the multi-noun messages with an XML definition.

For example, the following XPATH expression shows a Split Tag field value:

```
{http://www.ibm.com/xmlns/prod/tivoli/swkb}SoftwareCatalog/Products/Product
```

The bracketed content represents the namespace value. You must provide the namespace value for all tags that are assigned a namespace. In the example, the software catalog tag has an assigned namespace value.

Use the Select Action menu in the Enterprise Services application to perform the following configuration activities:

1 Create the integration controls.

2 Configure the enterprise service integration controls and assign default values.

3 Generate the schema files to describe the enterprise service.

Enterprise service configuration is required when you are working with external system and queue-based integration scenarios.

## Configuring Publish Channels

Use the Publish Channel tab in the Publish Channels application to perform the following configuration activities:

1 Create a publish channel.

2 Identify the object structure.

3 Assign the interface table name.

4 Assign the processing classes.

5 Create the processing rules.

Use the Select Action menu in the Publish Channels application to perform the following configuration activities:

1 Create the integration controls.
2 Configure the publish channel integration controls and assign the default values.

3 Generate the schema files to describe the publish channel.

4 Enable and disable the listener for the outbound integration events

Publish channel configuration is required if you are working with external system and queue-based integration scenarios.

**Configuring Invocation Channels**

Use the Invocation Channels tab in the Invocation Channels application to perform the following configuration activities:

1 Create an invocation channel.

2 Identify the object structures for request and response processing.

3 Assign an endpoint.

4 Assign the processing classes.

Use the Select Action menu in the Invocation Channels application to generate schema files to describe the invocation channel.
Advanced Topics

Chapter 10: Endpoints and Handlers
Chapter 11: Advanced Interface Table Polling
Chapter 12: JMS Queue Configuration
Chapter 13: Security
Chapter 14: Cluster Configuration
Chapter 15: Integration Framework Customization with Processing Rules
Chapter 16: Integration Framework Customization with Java and XSL
Chapter 17: Integration Component Additions and Modifications
Chapter 18: Integration Queries
Chapter 19: Integration Web Services
Chapter 20: Multiple Language Support
Chapter 21: Integration Modules
Chapter 22: Launch-in-Context Feature
Endpoints and their handlers facilitate the routing of outbound messages from the outbound Java Message Service (JMS) queue to its destination. Endpoints also route outbound data from an invocation channel directly to its destination. The endpoint and handler combination contains the protocol client and the necessary data to identify the specifics of communications with the destination, such as a URL.

The outbound queue cron task process or invocation channel invokes the handler and passes the message body and the metadata properties to it. The handler uses the metadata properties to determine the external system (for a publish channel message only) and override values for the configured endpoint properties. The handler then sends the data to the destination that is specified by the endpoint with which the handler is associated.
Endpoints

An endpoint represents an application component to which the system delivers outbound transactions. The following table lists the system available endpoints.

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXFLATFILE</td>
<td>FLATFILE</td>
<td>Writes flat files to a specified directory location.</td>
</tr>
<tr>
<td>MXIFACETABLE</td>
<td>IFACETABLE</td>
<td>Writes outbound transactions to local interface tables.</td>
</tr>
<tr>
<td>MXXMLFILE</td>
<td>XMLFILE</td>
<td>Writes XML files to a specified directory location.</td>
</tr>
<tr>
<td>MXCMDLINE</td>
<td>CMDLINE</td>
<td>Implements the CMDL 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.</td>
</tr>
</tbody>
</table>

Configuring Endpoints

Perform the following steps to configure an endpoint:

- Define the handler
- Define the endpoint
- Associate a handler to the endpoint
- Optional: Define handler properties for the endpoint

The handler defines the properties needed, and the endpoint provides the values for those properties. Use the predefined handlers or create new ones to enable additional physical entities, such as FTP servers.

For more information and detailed instructions for adding or modifying endpoints, refer to the online help in the End Points application.
Handlers

A handler specifies how to route outbound data to a specific endpoint location, in a specific format.

Enterprise Bean Handler

The Enterprise JavaBean (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 the target. The handler delivers system messages to the enterprise bean client. The target client can run on the local application server or a remote application server.

For the handler to establish a connection, the remote class and home 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.

Conversely, 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. This property 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 default exit runs the DefaultEJBExit. The system then 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, ? extends Map<String, ? extends 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.

```java
public void responseOk() throws MXException
```

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

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

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:

```java
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:

```java
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 applications/maximo/businessobjects/classes folder.
- 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

The following required property specifies the name by which the enterprise bean client is registered in the application server Java Naming and Directory Interface (JNDI) tree.

### 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.

**FLATFILE Handler**

The FLATFILE handler converts outbound data from the queue 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 that you associate with a publish channel must be marked as flat supported. Resolve all object structure alias conflicts and format the XML message according to the system schema to write the message to a flat file.

**Flat File Naming Convention**

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 file name EXTSYS1_MXASSETInterface_10971102668641498.dat indicates that the file was generated by the system to send data to the external system EXTSYS1. The file name also indicates that the file contains a message that was published through the MXASSETInterface publish channel.

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 Formatting

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

This handler has the following properties:

**FILEDIR Property**

This required property specifies the location of the flat file. The location must exist on the local server where the application server runs, or on a shared network drive.

This property value defaults to the Integration global directory/flatfiles folder when no value is specified. The mxe.int.globaldir value controls the integration global directory value.

**FLATFILESEP Property**

This required property 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 data 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.
This optional property is used for customization. This property 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. The Java class must be in the class path of the handler.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java class</td>
<td>DefaultHTTPExit.java</td>
</tr>
<tr>
<td>Package</td>
<td>psdi.iface.router</td>
</tr>
<tr>
<td>HTTPEXIT Property</td>
<td>psdi.iface.router.DefaultHTTPExit</td>
</tr>
</tbody>
</table>

The system provides a default implementation of the HTTP exit. The Java class is DefaultHTTPExit, which is in the psdi.iface.router package and implements the psdi.iface.router.HTTPExit interface.

The Java class has the following key methods:

- `processResponseData()`
  
  This method has the following signature:
  
  ```java
  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:
  
  ```java
  public Map<String, String> getURLProperties(Map<String, 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.
Handlers

- **getHeaderProperties()**
  
  This method has the following signature:

  ```java
  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:

  ```java
  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.

**HTTPMETHOD Property**

This optional property specifies the HTTP method that you use. The supported values are POST and GET. The default value is POST when no value is specified.

**CONNECTTIMEOUT Property**

This optional property specifies the connection timeout value in milliseconds.

**READTIMEOUT Property**

This optional property specifies the read timeout value in milliseconds.

**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 data from an outbound queue to an interface table in a local or remote database. There are no Java exits for this handler.

The FLATFILE handler can be used only with publish channels, not invocation channels. The object structure that you associate with a publish channel must be marked as flat supported. Resolve all object structure alias conflicts and format the XML message according to the system schema to write the message to an interface table.

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

**ISREMOTE Property**

This required property specifies whether interface tables are available in the local system database in the system schema or in a different schema. Its value can be 0 or 1.
A value of 0 (false) indicates that the interface tables are available in the local system database in the system schema. You do not have to enter any other handler properties. For 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 ISREMOTE is 1.

**URL Property**
This property specifies the JDBC URL. This property applies only when the value of ISREMOTE is 1.

```
jdbc:db2://mea6:5000/MERLIN
```

The example contains the location, port number, and database name.

**USERNAME and PASSWORD Properties**
If access to the database instance requires a user name and password, these properties specify those values. These properties apply only when the value of ISREMOTE is 1.

**JMS Handler**
The Java Messaging Service (JMS) handler is a Java component that consists of several properties. The handler delivers XML data into a messaging system that is enabled through JMS. The system places messages in a virtual channel called a queue or topic, based on your defined messaging model.

In the point-to-point messaging model, a sender generates messages and places them in a queue. Only one receiver can obtain the message from the queue. In the publish-subscribe messaging model, a publisher generates messages and places them in a topic. Multiple subscribers can retrieve the message from the topic.

The following diagram illustrates the messaging models.
The messaging system represents a queue or topic that is available on the local application server, a remote application server, or 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 are used by the system. The standard internal queues reside on the local application server where the system runs.

The JMS handler has the following properties:

**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.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic</td>
<td>javax.jms.Topic</td>
</tr>
<tr>
<td>Queue</td>
<td>javax.jms.Queue</td>
</tr>
</tbody>
</table>

**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 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not compress data</td>
<td>0</td>
</tr>
<tr>
<td>Compress data</td>
<td>1</td>
</tr>
</tbody>
</table>

Compressed messages must be decompressed after they are received. Decompress 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 Deflater() class.
JMSEXIT Property

This optional property is used for customization. It 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.

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.

The Java class must be in the class path for the application server or in the application EAR file.

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 WebSphere Application Server PROVIDERURL value.

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

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.

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.

WEBSERVICE Handler

The WEBSERVICE handler is a Java component that consists of several properties. The handler invokes a specified Web service with system data as a SOAP request parameter. This handler is Dynamic Invocation Interface (DII) based on BP 1.1, a document-literal compliant Web service.

The WEBSERVICE handler has the following properties:

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 sendreceive is used.

<table>
<thead>
<tr>
<th>Value</th>
<th>Web Service Operation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendreceive</td>
<td>Request and response</td>
</tr>
<tr>
<td>sendrobust</td>
<td>Request with void or fault response</td>
</tr>
<tr>
<td>fireandforget</td>
<td>Request only, no response, or fault</td>
</tr>
</tbody>
</table>

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 SOAPAction HTTP header to be used when invoking the Web service is called. The default value is an empty string value "". You can use the WSEXIT class to override the value specified in the user interface before you invoke the Web service.
Handlers

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
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.

XMLFILE Handler

The XMLFILE handler is a Java component that writes XML data into a file.

FILEDIR Property
This optional property specifies where the handler creates the XML files. The default value is mxe.int.globaldir/xmlfiles.

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_1097110268641398.xml indicates that the file was generated from a system to send data to the external system EXTSYS1. The file name also indicates that the file contains the MXASSETInterface publish channel.

**CMDLINE Handler**

The CMDLINE handler is a handler that 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.

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

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 – The boolean value 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

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
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.

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.

### Writing Custom Handlers

To write a custom handler, implement the RouterHandler interface. That interface has the following two methods:

```java
getParameter()
```

This method returns a list of properties that the handler needs to send data to the endpoint.

- The `RouterPropsInfo` object represents a property. This method can return 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.
The following method sends data to the specified endpoint.

- Metadata provides information about the external system and the interface.
- Data is the XML data.
- DestinationMap specifies the endpoint.

```java
sendData(Map metaData, byte[] data, Map destinationMap)
```

The handler class displays the properties for which you must specify values. The FTPHandler.java file in the psdi.iface.samples directory contains an example of the handler.
The system provides the option of using interface tables to exchange data with external systems. 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 Java Message Service (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.
**Cron Tasks**

The interface table polling process uses a single default cron task called IFACETABLECONSUMER. This cron task reads all messages from all interface tables for all external systems that write to the tables.

**Configuring Cron Tasks**

For improved performance on single-server and multi-server environments, you can configure multithreaded interface table polling by defining multiple instances of the IFACETABLECONSUMER cron task with different property values. Multithreaded polling is useful in a clustered configuration, because different threads can run on different servers, thereby balancing the load.

To designate an instance of the cron task named instance1 to run on a specific application server:

1. In the Cron Task Setup application, set the TARGETENABLED property to 1.
2. In the application server1 setup, set
   
   ```bash
   -DIFACETBCONSUMER.instance1=1
   ```

When you implement multiple cron tasks, you also must implement mutually exclusive selectors to avoid processing a message more than once.

For more information about cron tasks, refer to the *System Administrator Guide*, and the online help for the Cron Task Setup application.

**Selectors**

You use a selector to add a WHERE clause to a cron task. If you define multiple instances of the cron task, you must define selectors, so that each instance reads mutually exclusive interface table rows. For example, if the system exchanges data with two external systems, the first thread might poll one system and the second thread might poll the second system.

**Defining Selectors**

You define selectors by assigning values to the EXTSYSNAME (external system) and ENTERPRISESERVICE (enterprise service) parameters in the Cron Task Setup application. You can add an IN clause to a selector by entering a pipe-delimited set of values.

To direct a cron task to select only purchase order records for system EXTSYS1 from the queue tables:

1. Set ENTERPRISESERVICE=MXPOInterface
2. Set EXTSYSNAME=EXTSYS1
To list multiple interface names in the ENTERPRISESERVICE property:

- Set ENTERPRISESERVICE=MXP0Interface|MXPRInterface|…

Requirements for configuring multiple instances of the cron task:

- The selectors must be mutually exclusive, so messages are not processed multiple times.
- The selectors must retrieve all the enterprise service transactions that you use, so no messages are left unprocessed.

Queue Tables

One method of improving interface table processing is to use multiple queue tables. The MXIN_INTER_TRANS queue table is the default queue table. The IFACETABLECONSUMER cron task reads the MXIN_INTER_TRANS table and uses it as a driver to find and process the corresponding data in the interface tables.

You can improve performance by setting up multiple 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, or the message is processed multiple times.
You can use Java Message Service (JMS) queues to exchange data with an external system. When an enterprise service message is received, it writes the message to a JMS queue. Messages remain in an inbound queue until they are successfully processed into the system or deleted from the queue.

When the system sends publish channel messages, the message is written to a JMS queue. The message is then sent to an external system. Messages remain in an outbound queue until they are successfully processed or deleted from the queue.

Your JMS queue implementation can operate on a single application server or across a cluster of application servers. There are three default message queues:

- One outbound sequential queue and two inbound queues
- One sequential
- One continuous
Creating and Configuring a Queue

You can use separate queues, one for each external system that you use. Queues can be configured for use with publish channels and enterprise services that integrate with one or more external systems. Queue creation and configuration involves multiple steps.

1. Create and configure the message queue on the application server.

2. In the External Systems application, add properties to the queue.

The following properties are available for each queue.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue JNDI Name</td>
<td>References the Java Naming and Directory Interface (JNDI) name that is configured on the application server. A default value is provided.</td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>References the Connection Factory name that is configured on the application server. A default value is provided.</td>
</tr>
<tr>
<td>Initial Context Factory</td>
<td>A value that you must configure when you do not to use the default queues and do not use a provided application server.</td>
</tr>
<tr>
<td>Provider URL</td>
<td>A value that you must configure when you do not to use the default queues and do not use a provided application server.</td>
</tr>
<tr>
<td>User ID</td>
<td>The user ID that you configure when the queue is secured on the application server.</td>
</tr>
<tr>
<td>Password</td>
<td>The password that is configured when the queue is secured in the application server for the user ID.</td>
</tr>
<tr>
<td>Inbound</td>
<td>Identifies whether the queue is used for inbound processing. If the inbound value is null, the system uses the queue for outbound processing.</td>
</tr>
<tr>
<td>Sequential</td>
<td>Identifies whether the queue is a sequential queue. If the sequential value is null, the system uses the queue for continuous queue processing.</td>
</tr>
<tr>
<td>Compress</td>
<td>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.</td>
</tr>
<tr>
<td>Maximum Try Count</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

Use the System and Enterprise Services tabs in the External Systems application.
3 Configure your external system and enterprise services to use the queues.

4 To enable message processing, you must configure the consumers value of the message queue.

The system processes the messages after they are written to the queue.

## Sequential Queues

The sequential queue is a JMS queue with a system cron task as a consumer. 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 file 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.

The following table describes the cron task parameters that you can configure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGEPROCESSOR</td>
<td>Java class that processes the messages from the queue. The system provides this class.</td>
</tr>
<tr>
<td>QUEUENAME</td>
<td>Queue JNDI name, when the queue is created on the application server.</td>
</tr>
<tr>
<td>SELECTOR</td>
<td>The WHERE clause for configuring an instance of the cron task to process a subset of messages in the queue. This parameter is optional.</td>
</tr>
<tr>
<td>TARGETENABLED</td>
<td>The Boolean flag that controls whether the cron task runs in a specific application server. The default is 0 (false).</td>
</tr>
<tr>
<td></td>
<td>This parameter applies only when the system is running a multi-application server environment.</td>
</tr>
</tbody>
</table>

## Continuous Queue

The continuous queue is a JMS queue with a message-driven bean (MDB) as a consumer.

By default, this queue is configured for enterprise services only. The queue does not process messages in first-in-first-out sequence, as the sequential inbound queue does. Messages are processed in a multi-threaded mode, which produces better system performance.
When message processing results in an error, the system generates an error file, then continues processing subsequent messages in the queue. There is one default continuous queue to process inbound messages.

### Enabling Message Beans

To enable the message beans, uncomment the following lines in the `ejb-jar.xml` file which is located in the `...\applications\maximo\mboejb\ejbmodule\META-INF\` folder. The code changes apply to both IBM WebSphere Application Server and WebLogic Server environments.

```xml
<!-- 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>
-->

Uncomment the following lines in the
`...\applications\maximo\mboejb\ejbmodule\META-INF\ibm-ejb-jar-bnd.xmi` file on a WebSphere Application Server environment.

```xml
<!-- MEA ejb for MDB
<ejbBindings xmlns:type="ejbbnd:MessageDrivenBeanBinding" xml:id="MessageDrivenBeanBinding_1"
  activationSpecJndiName="intjmsact">
  <enterpriseBean xmlns:type="ejb:MessageDriven" href="META-INF/
    ejb-jar.xml#MessageDriven_JMSContQueueProcessor_1"/>
</ejbBindings>-->

Uncomment the following lines in the
`...\applications\maximo\mboejb\ejbmodule\META-INF\weblogic-ejb-jar.xml` file on a WebLogic Server environment:

```xml
<!-- MEA MDB
<weblogic-enterprise-bean>
  <ejb-name>JMSContQueueProcessor-1</ejb-name>
  <message-driven-descriptor>
    <pool>
```

Continuous Queue Performance

You can improve performance by increasing the number of message-driven beans for a particular queue and by introducing additional application servers in a cluster. Because messages are processed in a multi-threaded mode, 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. By default, the server is configured to have five message-driven beans.

WebSphere Application Server

To configure the number of beans on your WebSphere Application Server:

1. In the administrative console, select JMS activation specification.
Continuous Queue

2 Select `intjmsact`.

3 Specify a value in the **Maximum Concurrent End Points** field.

**WebLogic Server**

By default, the WebLogic Server is configured to have three message-driven beans. Use the values shown in bold to configure the number of message-driven beans in the weblogic-ejb-jar.xml file:

```xml
<!-- 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>
-->
```

Start with a relatively low amount of message-driven beans, for example two to three message-driven beans. You then determine whether that amount provides proper support for the volume of messages in your system. You can incrementally increase the number of message-driven beans until you are satisfied with the processing performance of the messages in the continuous queue.

The number of message-driven beans and the volume of messages processing in a queue can use enough system resources to affect your application users. If you continue to increase the amount of message-driven beans, this change may have little or no impact on message processing performance. You can resolve some system performance issues by clustering your servers and isolating the inbound message processing to a specific server cluster.

**Message Caching**

Continuous queue processing uses a property called Maximum Batch Size. The property is under the Activation Specification definition. The batch size controls the number of messages that the system receives from the messaging engine in a single batch.

If three message-driven beans are enabled, and the batch size is 10, a maximum of 30 messages are 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 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.
Continuous Queue Errors

You can use different application server configurations to manage error messages. When you are loading a large volume of messages, you can encounter a large number of errors. The continuous queue continues to reprocess the messages at the top of the queue that are in error instead of new messages that are being added to the queue.

For example, you can run a nightly GL account update process that does not complete successfully. The following day you find that the GL accounts are not in the system and a high volume of system messages are generated. When a high volume of messages occur, the queue processing continues to work on the same messages and does not attempt to process new messages. You can avoid this situation by using server-specific configurations for your queues.

WebSphere Application Server

You can configure a continuous queue to have a corresponding error queue. If a message in the continuous queue encounters an error, the message is moved out of the continuous queue to the error queue. The system moves the message when the number of retries set in the maximum failed deliveries parameter is met. The continuous queue then processes new queue messages.

Configuring the Error Queue

To configure an error queue:

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 previously defined in step 1.

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.

You must control how often the message reprocessing occurs to avoid excessive system resource use. The integration framework message-driven bean implementation provides an MDBDELAY property that delays the processing of the messages, based on the property value.

When you configure the MDBDELAY property, the system retries the messages in the error queue in a controlled manner. At some point, your message can be successfully processed in a controlled environment.

Edit the ejb-jar.xml file to configure the message-driven bean delay. The default value is 30 seconds (30000 milliseconds).
To enable the message-driven beans on the error queue, uncomment the values shown in bold in the ejb-jar.xml file and set the MDBDELAY value.

```xml
<!-- MEA MDB for error queue
<message-driven id="MessageDriven_JMSContQueueProcessor_2">
  <ejb-name>JMSContQueueProcessor-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>psdi.iface.jms.QueueToMaximoProcessor</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>
</message-driven>
-->
```

You also need to uncomment the following code in the ibm-ejb-jar-bnd.xmi to enable the message-driven beans on the error queue.

```xml
<!-- MEA MDB for error queue
<container-transaction>
  <method>
    <ejb-name>JMSContQueueProcessor-2</ejb-name>
    <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
-->
```

**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.
For example, when 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.

As an alternative to setting the batch size property to -1, 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.

Message Header

The header can contain the following standard JMS header values:

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSMessageID</td>
<td>A message ID that is generated by the system.</td>
</tr>
<tr>
<td>JMSRedelivered</td>
<td>Identifies whether the message was reprocessed.</td>
</tr>
</tbody>
</table>

Message Properties

The properties contain the following mix of properties from the JMS provider and the integration framework. The integration framework properties are of the string data type.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAMessageID</td>
<td>The message ID that is generated by the integration framework.</td>
</tr>
<tr>
<td>destjndiname</td>
<td>The name of the queue or topic that the message is written to.</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>The name of the publish channel (outbound queue) and the enterprise service (inbound queue).</td>
</tr>
<tr>
<td>destination</td>
<td>The external system name for outbound messages.</td>
</tr>
<tr>
<td>SENDER</td>
<td>The external system name for the inbound messages.</td>
</tr>
</tbody>
</table>
Queue Selectors

The message body contains the XML message that is processed into the system or sent to the external system.

### 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.

<table>
<thead>
<tr>
<th>Type of Queue</th>
<th>Where to Identify Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential queue</td>
<td>As a property of the cron task</td>
</tr>
<tr>
<td>Continuous queue</td>
<td>In the ejb-jar.xml of the message-driven bean</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Description</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicates whether the message is compressed. Values can be true or false; the default value is false.</td>
<td>compressed</td>
</tr>
<tr>
<td>The standard Java Inflater and Deflater APIs (java.util.zip) are used for compression.</td>
<td>uncompressed_length</td>
</tr>
<tr>
<td>Stores the original message payload size before compression. This value must comply with the schema type int [xsd:int] and is present only when the compressed property is set to true.</td>
<td>MSG_TRK_ENABLED</td>
</tr>
<tr>
<td>The endpoint name for the outbound messages, fixed string MXJMS for inbound messages.</td>
<td>MSG_OP_MODE</td>
</tr>
<tr>
<td>Internal value.</td>
<td>MSG_TRK_STORE_MSG</td>
</tr>
<tr>
<td>External system name for outbound and inbound messages.</td>
<td>MSG_TRK_EXTSYS</td>
</tr>
<tr>
<td>A field name message key.</td>
<td>Msgkeyval</td>
</tr>
<tr>
<td>A comma-separated search field value.</td>
<td>searchfieldval</td>
</tr>
<tr>
<td>Indicates whether the publish channel or enterprise service contains a sync, create, update, or delete operation.</td>
<td>msgoperation</td>
</tr>
<tr>
<td>Indicates whether the message has a RECEIVED, ERROR, DELETED, or a PROCESSED status value.</td>
<td>msgstatus</td>
</tr>
<tr>
<td>Contains the exception message text.</td>
<td>msgerrmsg</td>
</tr>
</tbody>
</table>
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 might 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:

```plaintext
SENDER='EXTSYS1' and INTERFACE in ('MXPOInterface', 'MXPRInterface')
```

Add the following content to the message bean configuration (ejb-jar.xml) to instruct the message-driven bean to process the MXPOInterface and MXPRInterface transactions from the corresponding external system:

```xml
<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.

## Queue Utilities

You can see the properties and the number of messages in a queue, but not the content of the messages. The system provides the following utilities for viewing and deleting messages in the queues.

<table>
<thead>
<tr>
<th>Utility</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewqueue</td>
<td>viewqueue.bat and viewqueue.sh</td>
</tr>
<tr>
<td>deletequeue</td>
<td>deletequeue.bat and deletequeue.sh</td>
</tr>
</tbody>
</table>

The viewqueue utility places the data in the view folder under the integration global directory.

Use the deletequeue utility in a test or development environment. The Message Reprocessing application provides a mechanism for deleting individual messages from the production queues.
The following property values are required utility values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required or Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dqueueename</td>
<td>Queue JNDI name</td>
<td>Required</td>
</tr>
<tr>
<td>-Dserviceurl</td>
<td>RMI URL of the application server</td>
<td>Required</td>
</tr>
<tr>
<td>-Dusername</td>
<td>J2EE user (if the queue is secured)</td>
<td>Optional</td>
</tr>
<tr>
<td>-Dpassword</td>
<td>J2EE user password (if the queue is secured)</td>
<td>Optional</td>
</tr>
<tr>
<td>-Dselector</td>
<td>The WHERE clause to process a subset of messages in the queue.</td>
<td>Optional</td>
</tr>
<tr>
<td>-Dcount</td>
<td>The number of messages to be viewed or deleted; an asterisk means all messages (based on selector)</td>
<td>Optional</td>
</tr>
</tbody>
</table>

IBM WebSphere MQ

You can use IBM WebSphere MQ as a standard for your queuing activities. The integration framework supports WebSphere MQ Version 6. Before you use the WebSphere MQ for any queuing activities, you must familiarize yourself with WebSphere MQ configurations and implementations.

Configuring JMS Endpoints and Handlers

The framework processing places the outbound publish channel message into the default queue. The integration framework then uses the JMS endpoint and handler to send the message to WebSphere MQ.

To accomplish this task you must configure the endpoint JMS handler to point to the WebSphere MQ. This configuration can be done in following ways:

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:
   - The destination JNDI name (DESTJNDINAME)
   - The connection factory JNDI name (CONFACTORYJNDINAME)

2. Configure the endpoint to point to the WebSphere MQ by using:
   - The destination JNDI name (DESTJNDINAME)
   - The connection factory JNDI name (CONFACTORYJNDINAME)
   - The provider URL (PROVIDERURL)
   - The initial context factory (CONTEXTFACTORY)
Configuring the Integration Queues and WebSphere MQ Provider

You can replace the integration framework queues with the WebSphere MQ queues. The queue setup is supported when you configure the WebSphere Application Server and the integration framework queue definitions.

To configure the integration queues and the WebSphere MQ:

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 box 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.

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).
Some security mechanisms are provided to prevent the unauthorized use of system components. You can enable security as needed, based on your implementation.

Security can be set at a user level; user identification and password authentication is necessary for queue access. Security can also be set at a component level, which controls access to component or method calls.
Integration Queue

The Java Message Service (JMS) queues that are used by integration processing support basic J2EE security (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. Multiple queues can use the same or different user identifications. You can set these J2EE restrictions by using the server administrative console.

Configuring J2EE Restrictions

The following property names must be configured to enable J2EE restrictions.

- java.naming.security.principal (for user identification)
- java.naming.security.credentials (for password)

To let the system integration producer and consumer programs access the queue, enter the same user ID and password in the Add/Modify Queues dialog box in the External Systems application. You access this dialog from the Select Action menu.

For the continuous queue, update the message-driven bean (the consumer of the continuous queue) deployment descriptors as follows, to access a secured queue.

1. Under the <enterprise-beans> section in the ejb-jar.xml file, add the following elements shown in bold text:

```xml
<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>
</enterprise-beans>
```
2 Under the `<assembly-descriptor>` section in the `ejb-jar.xml` file, add the following elements shown in bold text:

```xml
<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>
```

---

**Enterprise Bean Access**

The integration gateway implements an Enterprise JavaBean (EJB) as the entry point for receiving data from an external system. An enterprise bean supports the basic J2EE security.

**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.

1 Change the ALLOWDFLTLOGIN value to 0 (false), to force system authentication.

Under the `<enterprise-beans>` section of the `ejb-jar.xml` file, three integration EJBs (enterprise service, object structure service, and standard service) are deployed with a default value of 1, no authentication.

The `<ejb-name>` to service mapping is:

<table>
<thead>
<tr>
<th><code>&lt;ejb-name&gt;</code></th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>enterpriseservice</td>
<td>Enterprise Service</td>
</tr>
<tr>
<td>mosservice</td>
<td>Object Structure Service</td>
</tr>
<tr>
<td>actionservice</td>
<td>Standard Service</td>
</tr>
</tbody>
</table>

The following code illustrates that the enterprise service is deployed with an ALLOWDFLTLOGIN value of 1.

```xml
<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>
  </session>
</enterprise-beans>
```
<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>1</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>1</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>1</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: secureProcessExtrenalDataAsync(..), secureProcessExtrenalDataSync(..)
- Object Structure Service: secureProcessMOS(..)
- Standard Service: secureAction(..)

To create a secure context for invoking the enterprise bean, do either one of the following procedures:

- Add the following sample code to the client code:

  ```java
  Properties env = new Properties();
  
  if(userid != null && password != null)
  {
      env.put(Context.SECURITY_CREDENTIALS, password);
      env.put(Context.SECURITY_PRINCIPAL, userid);
  }

  Context ctx = new InitialContext(env);
  //instead of using the default InitialContext() constructor
  ```

- Use the default InitialContext constructor to pass the security information through the –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 Internet Inter-ORB Protocol performs data encryption in the provider URL, while the system communicates with the enterprise bean.

**HTTP Servlet**

The HTTP servlet is a J2EE component that handles inbound HTTP posts to the system integration and follows the J2EE security principles. To secure the HTTP servlet, you first secure the enterprise bean.

**Securing the HTTP Servlet**

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.

To enable HTTP basic authentication, modify the web.xml file of the Web application:

1. Remove the comments from the <security-constraint> section of the integration servlets. One per service type. There are three security-constraint
blocks, one for each type of service: enterprise service, object structure service, and standard service.

The <web-resource-name> to service mapping is:

<table>
<thead>
<tr>
<th>&lt;web-resource-name&gt;</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Service Servlet</td>
<td>Enterprise Service</td>
</tr>
<tr>
<td>App Service Servlet</td>
<td>Standard Service</td>
</tr>
<tr>
<td>Object Structure Service Servlet</td>
<td>Object Structure Service</td>
</tr>
</tbody>
</table>

The following integration servlet code samples have the security constraint comment sections removed for each service type.

<!--
<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 guarantee</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 guarantee</description>
  <transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
  <web-resource-name>Service Servlet</web-resource-name>
  <description>Enterprise Service Servlet Object Structure Service Servlet Object Structure Service

HTTP Servlet
The preceding `<security-constraint>` section refers to a single role, maximouser, which is defined farther down in the web.xml file. By default, the security constraint section is not commented out.

In addition, change the following web.xml value for useAppServerSecurity from 0 to 1 in the web.xml:

You can securely deploy a Web service by using SSL (HTTPS). Set up the SSL in the application server with the appropriate digital certificates.

**Web Services**

You can secure integration Web services by using HTTP basic authentication in standard J2EE security. These security settings let authorized users with a valid user name and password access Web services.

**Securing Web Services**

To enable Web service security, use the steps previously described for HTTP, referencing the following Web service-specific security-constraint blocks.

The application web.xml file contains a `<security-constraint>` section for the web service invocation. By default, this section is commented out.
Remote Integration APIs (MicService)

1. Use the following the security-constraint block for a Web service servlet.

   `<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 guarantee</description>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
   </security-constraint>

2. For Web service invocation, ensure that the client program uses the following user name and password calls in the JAX-RPC Call object:

   ```java
   call.setProperty(Call.USERNAME_PROPERTY, username);
   call.setProperty(Call.PASSWORD_PROPERTY, password);
   ```

   You can securely deploy a Web service by using SSL (HTTPS). Set up the SSL in the application server with the appropriate digital certificates.

Remote Integration APIs (MicService)

Some MicService remote APIs have a secure access by forcing the user of those methods to provide the UserInfo object. If you do not provide a valid UserInfo object, an error occurs and the call is not completed.

The following remote methods that provide sensitive information or perform sensitive data transaction processing have been system protected.

- `exportData(..)`
- `deleteQueueData(..)`
- `processExternalData(..)` (both versions)
- `query(..)`
- `viewQueueData(..)`
- `loadData(..)`
- `loadSystemData(..)`
Interface Table Security

- 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 (user, password, locale, language, and time zone information). The system uses the UserInfo object for security purposes.

The system uses Java RMI/JRMP. You can communicate to the system services by using a secure version of JRMP protocol using SSL.

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.

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.

Object-Level Authorization

The system provides object-level authorization based on the security configuration set within the system. If an object or attribute is marked as read-only or hidden, then inbound message data processing is limited to data object queries. You cannot insert, update, or delete data in that object. The authorization level for a business object and object attribute can be configured in the Data Restrictions tab in the Security Groups application.

Except for standard services, integration messages are not processed according to application-level authorization.

The authorization that is used for system users controls the standard service authorization. A signature option can be assigned to a standard service to limit authorization to the users or groups that have authorization for a selected option.
For example, you can assign the status option to the changeStatus method of the PO standard service. To run the standard service by using HTTP, enterprise bean, or SOAP, you need authorization to the STATUS option.

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.
The system can be implemented within a cluster of application servers, and integration services can run across the cluster. You can implement a cluster of application servers based on the size of your system implementation.

First, determine whether a single server implementation of the continuous queue can handle your message volume. If a single server does not meet your needs, add central processing units, hardware, and a cluster configuration to improve message processing performance.

When you implement a 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.
Configuring the Cron Task

The interface table cron task, the data import cron task, and the Java Message Service (JMS) queue cron task are cluster-aware functions. By default, the cron task framework runs a task on a randomly chosen server.

If you want to configure your cron task to run on a specific application server within your cluster, configure the cron task as shown in the following example. The name of the cron task instance is instance1.

1. In the Cron Task Setup application, set the TARGETENABLED property for instance1 to 1.

2. In the application server1 setup, set the –D crontaskname.instance1 to 1, and restart the server.

In this configuration example, the task is not pushed to another server if the targeted server stops. To replace a stopped server, set the –D crontaskname.instance1 value to 1, then restart the application server that resumes the polling.

To configure a group of servers to support failover, set the –D crontaskname.instance1 value to 1 in each server. The integration framework prevents more than one instance of the cron task from running at the same time.
In a cluster configuration, the JMS queues are associated with one member of the server cluster. Access to the queues is provided by the Java Naming and Directory Interface (JNDI) service. The JNDI service is available across all the members on the cluster as shown in the following diagram.

The sequential queue is accessible to all cluster members to produce messages into the queue. Additionally, messages are read by a single-threaded cron task to support first-in-first-out processing.

The continuous queue is multi-threaded on the consumer side. The queue supports 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.
The following diagram shows one example of a cluster configuration on the IBM WebSphere Application Server.

The diagram is an example of a continuous queue integration bus configuration in a cluster of three application servers. To gain the benefit of multi-server processing within a cluster, each member of the cluster needs to be configured with its own message engine.

The processing occurs when the integration framework receives enterprise service messages by using the HTTP, EJB (Enterprise JavaBeans), and SOAP actions. The load balancer directs the enterprise service messages to an application server, which then places the message into the continuous queue. Each application server in the cluster places messages into its own message store.

Enterprise service messages that are sent using the Data Import feature process messages into the application server queue store, which limits the processing to a single server.
Integration cron tasks deliver enterprise service messages from flat or XML files and interface tables. The cron task places messages into the queue store on the server where the cron task is running, which limits the processing to a single server.

To configure integration cron tasks to place messages into multiple queue stores on different application servers:

- **Interface tables** – Enable multiple instances of the interface table cron task to different servers and use the cron task properties to select different groups of data. You can use the External System, Interface, and Queue table properties to select data.

- **Import XML file data** – Enable multiple instances of the XML file import cron task to different servers and use the cron task properties to select different files to import. Use the source directory and file name properties to select the import files.

- **Import flat file data** – Enable multiple instances of the flat file import cron task to different servers. Use the cron task property source directory to select different files to import.

After messages are in the queue, all application servers can simultaneously pull messages from their respective queue stores and process them into the system. Message-driven beans must be enabled on each application server in the cluster.
The following diagram shows one example of a cluster configuration on the WebSphere Application Server. The sequential queue receives JMS producer messages and processes the JMS consumer messages.

A sequential queue integration bus cluster consists of three application servers and one message engine on a single application server.

**ATTENTION**

Do not implement the sequential queue on the continuous queue integration bus. When you use the continuous queue integration bus for sequential queue implementations, messages may be unprocessed or may be processed out of order.

The load balancer directs the application server to drop enterprise service messages into the sequential queue. When messages are in the queue, the application server that is running the JMS consumer cron task processes messages sequentially.
Unlike the continuous queue, there is no multi-threading of messages. A cluster implementation does not significantly impact the processing performance of messages that are processed through the sequential queue.

**WebLogic Server Continuous Queue Cluster**

The following diagram shows one example of a cluster configuration on the WebLogic 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.

**WebLogic Server Sequential Queue Cluster**

The following diagram shows one example of a cluster configuration on the WebLogic Server. The sequential queue receives JMS producer messages and processes the JMS consumer messages.

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.

**Dedicated Message Processing Servers**

High volumes of system inbound messages can cause the application server to run slowly, impacting online application users. One way to improve server efficiency is to move the integration framework inbound message processing to a separate application server or cluster. A separate server or cluster that is dedicated to inbound processing enables the system to process higher volumes of messages without compromising application usability.

**Server Cluster Configuration**

To configure a server cluster that is dedicated to processing inbound messages, deploy a separate application EAR file and the target-enabled cron tasks for polling the inbound queues, interface table, and inbound file processing to the server cluster.

If applicable, configure the message-driven beans to pull data from the inbound continuous queue. You can run other background processing cron tasks on this server cluster. You can grant user interface access to users who must make use of the integration framework Data Import feature.

When you configure separate inbound processing, do not enable message-driven beans for inbound processing within the server or cluster that is dedicated for online users.

**Global Directory**

In a cluster environment, the global directory must be accessible to all members of the cluster. To define the global directory, update the mxe.int.globaldir system property in the System Properties application.

**Inbound Message Receipt**

All object structure, enterprise, and standard services can be accessed by using the following components:

- Java™ remote method invocation (RMI), Internet Inter-ORB Protocol™ (IIOP), and enterprise beans
- HTTP and HTTPS Servlet
- SOAP, HTTP, and HTTPS Servlet
The noted components are standard J2EE components which are deployed as part of the application EAR file. All these components are cluster-aware.

**Enterprise Beans**

With a single server, the provider URL for accessing the JNDI tree is the single server URL. With a cluster, the provider URL is the URL of any one of the servers that has the enterprise beans deployed. All members of the cluster share the JNDI tree, so any member of the cluster can look for and retrieve a cluster.

As a result of the 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 the 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, the URL is the HTTP and HTTPS URL of that server. With a cluster, 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 value represents the mxe.int.webapp.url system property value. You configure this system property value in the System Properties application.

<table>
<thead>
<tr>
<th>Service</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Structure Service</td>
<td><a href="http://hostname:port/meaweb/os/object">http://hostname:port/meaweb/os/object</a> structure name</td>
</tr>
<tr>
<td>Enterprise Service (bypassing queue)</td>
<td><a href="http://hostname:port/meaweb/es/extsysname/enterprise">http://hostname:port/meaweb/es/extsysname/enterprise</a> service name</td>
</tr>
<tr>
<td>Enterprise Service (through the queue)</td>
<td><a href="http://hostname:port/meaweb/esqueue/extsysname/enterprise">http://hostname:port/meaweb/esqueue/extsysname/enterprise</a> service name</td>
</tr>
<tr>
<td>Standard Service</td>
<td><a href="http://hostname:port/meaweb/ss/standard">http://hostname:port/meaweb/ss/standard</a> service name</td>
</tr>
</tbody>
</table>

**Integration 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:

- Integration Web Application URL `mxe.int.webappurl`
- Integration UDDI Registry Inquiry URL `mxe.int.uddiinurl`
- Integration UDDI Registry Publish URL `mxe.int.uddipuburl`

The URL to access a Web service is `http://hostname:port/meaweb/services/web service name` where the meaweb value represents the `mxe.int.webappurl` system property value.
You can use processing rules to change the behavior of predefined integration processing without having to write Java classes. You can implement processing rules on outbound messages of a publish channel or inbound messages of an enterprise service. You cannot use processing rules in invocation channels, object structure services, or standard 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.
Objects and Records

Objects and Records

An object structure consists of one or more records that correspond to system objects. 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.

Object Rule Definitions

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 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 objects, the processing time for inbound messages 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.

After you select the applicable service or channel, the application displays the records that comprise the corresponding object structure. Select the record on which the processing rule applies.

To define a processing rule, specify the record or object to which the rule applies.

Processing Rule Initiation

A system database action (event) on the primary object within the corresponding object structure initiates a processing rule. Use one or more of the following actions to initiate the rule:

- An insertion to the primary object
- A deletion to the primary object
- An update to the primary object

You can place the rule on a primary record or a child record or an object. However, the database action that initiates a rule must occur on the primary object that is associated with the object structure.

To prevent users from changing the information about an existing PERSON record, place a processing rule with a Stop action on the PERSON object. Specify that the rule that is to be applied when the primary PERSON object is updated. This rule performs the stop action if a user changes any attribute on a person record.

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 action settings. The system also runs all processing rules regardless of the settings that you specify for the processing rule properties.

For inbound transactions, flags apply only to enterprise services with an operation of Sync. Flags are evaluated against the action attribute of the primary object that is represented in the XML message.

Processing Rule Action

A processing rule can act on an enterprise service or 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 manipulate the value in a data field within a service or channel message: combine, split, set, and replace.

Message Processing Actions

The following processing rule actions stop or skip the enterprise service or publish channel message as a whole, or skip entire records within the message.
Processing Rule Definitions

**SKIP action**

The skip action bypasses a message that meets the specified criteria. The inbound message is not processed and the outbound 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.

For inbound messages, the message is cleared from the inbound queue, as it would like a successfully processed message. For outbound messages that are skipped, nothing is written to the queue.

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 the specified criteria. The 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.

Whenever possible, use the skip action rather than the stop action for inbound enterprise services. The latter 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.

For example, if a processing rule with a stop action applies 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.

**SKIPCHILDREN action**

The skip children action is available only for 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 e-mail, use the skip children action on a person to strip the phone and e-mail 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 e-mail information.

**SKIPRECORD action**

The skip record action is only available for 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 their child records and objects, that you want the system to skip.

For example, if the person 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 when the external system needs a work phone number but not the home phone number. The skip record rule needs a condition to identify a home phone record to force the system to skip that record.

**Field Transformation Actions**

Apply the field transformation rule to the 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.
The following field transformation actions manipulate data fields within a message.

**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, you 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 an external system and the system have different identifiers for the same entity. For example, the plant identifier of an external system might translate to your site identifier.

For example, to replace the SITEID value in a publish channel with an external PLANTID value, and to replace the external PLANTID value in an enterprise service with a SITEID value, you can use a single cross-reference control.

**COMBINE action**

The combine action concatenates values from multiple source fields into a single target field. When you define the rule, you identify the target field and the source fields. You also identify the sequence in which the source data is to be written to the target field. 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, for example, a two-part external system key that maps to a single part key in the system.

An enterprise service processing rule can combine a vendor ID and a vendor location field from an external system into the COMPANY field. A publish channel processing rule with the split action can then 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.

### Fixed-length Data 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.

### Variable-length Data Segments

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 one or more of the following evaluations or comparisons:

- XML field data
- Object field
- Object set
- Integration control
- 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. For example, if an evaluation checks whether the values of two fields are equal, 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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML field</td>
<td>Evaluate a value in an integration object record field, or compare the values in two record fields.</td>
</tr>
<tr>
<td>Object field</td>
<td>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.</td>
</tr>
<tr>
<td>Object set</td>
<td>Check for the existence of records in a related object.</td>
</tr>
<tr>
<td>Control</td>
<td>Evaluate a value or boolean integration control or a system control.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Direction of Processing Rule</th>
<th>XML Field Evaluation</th>
<th>Object Field Evaluation</th>
<th>Object Set Evaluation</th>
<th>Control Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outbound</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>Inbound (record)</td>
<td>Available</td>
<td>Not available</td>
<td>Not available</td>
<td>Available</td>
</tr>
<tr>
<td>Inbound (Object)</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Type of Evaluation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQUALS</td>
<td>The value in the specified field is equal to the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>NOTEQUALS</td>
<td>The value in the specified field is not equal to the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>GREATER</td>
<td>The value in the specified field is greater than the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>GREATEROREQUAL</td>
<td>The value in the specified field is greater than or equal to the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>LESS</td>
<td>The value in the specified field is less than the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>LESSOREQUAL</td>
<td>The value in the specified field is less than or equal to the value of a second field (the comparison field).</td>
</tr>
<tr>
<td>LIKE</td>
<td>The value contains the expected value.</td>
</tr>
<tr>
<td>NOTLIKE</td>
<td>The value does not contain the expected value.</td>
</tr>
<tr>
<td>ISNULL</td>
<td>The specified field contains no value or a null value.</td>
</tr>
</tbody>
</table>
When to Evaluate the Field

For XML field and object field evaluations, the processing rule first determines whether it evaluates the specified data. The system evaluates the data by checking the Evaluate When field, which can have one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGED</td>
<td>The evaluation continues only if the activity that generated the message changes the specified field.</td>
</tr>
<tr>
<td>NOT CHANGED</td>
<td>The evaluation continues only if the activity that generated the message does not change the specified field.</td>
</tr>
<tr>
<td>ALWAYS</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

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 user interface displays subsets depending on the type of evaluation (XML field, object field, object set, or control).
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration control</td>
<td>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.</td>
</tr>
<tr>
<td>Value</td>
<td>Compare the value in the specified field with a predefined value. This option is available for user-defined conditions. 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.</td>
</tr>
<tr>
<td>MAXVAR</td>
<td>Compare the value in the specified field with the value in a system control (a value in the MAXVARS database table). Evaluate the OWNERSYSID on any enterprise service or publish channel to determine if it is the same as MAXVARS.MXSYSID.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Compare the value in the specified field with a Boolean value (true or false). Example: Compare the GLDEBITACCT value and GLCREDITACCT value on a PO line or a journal entry to determine if they are equal.</td>
</tr>
<tr>
<td>Comparison field</td>
<td>Compare the value in the specified field with another field in the same object. Example: Check the OWNERSYSID of inventory in the system for the item-storeroom values on a receipt line or a PO Line.</td>
</tr>
<tr>
<td>Object relationship, and field</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 system master data and documents are stored at the system level, organization level, or site level. For example, item data is stored at the system level, accounting information at the organization level, and storerooms, inventory, and work orders at the site level. An implied hierarchy exists among these levels, because organizations are defined for a system, and sites are defined within organizations. Accordingly, an integration control can be configured to override values at any of the following levels:

<table>
<thead>
<tr>
<th>Control Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-level</td>
<td>A system-level value applies to all system organizations and sites. If the control is not configured for organization- 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.</td>
</tr>
<tr>
<td>Organization-level</td>
<td>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.</td>
</tr>
<tr>
<td>Site-level</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
Integration Controls

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.
**Integration Controls**

**Multiplication Control**

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 Creations**

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.
Integration Framework
Customization with Java and XSL

The integration framework provides multiple placeholders within the transaction flow to customize transactions by using Java programs and XSL. Based on your requirements, you can use one or more of these placeholders, or user exits, for your customizations.

You can download sample Java classes or XSL files from the IBM® Tivoli® Open Process Automation Library (OPAL).

After changing the Java class files, rebuild and redeploy the application EAR file. You can reference the XSL file in the application EAR file or in a directory path in your file system.
Outbound Customization

The integration framework processes outbound messages by using either a publish channel or invocation channel. Publish channels and invocation channels use placeholders, or user exits, in the outbound processing flows.

Publish Channel

The integration framework processes asynchronous outbound messages by using publish channels. Publish channels provide placeholders, or user exits, in the outbound processing flows for customization.

You can customize a publish channel in the following ways:

- Publish channel processing rules.
- In the following Java exits:
  - First user exit (user exit preprocessing)
  - Processing class
  - Second user exit (user exit postprocessing)
- With XSL mapping
- In a handler exit

The following diagram shows the flow of a message that uses a publish channel. The process flow starts from the object structure to the endpoint. The endpoint delivers the message to its destination.

Invocation Channel

The customization of an invocation channel is comparable to the publish channel. On the invocation side, the invocation channel has an XSL file and three Java exits: preprocessing user exit, processing class, and postprocessing user exit. The same level of customization is available to the transaction response component. The exit layer can map the response from the external service back to an object structure which can be used to update system business objects.

Processing rules are not available to an invocation channel.
Outbound Customization

The primary difference between an invocation channel and a publish channel is that application code invokes the invocation channel, while an event or a data export initiates a publish channel. Additionally, the synchronous processing of an invocation channel can return a response and update the system business objects.

For invocations, you can customize an invocation channel in the following ways:

- In the following Java exits:
  - First user exit (user exit preprocessing)
  - Processing class
  - Second user exit (user exit postprocessing)
- With XSL mapping
- In a handler exit

For responses, you can customize an invocation channel in the following ways:

- In the following Java exits:
  - First user exit (user exit preprocessing)
  - Publish channel processing class
  - Second user exit (user exit postprocessing)
- With XSL mapping

The following diagram illustrates the flow of a message that uses an invocation channel. The process flow starts from the object structure to the endpoint. The endpoint delivers the message to its destination and a response can be returned to the invoker of the invocation channel.

**Processing Rules**

You can use the publish channel processing rules engine to customize outbound transactions without writing Java code or using XSL.

Processing rules are available only to a publish channel, not to an invocation channel.
Java Exits

The following figure illustrates the hierarchy of standard Java exit classes for outbound processing. These Java classes are available to both the publish channel and the invocation channel.

The hierarchy illustration does not apply to processing rules, XSL, and handler exits.

Integration processing for publish channel messages can be based on system events or initiated events by using the Data Export feature. Publish channels subscribe to these events and receive an event notification when a system transaction is saved using the business objects.

Publish channels subscribe to these exit events and receive an event notification when a transaction is saved based on the business objects. An integration XML message is created based on the publish channel object structure when the Java exits run. Both the XML message and the business objects can be referenced. The data in the business object is saved, but not committed, so the Java exit cannot update the values of the business object.

You can access a business object that is not part of the publish channel object structure by using any relationship from an object available in the exit. You can also instantiate a new object from the database as needed. You can use either option to retrieve additional data from the system. You then can map the data to the outbound XML message so that it is available to the external systems.

User Exit Preprocessing

The user exit class has two methods: one exit runs before and one exit runs after the execution of the channel processing class. In the first user exit, the input is the XML, as generated by the object structure. If you have an ERP adapter processing class, use either the preprocessing method or postprocessing method to run your custom logic depending upon your requirement.

If necessary, you can change data in the message to alter the processing logic in the publish channel processing class.

Outbound message customization can be done in the first user exit by using the following method:

```java
public StructureData setUserValueOut(StructureData irData)
```

This method can perform the following processing:

- Validate data
Outbound Customization

- Change system data by changing the IR record to be sent to the external system
- Stop the message from being saved in the database by throwing a system exception. The entire message is rolled back, including any changes you make in the application that initiated the event.
- Stop the message from being sent to the external system by throwing a SKIP_TRANSACTION exception
- Log the transaction

Processing Class

If you use a provided adapter, the publish channel processing class contains the processing logic. You can implement your code in the preprocessing or the second postprocessing exit. If your customization is dependant upon the logic of the adapter, you must implement your custom logic in the postprocessing or after exit class.

If you create a publish channel, use the following method to customize the publish channel processing class.

public StructureData setDataOut(StructureData sData)

The following method can perform the noted processes:

- Validate data
- Change system data by changing the IR record to be sent to the external system
- Stop the transaction from being saved in the database by throwing a system exception
- Stop the message from being sent to the external system by throwing a SKIP_TRANSACTION exception
- Log the transaction

You identify the publish channel processing class on the Publish Channel tab in the Publish Channels application.

User Exit Postprocessing

In the postprocessing method of the user exit, both the IR (XML from object structure) and ER (XML output from the processing class) records are available for processing. Use the following method to customize outbound messages.

public StructureData setUserValueOut(StructureData irData, StructureData erData)
The following method can perform the noted processes:

- Validate data
- Change data by changing the ER record to be sent to the external system
- Map additional data from the IR to the ER
- Stop the transaction from being saved in database by throwing a system exception
- 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 on the Publish Channel tab in the Publish Channels application.

**XSL Mapping**

The framework provides a placeholder to implement an XSL file. You can use the XSL file to manipulate the data that is sent to the external system after the Java exit processing completes.

The XSL file is always called with the ER XML message. You identify the publish channel XSL mapping on the Publish Channel tab in the Publish Channels application.

The XSL file can be part of the EAR file and must reside in the businessobjects/classes/... folder structure. You also can define the XSL file on a publish channel by using a directory file path that is not part of the EAR file.

Register the XSL file mapping in the following manner when the XSL file mapping.xsl resides in the businessobjects/classes/psd/iface/xsl folder.

```
psdi.iface.xsl.mapping
```

The .xsl extension is omitted when the mapping.xls is registered.

Register the XSL file mapping in the following manner when the XSL file mapping.xsl is located in a file directory that is accessible by the application server. The format is specific to the operating system.

```
c:/psdi/iface/xsl/mapping.xsl
```

The .xsl extension is not omitted when the mapping.xls is registered.

**Handler Exits**

Multiple predefined methods (handlers) are provided so that you can send data to an external system, including HTTP, Enterprise JavaBean (EJB) call, and interface tables. Some of these methods have user exit placeholders available for customization.
The HTTP Processing Exit Class

You can use an HTTP exit placeholder for customization when sending transactions from the system to an external system using HTTP. This exit class is optional and is called as part of the response from the HTTP call.

The following figure shows the HTTP processing exit class that the system runs when a response is returned to the HTTP Post from an external system.

In the default implementation, psdi.iface.router.DefaultHTTPExit, the response code received from the external system is compared to a range of response codes. The range 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 if the external system accepted the message or not. 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 in 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.

This class must implement the psdi.iface.router.HTTPExit interface and implement the following method:

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

If necessary, this class can interpret the response code and throw an exception.

You enter the fully qualified name of the Java class in the HTTPEXIT property of the endpoint that implements the HTTP handler.

HTTPResponseUser

This class does the following processing to analyze the response from the HTTP post:

- Checks 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.
The Enterprise Bean Processing Exit Class

You can use an exit placeholder for class customization when transactions are sent to an external system using an enterprise bean. This exit class is optional and is called before the enterprise bean is called.

The following figure shows the various enterprise bean processing exit classes the system uses when sending transactions to an external system.

![Enterprise Bean Processing Exit Classes diagram]

The implementation of this Java class must resolve the method signature of the enterprise bean that is being invoked through 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:

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

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

The following method can perform the noted processes:

- Create an array of parameter values that correspond with the signature of the external enterprise bean method to be called.

```java
public Object[] getObjectParams(byte[] data, String interfaceName, String destinationName)
```

The parameters of the enterprise beans are returned in the form of an array of Java Objects.

The following method can perform the noted processes:

- Returns callback after the successful invocation of the enterprise bean with an object as the response of the invocation.

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

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

The following method can perform the noted processes:

- Returns callback after the unsuccessful invocation of the enterprise bean.

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

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

You enter the fully qualified name of the Java class in the EJBEXIT property of the endpoint that implements the enterprise bean handler.
The Java Message Service Processing Exit Class

You can use an exit placeholder for customization when sending transactions from the system to an external system using Java Message Service (JMS). This exit class is optional and is called before the JMS is called.

The following figure shows the JMS processing exit class the system uses when sending transactions to an external system.

This class must implement the `psdi.iface.router.JMSEXIT` class and the following method:

```java
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.

The Web Service Processing Exit Class

You can use an exit placeholder for customization when sending transactions from the system to an external system using a Web service. This exit class is optional and is called before the Web service is invoked.

The following figure shows the Web service processing exit class the system uses when sending transactions to an external system.

This class must implement the `psdi.iface.router.WSExit` interface and the following methods:

```java
public String getServiceName(Map metaData, String endpointURL, String serviceName, String interfaceName, String targetNameSpace) throws MXException
```

The `getServiceName()` method returns the service name of the Web service to be invoked.

```java
public String getEndpointURL(Map metaData, String endpointURL, String serviceName, String interfaceName, String targetNameSpace) throws MXException
```

The `getEndpointURL()` method returns the endpoint URL of the Web service to be invoked.
public void responseOk(org.w3c.dom.Document response) throws MXException

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

public void responseError(Exception e) throws MXException

If an error was encountered during the Web service invocation, the **responseError()** method is called with the originating exception as a parameter.

public boolean getOneWayWsInfo(Map metaData, String endpointURL, String serviceName, String interfaceName, String targetNameSpace, boolean oneWayWs) throws MXException

The **getOneWayWsInfo()** method returns a Boolean value that specifies whether the Web service being invoked is one-way.

public String getSoapAction(Map metaData, String endpointURL, String serviceName, String interfaceName, String targetNameSpace, String soapAction) throws MXException

The **getSoapAction()** method returns the SOAPAction HTTP header to be used while invoking the Web service.

You enter the fully qualified name of the Java class in the WSEXIT property of the endpoint that implements the Web service handler.

There is a default implementation of the WSExit interface, psdi.iface.router.DefaultWSExit. This class overrides the getEndpointURL() method to concatenate the service name at the end of endpoint URL, to form the new endpoint URL.
Inbound Customization

The integration framework processes inbound messages by using enterprise services. Enterprise services provide placeholders, or user exits, in the inbound processing flows for customization.

You can customize an enterprise service in the following ways:

- In an interface table user exit
- In the following Java exits:
  - First user exit (user exit preprocessing)
  - Enterprise service processing class
  - Second user exit (user exit postprocessing)
- With XSL mapping
- With object structure and object processing rules
- In a business object user exit

The following diagram shows the flow of a message that uses an enterprise service. The process flow starts at the receipt of a message that uses one of multiple protocols supported by the framework, through to the setting of the data in the object structure and system business objects. The enterprise services that use XML can bypass the JMS queue.

Interface Table User Exit

The external system can deliver an enterprise service message by using interface tables. 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 following figure shows the interface table processing exit classes that the external system uses when sending transactions to the system.
The interface table polling program is run by using the system cron task manager. The cron task, IFACETABLECONSUMER, has an optional property called EXITCLASS, where you can place 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, with each map representing 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 (Integers, Date, Float, Double) are converted to their localized string format before they are set in the map.

The EXITCLASS class must implement the psdi.iface.intertables.IfaceTbExit interface and the following three methods:

```java
public void beforeQueue(long transid, String extSys, String ifaceName, List data, Connection conn)
```

This method is called after the data is pulled from the interface table and before the data is inserted into one of the inbound queues.

```java
public void afterCommit(long transid, String extSys, String ifaceName, Connection conn)
```

This method is called after the data has been inserted to an inbound queue and deleted from the interface queue table, and the database commit is done.

```java
public void afterRollback(long transid, String extSys, String ifaceName, Connection conn)
```

This method is like the afterCommit method but called if the transaction is rolled back.

This class can perform the noted 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 in the Configuration module.

Java Exits

The following figure illustrates the hierarchy of standard Java exit classes for inbound processing. These Java classes are available to the enterprise service.

The hierarchy illustration does not apply to processing rules, XSL, or interface table exits.

When the preprocessor, processor, and postprocessor Java exits are run, the business objects for the enterprise service message have not been created. The business object user exit can take a processing action against the business objects that are created for the incoming message. However, in all exits you can instantiate a new object from the database to perform lookups or gather additional system information as part of your custom logic.

User Exit Preprocessing

The user exit class consists of two methods: one exit runs before and one exit runs after the enterprise service processing class. If you have an ERP adapter that provides a processing class, your user exit logic can be implemented to run before or after the ERP adapter logic.

In the first user exit, the input is the XML message that is delivered from the external system. If necessary, you can change data in the message that affects the processing logic in the processing class.

Inbound transaction customization can be done in the first exit using the following method:

```java
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
Inbound Customization

- Stop further processing of the transaction by throwing an exception
  In this case, the transaction remains in the queue to be retried.

- Stop the message from being sent to the external system by throwing a skip_transaction exception
  In this case, the message is not saved in the system; the message is removed from the queue.

- Log the transaction

You identify the interface user exit class on the Enterprise Service tab in the Enterprise Services application.

Enterprise Service Processing Class

If you use a provided adapter, the enterprise service processing class contains the processing logic. You must implement your code in the first or second user exit.

If you create an enterprise service, customization can be done in the enterprise service processing class by using the following method:

```java
public StructureData setDataIn(StructureData sData)
```

This method can perform the following processes:

- 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 throwing an exception. In this case the transaction remains in the queue to be retried

- Stop the message from being processed into the system by throwing skip_transaction exception
  In this case, the message is not saved in the system; the message is removed from the queue.

- Log the transaction

You identify the interface processing class on the Enterprise Service tab in the Enterprise Services application.

User Exit Postprocessing

In the second exit, both the IR and ER are available for processing. Inbound transaction customization can be done in the second exit using the following method:

```java
public StructureData setUserValueIn(StructureData irData,
StructureData erData)
```
The following method can perform the noted processes:

- Validate data
- Change external data by changing IR record to be saved in the system
- Map additional data from the ER to the IR
- Stop further processing of the transaction throwing an exception
  In this case, the transaction remains in the queue to be retried.
- Stop the message from being sent to the external system by throwing a skip_transaction exception
  In this case, the message is not saved in the system; the message is removed from the queue.
- Log the transaction

You identify the enterprise service user exit class on the Enterprise Service tab in the Enterprise Services application.

The Java user exits also provide an exit point that runs just before the object is saved in the system.

**XSL Mapping**

The enterprise service has a placeholder to implement an XSL file and to manipulate the data to be set to the object structure after the Java exit processing is completed.

The XSL file is always called with the XML message that is output from the Java exit processing. You identify the XSL file on the Enterprise Service tab in the Enterprise Services application.

The XSL file can be a part of the EAR and must be under the businessobjects/classes/... folder structure. The XSL file can also be referenced using a directory file path which is not part of the application EAR file.

For example, use the following reference to register the XLS when the file mapping.xsl is under the businessobjects/classes/psd/iface/xsl folder:

```
psdi.iface.xsl.mapping
```

The .xsl extension is omitted when the mapping.xsl is registered.

Register the XSL file mapping in the following manner when the XSL file mapping.xsl is located in a file directory that is accessible by the application server. The format is specific to the operating system.

```
c:/psdi/iface/xsl/mapping.xsl
```

The .xsl extension is not omitted when the mapping.xls is registered.
Object Structure and Object Processing Rules

For information about the processing rules engine, see Chapter 15, "Integration Framework Customization with Processing Rules," on page 199.

Business Object User Exit Processing

The Java user exits provide an exit point that runs before the objects that are created from the object structure XML are saved in the system. This object user exit is called after the system processing and can run against 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 noted processes:

- Validate data
- Stop the transaction from being saved in the system by throwing a system exception
  In this case, the transaction remains in the queue and is retried.
- Log the transaction

You identify the interface user exit class on the Enterprise Service tab in the Enterprise Services application.
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.
Object Structure Creations

The integration framework provides predefined object structures for enterprise services, publish channels, and invocation channels across multiple functional areas. The functional areas include purchasing, work management, and inventory. You can create an object structure, an enterprise service, and a publish channel to support the exchange of data to and from the integration framework.

Required Object Determinations

To determine what data to include in an object structure, you need some knowledge of the system applications and objects. Begin by examining 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 system 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.

For example, assume that you must send a person object to and from the system, but the system does not provide a predefined object structure with person data (the system does provide a predefined purchase requisition object structure, MXPERSON). After you use the person application, you determine that the data that you need resides in the following tables: PERSON, PHONE, EMAIL, SMS.

The tables have the same name as the corresponding objects. Include the PERSON, PHONE, EMAIL, and SMS objects in the object structure that you create. The resulting enterprise service and publish channel contain the data fields in those objects.

Building the Object Structure

After you identify the necessary objects, create the object structure on the Object Structure tab in the Object Structures application. Perform the following activities to build an object structure that contains the person object.

This procedure creates a standard, hierarchical object structure. An object structure can have the same object more than once in its definition. However, the objects must have a valid parent-child relationship, and you cannot reference the object more than once in the same hierarchical structure.

XML requires that the name of the object structure must begin with an alphabetic character.

1 Enter PERSON as the primary (top-level) source object, then PHONE as the child object. PERSON becomes the parent of PHONE.

2 Select the predefined object relationship that contains the appropriate WHERE clause that links the PERSON and PHONE object. If multiple relationships exist, select the WHERE clause that applies to the relationship that you are creating.
3 Repeat the preceding steps for the remaining required objects. Indicate the relationship between those two objects. Define PERSON as the parent to EMAIL and as the parent to SMS.

Alternate Keys

An inbound message normally uses the primary key of an object to look up and process records in the system. However, sometimes a primary key is an internally generated value that is not available to an external system. In such cases, the external system passes an alternate key, and the system uses that value to process the object.

Specifying an alternate key

To specify an alternate key for your object:

1 Identify the fields that comprise the alternate key and create a unique index on the object.

2 Specify the new index as the alternate key of your object.

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

For example, the primary key of the PERSON object consists of PHONE and EMAIL. The system normally tries to find the PERSON object by using the PHONE and EMAIL values on an inbound message. If the object structure indicates an alternate key, the system uses that value, instead of the primary key, to access the object.

Required 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 add nonpersistent object structure columns and exclude persistent object structure columns.

Persistent Fields

By default, the system includes in object structures all persistent columns in the component objects. A persistent column is a data field that an object uses to write data to a database table after processing. You can exclude persistent columns that you are not mapping.

Only the included persistent columns are written to 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.

Nonpersistent Fields

By default, the system 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. The object structure includes the former persistent column and excludes the latter nonpersistent column.
By default, nonpersistent columns are excluded in the object when the object structure is created. If you modify the object structure, you can include new nonpersistent columns to the object definition in the object structure.

**Interface Table and Flat File Considerations**

If you use the object structure in interface tables or flat files, check whether duplicate column names exist in the object structure. If duplicate names exist, modify the alias value for the duplicate columns. 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.

**Predefined Object Structure Modifications**

You can modify predefined object structures, subject to the following restrictions.

**Content Objects of an Object Structure**

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

**Object Columns**

You can include and exclude persistent and nonpersistent columns within the scope of the standard validations.

Outbound messages include the columns for objects that you add to an object structure. Test inbound messages to ensure that the added object columns are processed successfully. If the object columns are not processed successfully, add an enterprise service processing class to handle the inbound processing.

**Interface Table and Flat File Considerations**

If you use the object structure in interface tables or flat files, check for object structure column duplications.

If you use interface tables, regenerate the table for every enterprise service or publish channel that uses the modified object structure.

**Endpoint Creations**

An endpoint identifies a location and the processing logic of data publication. In the defined endpoint handler, you can identify how to route outbound data to a specific location. You also can define which data format the integration framework uses by using the handler.

**Handler**

A handler specifies how to route outbound data to a specific endpoint location in a specific format.

When you create a handler, you identify the Java class file that contains the associated processing logic. You then associate the handler with the endpoint location when you create an endpoint.
You can modify a user-defined handler. You cannot modify or delete a predefined handler. You can delete a handler only if you do not associate it with an endpoint for an external system.

Predefined Endpoint Modifications

You can change the property values of an endpoint to meet your individual business needs. You can change the value field to identify the endpoint property.

You cannot modify the End Point, Handler, or Consumed By fields after the initial save.

Adapter, Service, and Channel Creations

Services and channels exchange object structure data with the external system.

Adapters

You can create an adapter, or you can use the predefined MAXIMO adapter. If necessary, you can create multiple adapters that group services and channels to meet your business needs.

Services

You can create new services for the predefined and user-defined adapters. The services that you create determine how the adapter processes inbound integration messages.

Enterprise Services

The enterprise service specifies the type of operation that the service supports. Processing rules change data in the enterprise service before objects are created. You can change data without having to use Java class files.

Object Structures

The object structure provides the XML message content for enterprise service data processing.

Operation Type

You can select an operation type from a list of system options. The available operations depend on whether you selected the Query Only check box in the object structure.

When you select the Query Only check box the QUERY operation is available for use. When you clear the Query Only check box, the CREATE, UPDATE, DELETE, SYNC, and QUERY operations are available for use.

Enterprise Service Processing

You can provide a processing class, a user exit class, or an XSL map file, to map the XML to the object structure XML. You can apply processing to both the enterprise service configuration and enterprise service response configuration for query and create operations. If the inbound data is not in the XML format of the object structure after the enterprise service runs, a processing error occurs.
Processing Rules

The integration framework applies any specified processing rules to objects, before it saves the objects. Use object structure processing rules to set or evaluate XML fields. You can manipulate key data before the object is created.

You can also use processing rules to access and retrieve pertinent data from objects that you do not include in the object structure.

Integration Controls

You can create an integration control when the predefined integration controls do not meet your business needs. You can define integration controls globally and associate them to an enterprise service.

You can use integration controls to control the behavior of processing logic in a Java class or in a processing rule. Control values are used in enterprise service implementations to meet your business requirements.

Interface Tables

If necessary, you can change the interface table name for the service or channel.

Object Structure Services

Object structure services support the following operations: CREATE, UPDATE, DELETE, SYNC, and QUERY.

Object Structures

The object structure is the basis for the object structure service processing.

Object Structure Processing

You can provide customized inbound and outbound processing logic in the inbound processing and an outbound definition classes. Object structure services require XML messages that comply with the object structure XML schema definition.

Standard Services

You can use standard services to expose system application service methods for invocation by external applications. You can invoke these services by using HTTP, EJB, and Web services.

Web Services

You can use the integration framework to expose all integration services as Web services.

Object Structure Web Service

You can create a Web service from an object structure service. The object structure service contains the logic for integration message processing.

Enterprise Web Service

You can create a Web service from an enterprise service. You first must create an enterprise service and associate that service to an existing external system. The enterprise service contains the logic and rules for inbound integration message processing.

Standard Web Service

You can create a Web service from a standard service. The standard service contains the application-defined methods in the object for each application service.

Only the methods that are annotated in the application services are the methods that are available for selection when you create a Web service.
## Channel Creations

After determining which adapter to use, you can create new channels for the predefined and user-defined adapters. The channels that you create determine how the adapter processes outbound integration messages.

### Publish Channels

You can create publish channels in the Publish Channels application. When you trigger publish channel processing, the integration framework message data moves into the outbound queue. The message data is then transmitted to the external system through the endpoint transport mechanism.

#### Object Structures

The object structure provides the XML message content for publish channel data processing.

#### Publish Channel Processing

You can provide a processing class, a user exit class, or an XSL map file, if necessary, to map the XML to the external system XML. You also can implement processing rules or Java classes for customized publish channel processing.

#### Processing Rules

The integration framework applies any specified processing rules to objects, before it saves the objects. You can use processing rules to access and retrieve pertinent data from objects that you do not include in the object structure.

### Integration Controls

You can create an integration control when the predefined integration controls do not meet your business needs. You can define integration controls globally and associate them to a publish channel.

You can use integration controls to control the behavior of processing logic in a Java class or in a processing rule. Control values are used in publish channel implementations to meet your business requirements.

### Interface Tables

If necessary, you can change the interface table name for the service or channel.

### Invocation Channels

You can create an invocation channel in the Invocation Channels application. You can use an object structure to define the data content of the invocation channel message and its associated response. You can map the response message of the external service to the response object structure of the invocation channel.

When you map a service to the response object structure, the message responses can be passed to the caller of the external service. The message data is then converted into the system input type by using the invocation channel response processing class.

#### Object Structures

The request and response object structures are the basis for invocation channel processing.

#### Invocation Channel Processing

The invocation channel can perform a synchronous invocation of an external service and can process the response of that service into the system. You can define a processing class, a user exit class, or an XSL file, to map the object structure XML to the external service XML. You also can apply processing to both the service request configuration and service response configuration. If the response data is not in the XML format of the object structure after the response process runs, a processing error occurs.
All request and response processing classes and user exits have access to the invocation channel request and response objects by using the `IntegrationContext.getCurrentContext()` API.

**Endpoint**

The endpoint definition identifies how an invocation channel routes outbound data to a specific location.

Invocation channel endpoints can be modified by using the `IntegrationContext.getCurrentContext().getProperty(MetaDataProperties.ENDPOINTPROPS)` API. You can use this class when no endpoint is specified on the invocation channel.

The endpoint name can be passed when the invocation channel invokes the API. The endpoint name can also be set in the class by using the integration context. You must set the context API to `IntegrationContext.getCurrentContext().setProperty(MetaDataProperties.ENDPOINT,<endpointname>).`.

**Predefined Service and Channel Modifications**

Rather than create a service or channel, you can duplicate predefined services and channels, and then modify the duplicate record.

You can work with the service configurations to redefine the following characteristics:

- Classes
- Event filtering
- XSL files
- Processing rules

You can work with the channel configurations to redefine the following characteristics:

- Classes
- Extensible Stylesheet Language (XSL) files
- Processing rules

Your modified definitions determine the data source and how the service and channel handle customized data processing, filtering, and mapping.

**Creating an External System**

You can create an external system and associate the applicable services and channels with it.
Queues

A new external system can use the predefined Java Message Service (JMS) queues or queues that you create. Assign at least one inbound or outbound queue to the external system.

Working with Enterprise Services

To work with the enterprise service that the external system uses:

1. Indicate whether the service processes through the sequential or continuous JMS queue.

   You import data by using the Data Import button on the Enterprise Services tab.

2. Enable the enterprise service for inbound processing.

3. Identify the action attribute that the external application uses to submit an inbound message. The attribute defines the processing that the object must perform.

Working with Publish Channels

To work with the publish channel the external system uses:

1. Decide whether the channels run through the Data Export feature or an event-based message (data is updated by a system application).

   You export data by using the Data Export button on the Publish Channels tab.

2. Enable the publish channel for outbound processing.

3. Enable the listener that is associated with a publish channel. To enable the listener, select the Enable Event Listener Select Action menu item in the Publish Channels application.

Predefined External System Modifications

You can modify any property of a predefined external system except the external system name. You can create an external system by copying the predefined system, then updating the copy. Copy an external system copies its services and channels and their settings.
The integration framework supports external system queries. 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 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. 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.
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 use the Object Structures application to create additional object structures which provide support for the Query operation.

Creating an Enterprise Service Query

To create an enterprise service query:

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 previously created object structure.
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.

The system provides a sample query enterprise service, MXINVBALQInterface. Use this sample as a reference when you build query enterprise services.

Query Root Element

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.
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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Type</th>
<th>Applicable to</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueResult</td>
<td>Specifies whether the query expects one record or multiple records in a response.</td>
<td>Boolean</td>
<td>Query</td>
</tr>
<tr>
<td></td>
<td>Value 0 (default): The query can return multiple records.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value 1: The query can return a single record; otherwise, an error occurs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxItems</td>
<td>If the query can return multiple records, this attribute limits the number of records to be returned at one time.</td>
<td>PositiveInteger</td>
<td>Query</td>
</tr>
<tr>
<td></td>
<td>If this attribute is not specified, the response contains the entire result set.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rsStart</td>
<td>In the query request:</td>
<td>Integer</td>
<td>Query</td>
</tr>
<tr>
<td></td>
<td>Use with maxItems to specify the first record to be returned in a response.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rsStart</td>
<td>In the query response:</td>
<td>Integer</td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td>This value matches the rsStart value in the corresponding request.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the corresponding request contains a maxItems value, the rsStart value in requests for additional records is rsStart + rsCount + 1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

```
creationDateTime="2008-09-28T21:49:45" baseLanguage="EN"
transLanguage="EN" messageID="12345" maximoVersion="7.1."
uniqueResult="0" maxItems="10" rsStart="11">
  <max:MXINVBALQuery orderby="string" operandMode="OR">
    .
    .
  </max:MXINVBALQuery>
</max:QueryMXINVBALResponse>
```

The following query result set contains a total of 35 rows, as noted by rsTotal, but only rows 11 through 20 are returned.

```
creationDateTime="2008-09-28T21:49:45" baseLanguage="EN"
transLanguage="EN" messageID="12345" maximoVersion="7.1"
uniqueResult="0" maxItems="10" rsStart="11">
  <max:MXINVBALSet>
    .
    .
  </max:MXINVBALSet>
</max:QueryMXINVBALResponse>
```
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 following graphic represents the content of the QueryMXPERSON element:

- 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 additional 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 following graphic represents the content of the QueryMXPERSONResponse element:

- 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 Query Response element uses the following additional 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.

**Object Structure Element**

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 or a range of values. The integration framework supports the use of query operators such as = or >.

Selection criteria applies only to attributes of objects in the top two levels of the object structure. However, the response includes data from all the objects in the object structure.

**Field Evaluation**

The operator attribute compares the value of a database field with one or more values. It has the following format:

```
operator = value
```

The value attribute can use the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
</tr>
<tr>
<td>EW</td>
<td>Ends With</td>
</tr>
<tr>
<td>SW</td>
<td>Starts With</td>
</tr>
</tbody>
</table>

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:

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <STATUS operator ="=">ACTIVE</STATUS>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```

**Field Selection**

A field-based query compares the value in a database field with the value in the XML field of the query request.

For example, the following query searches for employees:

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```

The following query searches for employees where PERSONID is equal to ATI and STATUS is equal to ACTIVE.

```xml
<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.

The following query searches for employees where PERSONID is like %ATI%. The operand represents the default behavior and requires no operator value.

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <PERSONID>ATI</PERSONID>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```
The following query searches for the inventory balances where the bin number is not null.

```xml
<QueryMXINVBAL>
  <MXINVBALQuery>
    <INVBALANCES>
      <BINNUM operator ="!="/></BINNUM>
    </INVBALANCES >
  </MXINVBALQuery>
</QueryMXINVBAL>
```

The following query searches for the inventory balances where the bin number is null.

```xml
<MXINVBAL>
  <INVBALANCES>
    <BINNUM>NULL</BINNUM>
  </INVBALANCES >
</MXINVBAL>
```

The following query uses the equivalent of a SQL IN clause to search for the employees whose status is ACTIVE or INACTIVE.

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <STATUS>ACTIVE,INACTIVE</STATUS>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```

The following query searches for the employees where the Person ID starts with the letter A.

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <PERSONID operator ="SW">A</PERSONID>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```

The following query searches for the employees where the Person ID ends with the letter Z.

```xml
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <PERSONID operator ="EW">Z</PERSONID>
    </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.
Web Service Queries

For example, the following query searches for the purchase orders where DEPARTMENT is greater than 1000.

```
<QueryMXPERSON>
  <MXPERSONQuery>
    <PERSON>
      <DEPARTMENT operator="&gt;=">1000</DEPARTMENT>
    </PERSON>
  </MXPERSONQuery>
</QueryMXPERSON>
```

The following query searches for person records where 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>
```

Web Service Queries

Query services that are created in the Enterprise Services and the Object Structures applications can be deployed as Web services. 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="2008-09-28T21:49:45" baseLanguage="EN" transLanguage="EN" messageID="12345" maximoVersion="7.1" 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
```

- `hostname:port/meaweb` is the value of the Integration Web Application URL property.
- `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.
External applications can use Web services to query or to send transactions to the system in the integration framework. You can deploy standard services, object structure services, and enterprise services as Web services.

The system Web services are document-literal style Web services that comply with the Web Services Interoperability Organization (WS-I) Basic Profile 1.1. When XML schema files are generated, they contain the content of each service and a Web Service Definition Language (WSDL) file to describe the service. The system uses Apache Axis 2.0 as the Web service engine.
External applications can use Web services to query or send messages to the integration framework. The object structure service contains the logic for integration message processing.

You can create an object structure Web service from a predefined or a user-defined object structure. The Consumed By value is INTEGRATION.

Object structure Web services have the following characteristics:

- Support five operations through its Web Service Description Language:
  - Create
  - Update
  - Delete
  - Sync
  - Query

- Provide response content only for the Create and Query operations

- Provide the primary or alternate key of the primary object that you have defined in the object structure, as a response to the Create operation

- Provide a response for the Query operation in the object structure XML schema format

Enterprise Web Services

You can create an enterprise Web service from a predefined or a user-defined enterprise service.

Enterprise Web services differ from object structure Web services in that the enterprise Web service supports additional exit processing, business rules, and transformations not available to object structure Web services. Enterprise Web services also have the following characteristics:

- A service exists for each operation that is contained in an enterprise service record (one service per operation).

- When you use the exit processing layer, you can map an external schema XML to the object structure XML for both the invocation and the response.

- Provide response content only for the Create and Query operations that are processed in the queue.

- The enterprise service definition notes whether the service supports a response by using a ProcessResponse flag.
Can use a Java Message Service (JMS) queue (asynchronous process) or bypass the JMS queue (synchronous process).

Enterprise Web services that are processed in a JMS queue do not support the Query operation and do not provide any response content to the caller.

Enterprise Web services that are configured with the Query or Create operations provide XML content in the service response.

Standard Web Services

You can create a standard Web service from methods that are annotated in application services. In order to be available as Web services, annotated methods such as ChangeStatus, 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 system links the input and output parameters of the methods to the input and output parameters for the standard Web service.

Web Service Deployment Actions

You deploy a Web service through the Deploy Web Service action. When you deploy a Web service, the following events occur:

- Schemas are generated and regenerated for existing service schemas.
- The Web Services Description Language file is generated for the service interface.
- The Web service is deployed for the selected service.
- If 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.

Web services are deployed on the Axis Server that is based on the IsDeployed flag in the service definition. When you restart the application server, all Web services are redeployed based on the registry.

The list of deployed Web services is available in the Web Services Library application.
The generated schema files are stored in a folder called schema, which is in the location that you specify in the `mxe.int.globaldir` system property. This folder contains the following subfolders:

- Common/COMMON/META
- Common/COMMON/MOS
- Common/COMMON/MBO
- SERVICE

If you set the `mxe.int.resolveschema` includes global property in the System Properties application to true, the system resolves all include files. The entire schema content resides in a single file. This property should be used with the `mxe.int.wsdlincludesschema` property.

When you set the resolve schema property value to true (1), the resolved schema is included inside the Web Services Description Language file. If both the properties value are set to true (1), the generated Web Services Description Language has no include constructs.

The XML schema folders contain the following files.

<table>
<thead>
<tr>
<th>Subfolder</th>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>META</td>
<td>MxMeta.xsd</td>
<td>Contains meta data schema provided by the system.</td>
</tr>
<tr>
<td>MOS</td>
<td>ObjectStructurename.xsd</td>
<td>Contains object structure schema files.</td>
</tr>
<tr>
<td>MBO</td>
<td>Objectname.xsd</td>
<td>Contains the object schema files.</td>
</tr>
<tr>
<td>SERVICE</td>
<td>Servicename.xsd</td>
<td>Contains service level schema files (object structure, enterprise, and standard).</td>
</tr>
</tbody>
</table>

There is an external folder in the schema folder. If you want to use an external service to deploy an enterprise service as a Web service, place the service schema files in the external folder.

You can view the generated schema files using the following URLs.

<table>
<thead>
<tr>
<th>Schema</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>META</td>
<td><code>http://hostname:port/meaweb/schema/common/meta/file name</code></td>
</tr>
<tr>
<td>MOS</td>
<td><code>http://hostname:port/meaweb/schema/common/mos/file name</code></td>
</tr>
<tr>
<td>MBO</td>
<td><code>http://hostname:port/meaweb/schema/common/mbo/file name</code></td>
</tr>
<tr>
<td>SERVICE</td>
<td><code>http://hostname:port/meaweb/shema/web service name</code></td>
</tr>
</tbody>
</table>

- `hostname:port` is the value of the Integration Web Application URL property.
- `file name` is the name of the integration object schema you are retrieving with an .xsd file extension.
Web Service Definition Language Generation

The Web Service Definition Language file is generated during the deployment of a Web service. The generated file refers to the XML Schema Interface file in schema import. The Web Service Definition Language file defines the operation which varies depending upon the type of Web service.

The Web Service Definition Language also uses the schema definitions to specify the XML structure of the input and output messages to this operation. The Web service style is document and the binding for the input and output messages is literal, as mandated by Web Services Interoperability Organization (WS-I) Basic Profile 1.1. The Web Service Definition Language also specifies the URL on which to invoke the Web service.

A client program needs the schema definitions and Web Service Definition Language to generate client stubs. The client program uses a programming language (for example, Java or C#) to invoke the Web service.

You can access and view the system generated Web Service Definition Language files on your application server. Your configuration of the `mxe.int.globaldir` global property determines the exact the location of the Web Service Definition Language files.

You can view the generated Web Service Definition Language files at the following URL:

```
http://host:port/meaweb/wSDL/service name.wsdl
```

- `host:port/meaweb` is the value of the Integration Web Application URL property.
- `service name` is the name of the service whose Web Service Definition Language you are retrieving.
## UDDI Registration

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

<table>
<thead>
<tr>
<th>System Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxe.int.uddipuburl</td>
<td>Integration UDDI Registry Publish URL</td>
</tr>
<tr>
<td>mxe.int.uddiinqurl</td>
<td>Integration UDDI Registry Inquiry URL</td>
</tr>
<tr>
<td>mxe.int.uddiname</td>
<td>Integration UDDI Registry User ID</td>
</tr>
<tr>
<td>mxe.int.uddipassword</td>
<td>Integration UDDI Registry Password</td>
</tr>
</tbody>
</table>

If you specify values for the UDDI registry publish URL and inquiry URL properties, the system registers the Web service in the UDDI registry. To bypass UDDI registration, do not specify any values for these properties.

Only the tModel for the Web Service Definition Language is registered in the UDDI registry. The businessEntity, businessService, and bindingTemplate are not registered. The UDDI registration entry contains the URL to the Web Service Definition Language document.

## Integration Web Service Invocations

You can invoke integration Web services by using Dynamic Invocation Interface (DII) that uses the JAX-RPC API. You can also generate client stub programs by using a Web services tool of your choice, such as IBM WebSphere Application Server or Microsoft® .NET Framework. The Web service tool must reference the system generated Web Service Definition Language and schema files. The WebSphere Application Server provides the tool WSDL2Java.

When a Web service is invoked, the response is synchronous. A successful response returns an HTTP response code of 200.

By default, SOAP faults that are returned do not contain the server side stack trace, as it represents a potential security risk. However, you can change this function by changing the following parameter value in the axis2.xml file from false to true. You can access the axis2.xml file in the global directory/axis2 directory.

```xml
<parameter name="sendStacktraceDetailsWithFaults" locked="false">true</parameter>
```

When invoking any of the integration Web services, the system processes the message to the business objects. The system then returns a response to the invoker.

If you have configured the enterprise service to use a JMS queue, the system does not return any content with the response. As soon as the message is dropped into the queue, the system returns a response to the invoker and asynchronously moves the message from the queue to the business objects.
For all operations, except for the Query and Create operations, the SOAP body returns empty:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

The Query operation returns the response content according to the object structure definition.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<max:PERSONMboKeySet>
<max:PERSON>........
```

The Create operation returns the primary (or alternate, if configured) key of the primary object of the object structure.
Your system database can contain some elements, such as descriptions and long descriptions, in multiple languages. Publish channels and enterprise services can include these translated columns.

A single system 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 non-base language data to and from external systems.
Default Processing for Translatable Columns

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.

Language Tables

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

Service or channel root elements use the following attributes to manage multilanguage processing:

The baseLanguage Attribute

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.

The languageEnabled Attribute

Output XML includes the attribute on every translatable column, as shown in the following example:

<DESCRIPTION langenabled="1">Item 1 description</DESCRIPTION>

The transLanguage Attribute

The 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.
An integration module provides a mechanism for a process management product, such as Change or Release, to invoke an external operational management product, such as IBM Tivoli Provisioning Manager. You can configure an integration module to automate logical management operations, such as software deployment, by using an operational management product.

Predefined integration modules may be available for your use when you own an IBM operational management product. You can access integration modules on the operational management product CD, or by using the Tivoli Provisioning Manager. You can download integration modules from the Tivoli Open Process Automation Library (OPAL). Use the process solution installer to load integration modules 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.

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.
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 the integration module, integration module artifacts with the Java class and XSL files are provided. An integration module also provides logical management operation definitions.

The integration module definition includes the following information:

- Name, version, and description
- Operational management product and version - Identifies the operational management product and version that the integration module can run.
- Handler - 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, such as Tivoli Provisioning Manager, 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.

You can load operational management products and configuration items into the system database from a discovery engine, such as Tivoli Application Dependency Discovery, or an asset management database. First load your operational management product and configuration item data into the discovery engine using the bulk loader interface. The discovery engine combines data from multiple sources into one data model. You can use the IBM® Tivoli® Integration Composer.
to move data from the discovery engine into the database. You can also load this data by using the services configured in the integration framework.

Logical Management Operations

Logical management operations identify the actions that integration modules support, and the actions that the process management products request. Examples of logical management operations are Get Status and Deploy Software.

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 or product component, such as an integration module. You also can create a logical management operation definition in the Logical Management Operations application.

Logical Management Operation Properties

The definition of a logical management operation contains the following properties:

- **Name** - The name of the action (for example, Get Deployment Status)
- **Namespace** - The unique qualifier (for example, com.ibm.tivoli.deployment)
- **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.
Integration Module Prerequisites

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.

Integration Module Prerequisites

The integration module implements the logical management operation task using the operational management product. 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, you pass the logical management operation name and name space as input. The input arguments dictate the action that the integration module takes.
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 Management Operation 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.

Integration Module Implementation Properties

The following properties are necessary to implement an integration module.

Integration Module Parameters

The integration framework provides integration module input parameters. Parameters include 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>The instantiated ServiceInvoker object which can be a Java integration module or an invocation channel.</td>
</tr>
<tr>
<td>IMNAME</td>
<td>The name of the integration module.</td>
</tr>
<tr>
<td>IMVERSION</td>
<td>The version of the integration module.</td>
</tr>
<tr>
<td>LMONAME</td>
<td>The name of the logical management operation that is invoked.</td>
</tr>
<tr>
<td>LMONAMESPACE</td>
<td>The name space of the logical management operation that is invoked.</td>
</tr>
<tr>
<td>OMPGUID</td>
<td>The globally unique identifier (GUID) of the operational management product that the integration module uses.</td>
</tr>
<tr>
<td>ENDPOINTNAME</td>
<td>The name of the endpoint that the integration module uses to communicate with the operational management product.</td>
</tr>
</tbody>
</table>
Java-based integration modules are responsible for populating 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 tasks:

- Retrieve input from the source object or source object set
- If applicable, retrieve the integration module-specific properties
- Perform all processing logic that the logical management operation requires before it calls the operational management product
- Call the operational management product
- If applicable, handle the operational management product response
- If applicable, populate the response object or response object set with the return data
- Return processing information back to the caller

### Integration Modules and Endpoints

The integration framework provides endpoints that facilitate the invocation of an external service, such as an operational management product service. 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 that the integration module uses to communicate with an operational management product. Each endpoint must have a different set of handler properties. For example, an endpoint with a Web service handler has different properties than an endpoint with a command-line handler.

Endpoints can be overwritten at run time when the process management product passes endpoint properties as input to the integration module. The `USERNAME` property is used to communicate with the endpoint.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENDPOINTPROP</td>
<td>A map of <code>String.pdsi.iface.mic.MaxEndPointPropInfo</code> properties that override the endpoint properties.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>The user name that the integration framework uses to communicate with the endpoint.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>The password for USERNAME.</td>
</tr>
</tbody>
</table>
and PASSWORD properties that are returned from the credential mapper can also override the endpoint properties. Integration module endpoint properties are overwritten by the endpoint properties that are passed by the process management product. For example, the USERNAME and PASSWORD properties.

Java Integration Modules

Java integration modules can communicate directly with an operational management product or external integration module, without the use of an endpoint. Excluding endpoint use eliminates the need for the Java integration module to convert the 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.

Endpoint Handlers

The integration framework provides some handlers to support different communication protocols, such as HTTP and Web services. If one of the predefined handlers does not 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.

Configure an endpoint on invocation channel-based integration modules. The endpoint determines how the integration module communicates with the operational management product. 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.

Java Class and Invocation Channel Usage

You can implement an integration module either as a Java class or as an invocation channel. When you use either approach, your integration module can communicate with an external product, such as an operational management product. Before you start your integration module development, review the advantages and disadvantages to Java class and invocation channel usage.

Implementing an integration module by using a Java class has the following advantages:

- Requires less registration and configuration of system artifacts.
Integration Module Implementation Properties

- 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 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 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.
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.

**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 Java interface `psdi.iface.mic.ServiceInvoker`. 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:

```java
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 configured 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 \textit{metaData} input map:

\begin{verbatim}
import psdi.iface.omp.IMConstants;
  
  String lmoName = metaData.get(IMConstants.LMONAME);
  String lmoNamespace = metaData.get(IMConstants.LMONAMESPACE);
\end{verbatim}

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:

\begin{verbatim}
String guid = sourceMbo.getString("GUID");
int packID = sourceMbo.getInt("PACKID");
boolean hasSubs = sourceMbo.getBoolean("HASSUBS");
\end{verbatim}

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.

\section*{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.

For example, if the process management product passes a work order as the primary object, the integration framework can build a transaction that consists of the work order and its related objects, such as planned material or job plans. The objects that are identified in the object structures are used in the XML message creation.

\section*{Java Methods for Inbound and Outbound Processing}

Invocation channels support optional inbound and outbound processing exits. Processing exits provide the opportunity for integration modules to implement additional logic, such as retrieving the configuration item source token. The inbound and outbound processing classes must be instances of the Java class \texttt{psdi.iface.migexits.ExternalExit}.

In the outbound direction, the integration framework calls the following method:

\begin{verbatim}
public StructureData setDataOut(StructureData irData)
\end{verbatim}

In the inbound direction, the integration framework calls the following method:

\begin{verbatim}
public StructureData setDataIn(StructureData erData)
\end{verbatim}
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:

```java
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.

For more information and detailed instructions on adding or modifying object structures, see the online help for the Object Structures application.

---

**Integration Module Invocation**

Process management products, such as Change or Release, use a system action to invoke an integration module. The action can be associated with an application escalation or a workflow process, or it might be initiated from a menu item or a button. The process management product must provide an action class that is designed to invoke the integration module and process the response the integration module returns.

To complete an integration module call, the custom action Java class that the process management product provides must perform the following steps:

1. Identify the logical management operation.
Integration Module Invocation

2 Identify the operational management product and integration module.

3 Invoke the integration module.

4 Process the integration module response.

Step 1: Identify the Logical Management Operation

The process management product provides processes to run logical management operations. 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.

Step 2: Identify the Operational Management Product and Integration Module

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.

Service Utility Methods

The integration framework includes a service that provides the following utility methods to assist the class with integration module lookups.

<table>
<thead>
<tr>
<th>Utility Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>psdi.iface.app.omp.OmpSetRemote getOMPListForIM(String imName, String imVersion)</td>
<td>Retrieves a list of the operational management products that are associated with the integration module.</td>
</tr>
<tr>
<td>psdi.iface.app.im.MaxIMSetRemote getIMListForLMO(String lmoName, String lmoNamespace)</td>
<td>Retrieves a list of the integration modules that implement the specified logical management operation.</td>
</tr>
<tr>
<td>psdi.iface.app.im.MaxIMSetRemote getIMListForLMOWithOMP(String lmoName, String lmoNamespace)</td>
<td>Retrieves a list of the integration modules that implement the specified logical management operation on any operational management products.</td>
</tr>
</tbody>
</table>
Step 3: Invoke the Integration Module

Integration module instances are called service invokers because they implement the Java interface `psdi.iface.mic.ServiceInvoker`. Integration modules can be implemented either as Java classes or as invocation channels. 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.

**getServicInvoker 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.
Integration Module Invocation

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 Method

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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaData</td>
<td>The property map that the get service invoker utility method returns.</td>
</tr>
<tr>
<td>Source object and object set</td>
<td>The object with the input attributes that are defined on the logical management operation. This property can contain a null value.</td>
</tr>
<tr>
<td>Target object and object set</td>
<td>The object and object set that contains the return data. This property can contain a null value.</td>
</tr>
<tr>
<td>Action</td>
<td>This parameter indicates whether existing objects in the targetMboSet are updated, or new objects are added.</td>
</tr>
<tr>
<td></td>
<td>The possible values for action are:</td>
</tr>
<tr>
<td></td>
<td>▼ psdi.mbo.MboSetRemote.INSERTONLY</td>
</tr>
<tr>
<td></td>
<td>▼ psdi.mbo.MboSetRemote.UPDATEONLY</td>
</tr>
<tr>
<td>Endpointname</td>
<td>The name of the endpoint that the integration module uses for communication.</td>
</tr>
</tbody>
</table>

**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.

```java
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.
Get Service Invoker Utility Methods

The operational management product service provides the following getServiceInvoker utility methods to retrieve an instance of an integration module for a logical management operation and an operational management product.

<table>
<thead>
<tr>
<th>Utility Method</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;String, Object&gt; getServiceInvoker (psdi.iface.omp.OmpImLmoRelInfo ompImLmoRelInfo, psdi.security.UserInfo userInfo)</td>
<td>Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.</td>
</tr>
<tr>
<td>Map&lt;String, Object&gt; getServiceInvoker (psdi.iface.app.im.OmpImLmoRelRemote ompImLmoRelRemote, psdi.security.UserInfo userInfo)</td>
<td>Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.</td>
</tr>
<tr>
<td>Map&lt;String, Object&gt; getServiceInvoker (String ompGUID, String imName, String imVersion, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo)</td>
<td>Retrieves the service invoker for the specified integration module, logical management operation, and operational management product.</td>
</tr>
<tr>
<td>Map&lt;String, Object&gt; getServiceInvoker (String ompGUID, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo)</td>
<td>Retrieves the service invoker for the preferred integration module for the specified operational management product and logical management operation.</td>
</tr>
<tr>
<td>Collection&lt;Map&gt; getServiceInvokerListForCIAndLMO(String actCIGUID, String lmoName, String lmoNamespace, psdi.security.UserInfo userInfo)</td>
<td>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.</td>
</tr>
</tbody>
</table>

Step 4: Process the Integration Module Response

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.
Launch-in-Context Feature

The launch-in-context feature opens a console for an external application that takes you directly to information that is relevant to your work within the system. 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, such as the IBM Tivoli Provisioning Manager or the IBM Tivoli Configuration Manager. Additionally, you can configure console URLs for consoles that use Java™ Web Start, such as Tivoli Application Dependency Discovery Manager. You cannot use a launch entry to open applications that are not enabled for the Web.

You can configure a launch point from any system application. Launch points include Select Action menu items, hyperlinks, and buttons. Some special launch-in-context capabilities are specific to the operational management product navigations, such as navigations to the Tivoli Provisioning Manager or the Tivoli Configuration Manager.
External Application Preparations

Most external applications have a Web-based console that you can launch from a URL in a Web browser. The Launch-in-context feature supports the Web application (Servlet or JSP), portal, and Java WebStart console types.

To perform a navigation, the external application console must support the land-in-context capability. To support context sharing, the land-in-context capability accepts data that is passed by the system to the URL. The external console uses the URL to open a window with the data that was passed to the URL.

For example, in the Tivoli Application Dependency Discovery Manager Web console, you can pass the name of a view and the globally unique identifier (GUID) of a configuration item to show in the view. If the external application does not have a land-in-context capability, then the launch-in-context capability can open a standard start page within the console.

If your application supports single sign-on, application users are authenticated and directed to the external console. Otherwise, the external console opens a login panel to authenticate the application user.

Defining Launch Entries

To make a launch entry available to the system applications:

1 Define the properties of the launch entry.
2 Create a user interface control, also known as a launch point, that is tied to the registered launch entry. Available controls include a push button, a select action menu item, and a hyperlink.
3 Assign proper security to the control. Limit access to the control by user and user group.
4 Optional: Assign a condition to the interface control to limit when the launch point is displayed, based on the data you are working with.

Launch Entry Properties

When you define launch entries, you enter these property values:

- The launch entry name.
- The launch entry description.
- The URL of the console of the external application.

The URL value can contain substitution variables that use data from the related business objects that you are viewing in the application. For example, if you have a launch entry that is implemented in your Person application, you can substitute the name value into the URL string by using the attribute name, {attributename}. 
For example, the following URL is a URL for the Person application that uses the PERSONNAME attribute.

https://extsyshost:9045/tcWebUI/interaction-handler?actionId=viewPerson&Person={PERSONNAME}

You can also use 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 Person application when the ADDRESS object is related to a PERSON object with a relationship named ADDRESS.

https://extsyshost:9045/WebUI/interaction-handler?actionId=viewCity&cityname={ADDRESS.CITY}

Use the `{sourcetoken}` and the `{reportinghostname}` values in your URL when you want to launch to an operational management product console that manages a configuration item.

- **The target browser window value.**

  The target browser window value identifies whether the window that you want to open the external application in:

<table>
<thead>
<tr>
<th>Browser Window</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The current browser window.</td>
<td>_usecurrent</td>
</tr>
<tr>
<td>A new browser window.</td>
<td>_blank</td>
</tr>
<tr>
<td>A named browser window.</td>
<td>The name of the window.</td>
</tr>
</tbody>
</table>

  A new browser window is opened on the first launch and then reuses the browser window on subsequent launches where the name is the same.

- **Launch entry business objects.**

  The business objects that you assign restrict the launch entry usage to applications that support these objects. The restriction applies when you associate a launch entry to a user interface control.

  A launch entry can support single or multiple business objects. No restrictions apply to the launch entry use when 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.
Operational Management Product-specific Properties

When you define launch entries, you enter these property values for the associated operational management product:

- **Specific to operational management product launches:**
  - Name and version of the operational management product.

  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.

- **Operational management product source token.**

  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.

- **Operational management product host name.**

  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 Tivoli Application Dependency Discovery Manager. If the host name information is not loaded from the discovery engine, you must add the operational management product server information to your database.
Associating the Launch Entry with a Signature Option

Before you add a launch point to an application, you must associate the launch entry record with a signature option.

To associate the launch entry with a signature option:

1. Open the Application Designer application.
2. Select the application record to which you want to associate the launch entry.
3. From the Select Action menu, select Add/Modify Signature Options.
4. In the Add/Modify Signature Options dialog box, select New Row.
5. Enter values in the following fields:
   - Option - A short name for the signature option.
   - Description - The description that is on the menu item or button.
6. In the Advanced Signature Options table window, select the Associate to Launch Entry To Enable The Launch in Context radio button.
7. In the Launch Entry Name field, specify a name for the launch entry.
8. Click OK to return to the Application Designer application.

Associating a Signature Option with a User Interface Control or a Toolbar Menu Item

When you associate the signature option with a user interface control, you are linking the launch entry with a launch point or a tool bar menu item. The control types determine the function of the launch point.

To associate a signature option with a user interface control:

1. Open the Application Designer application.
2. Select the application to which you are adding the menu item.
3. From the Select Action menu, select Add/Modify Select Action Menu or select Add/Modify Toolbar Menu.
4. In the Add/Modify Select Action Menu or Add/Modify Toolbar Menu dialog box, select New Row.
5. In the Element Type field, enter the value type OPTION.
6. In the Key Value field, enter your signature option name.
7 In the **Position** field, enter a number to specify the relative position of the select menu item.

8 In the **Tabs** field, select the **MAIN** value when the launch URL has substitution variables in it, otherwise select the **ALL** value.

9 Optional: Enter values in the following fields:

   ▼ **Header Description**
   ▼ **Tabs**

10 Click OK to save your changes and to return to the Application Designer application.

11 Click **Save Application Definition**.

12 Use the Applications tab in the Security Groups application to grant user and group access privileges to the new launch entry menu item.

   For more detailed instructions about granting access privileges to the menu item, refer to the online help in the Security Groups application.

---

**Associating a Signature Option with a Push Button**

When you associate the signature option with a push button you are linking the launch entry with a push button.

To associate a signature option with a push button:

1 In the Application Designer application, on the toolbar, click **Control Palette**.

2 Drag the Pushbutton control onto the application work area.

3 Click the push button control to open the Control Properties dialog window.

4 In the **Label** field, type the name that you want to appear on the button.

5 In the **Event** field, type the name of the signature option.

6 Click **OK** to save your changes and to return to the Application Designer application.

7 Click **Save Application Definition**.

---

**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 pre-existing 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; values are case-sensitive.

### Associating a Condition with a Signature Option

After you create a condition, you must associate the condition with the signature option. 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.

To associate a condition to a signature option:

1. Open the Security Groups application.
2. Select the group for which you want to apply the condition.
3. In the Application tab, select the application for which you want to apply the condition.
4. In the Options table window, select the condition value that you have created.
5. Click **Save Group**.
Collaboration Switches

Collaboration switches have been designed from an inbound integration point of view to provide users with a way to better manage data synchronization between the integration framework and external systems using a concept of ownership. These switches provide the ability to control sub-processes within specific system application functionality based on ownership of different data objects within a transaction.

Most master data and document integration objects in the system have an OWNERSYSID attribute present in the primary object of the object structure. By default, the system inbound integration processing does not specify any value in this field, and processing of enterprise services proceed as per standard system functionality.

Format of Collaboration Switches

Collaboration switches provide a flexible, user-defined way to control the processing of some inbound transactions, by letting the system bypass the default processing for certain types of transactions.

The collaboration switches reside in the system MXCOLLAB table. Each collaboration switch contains four elements, three of which combine to create a unique key. The following table lists these elements. An asterisk (*) indicates the elements that comprise of the key.

<table>
<thead>
<tr>
<th>Element</th>
<th>Corresponding MXCOLLAB field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process control ID*</td>
<td>PCID</td>
</tr>
<tr>
<td>System ID 1*</td>
<td>OWNER1SYSID</td>
</tr>
<tr>
<td>System ID 2*</td>
<td>OWNER2SYSID</td>
</tr>
<tr>
<td>Process control value</td>
<td>PCVALUE</td>
</tr>
</tbody>
</table>

Process Control ID

The process control ID identifies a business process in a system 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.

<table>
<thead>
<tr>
<th>Prefix of Process Control ID</th>
<th>Corresponding Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>Invoice</td>
</tr>
<tr>
<td>ITM</td>
<td>Item</td>
</tr>
<tr>
<td>IV</td>
<td>Inventory</td>
</tr>
<tr>
<td>LT</td>
<td>Labor</td>
</tr>
<tr>
<td>PO</td>
<td>Purchase order</td>
</tr>
<tr>
<td>PR</td>
<td>Purchase requisition</td>
</tr>
<tr>
<td>WO</td>
<td>Work order</td>
</tr>
</tbody>
</table>

For example, the IVRC, IVRCY, and IVWO collaboration switches are all related to inventory processing.

**System ID 1 and System ID 2**

System ID 1 and System ID 2 identify your system and an external system.

The values in these fields vary, depending on the transaction and the objects in the transaction. In general, System ID 1 identifies the system (system or external) that created the object, and System ID 2 identifies the system that created the record that is being referenced or updated.

**Process Control Value**

The process control value specifies whether the system business components should 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:

<table>
<thead>
<tr>
<th>Value of Process Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Performs default processing</td>
</tr>
<tr>
<td>1</td>
<td>Bypasses default system processing</td>
</tr>
</tbody>
</table>

**Default Collaboration Switches**

The system creates three default collaboration switches, with different combinations of system ID values for each process control ID. Authorized users can create additional switches as needed.
The default switches use various combinations of the following values in the System ID 1 and System ID 2 fields:

<table>
<thead>
<tr>
<th>System ID Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>THISMX</td>
<td>The system identified in MAXVARS.MXSYSID</td>
</tr>
<tr>
<td></td>
<td>The collaboration switches do not use the actual value in MXSYSID.</td>
</tr>
<tr>
<td>EXT</td>
<td>Any system other than the one identified in MAXVARS.MXSYSID</td>
</tr>
</tbody>
</table>

Authorized users can update only the process control value in the default collaboration switches.

Deleting a default collaboration switch or modifying a field other than the process control value can result in a system failure.

**Collaboration Switch Retrieval**

Each process control ID has at least three related collaboration switches (the defaults and any that the user adds). The system uses the following logic to determine which system ID values to set when retrieving a collaboration switch from the MXCOLLAB table:

1. The system uses the rules for deriving the System ID 1 and System ID 2 values for the process control ID in question.
   
   For example, the process control ID PRDEL, System ID 1 is the literal THISMX and System ID 2 is the system that owns the PR.

2. If the value in System ID 1 is now blank, null, or equal to the value in the MXSYSID row of the system MAXVARS table, the system uses THISMX for System ID 1.

3. If the value in System ID 2 is now blank, null, or is equal to the value in the MXSYSID row of the system MAXVARS table, and the process control ID is not PRPAB, the system uses THISMX for System ID 2.

4. If the process control ID is PRPAB, the value in System ID 2 will be null after step 1, and the system uses EXT for System ID 2. (if the blanket PO does not exist).

5. If both System ID 1 and System ID 2 now equal THISMX, the system performs the default system processing (that is, it acts as if it retrieved an MXCOLLAB record with a process control value of 0).
   
   If both System ID 1 and System ID 2 equal THISMX, ignore the remaining steps.

6. The system tries to find a record with the modified key in the MXCOLLAB table.
If the record exists, the system uses the record process control value to determine whether or not to bypass system processing.

If the record exists, ignore the remaining steps.

7 If the record does not exist, the system modifies the key as follows:

If System ID 1 now equals THISMX and System ID 2 does not equal THISMX, the system uses EXT as the System ID 2 value.

If System ID 1 value does not equal THISMX and System ID 2 equals THISMX, the system uses EXT as the System ID 1 value.

8 The system tries to find a record with the modified key in the MXCOLLAB table.

If the record exists, the system uses the record process control value to determine whether or not to bypass system processing.

If the record exists, ignore the remaining step.

9 The system uses EXT for both System ID 1 and System ID 2.

10 The system finds the record with the modified key in the MXCOLLAB table.

This record always exists, because every process control value has a default collaboration switch with both system IDs equal to EXT.

11 The system uses the record process control value to determine whether or not to bypass system processing.

View Collaboration Switches

You can use any database tool to display the values in the MXCOLLAB table. You can also create a report with the report-writing tools available in the system.

To display the collaboration switches for a single process control ID, use the following SQL query:

```sql
select pcid, owner1sysid, owner2sysid, pcvalue
from mxcollab
where pcid = 'PCID'
order by pcid, owner1sysid, owner2sysid;
```

To display all collaboration switches, use the following SQL query:

```sql
select pcid, owner1sysid, owner2sysid, pcvalue
from mxcollab
order by pcid, owner1sysid, owner2sysid;
```

To display a short description of the process control IDs, use the following SQL query:

```sql
select * from mxcollabref order by pcid;
```
Collaboration Switch Modification

Authorized users can use any database tool to modify the process control value of a collaboration switch.

Do not change the value of PCID, OWNER1SYSID, or OWNER2SYSID on existing collaboration switches.

To change the process control value in a collaboration switch, use the following SQL statement:

```sql
update mxcollab
set pcvalue = PCVALUE
where pcid = 'PCID'
and owner1sysid = 'OWNER1SYSID'
and owner2sysid = 'OWNER2SYSID';
```

The values in the MXCOLLAB table are case-sensitive.

Collaboration Switch Additions

Authorized users can add new collaboration switches to the MXCOLLAB table. New switches must use an existing process control ID, but they can use new system IDs.

Only the default collaboration switches can use the values THISMX and EXT in the system ID fields.

To add a collaboration switch, use the following SQL statement:

```sql
insert into mxcollab
(pcid, owner1sysid, owner2sysid, pcvalue)
values ('PCID', 'OWNER1SYSID', 'OWNER2SYSID', PCVALUE);
```

For example, you want the integration framework to integrate the system with an Oracle Financials® system and other systems. When Oracle Financials issues system-owned inventory, you want the system to accept the issue and update inventory balances and costs. However, when other applications issue system-owned inventory, you want to accept the issue, but you do not want to update inventory balances or costs.
Before you modify the MXCOLLAB table to reflect these conditions, the INV collaboration switches to have the following values:

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>System ID 1</th>
<th>System ID 2</th>
<th>Process Control Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>THISMX</td>
<td>EXT</td>
<td>1</td>
</tr>
<tr>
<td>INV</td>
<td>EXT</td>
<td>THISMX</td>
<td>0</td>
</tr>
<tr>
<td>INV</td>
<td>EXT</td>
<td>EXT</td>
<td>1</td>
</tr>
</tbody>
</table>

**Modifying the MXCOLLAB Table**

To modify the MXCOLLAB table:

1. Change the value of the INV/EXT/THISMX collaboration switches to 1 to bypass normal update processing.

   To update the collaboration switch, use the following SQL statement:

   ```sql
   update mxcollab
   set pcvalue = 1
   where pcid = 'INV'
   and owner1sysid = 'EXT'
   and owner2sysid = 'THISMX';
   ```

2. Select a system ID to represent Oracle Financials.

   This procedure uses ORC as the system ID.

3. Add a new collaboration switch INV/ORC/THISMX to the MXCOLLAB table.

   The single SQL statement following step 5 performs the processing for steps 3, 4, and 5.

4. Set the new collaboration switch process control value to 0.

   This value directs the system to perform the normal balance and cost updates when it receives issues from Oracle Financials.

5. Set the OWNERSYSID attribute on the inbound transaction to the system ID you chose in step 2—in this case, ORC.

   If OWNERSYSID is blank, the value in the DEFEXTSYS integration control is used.

To perform the processing described in steps 3, 4 and 5, use the following SQL statement:

```sql
insert into mxcollab
(pcid, owner1sysid, owner2sysid, pcvalue)
values ('INV', 'ORC', 'THISMX', 0);
```
After you perform this procedure, the INV collaboration switches will have the following values.

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>System ID 1</th>
<th>System ID 2</th>
<th>Process Control Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>THISMX</td>
<td>EXT</td>
<td>1</td>
</tr>
<tr>
<td>INV</td>
<td>EXT</td>
<td>THISMX</td>
<td>1</td>
</tr>
<tr>
<td>INV</td>
<td>EXT</td>
<td>EXT</td>
<td>1</td>
</tr>
<tr>
<td>INV</td>
<td>ORC</td>
<td>THISMX</td>
<td>0</td>
</tr>
</tbody>
</table>

For example, when you set the value of the ISUIN integration control to 1, the 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, so that the system bypasses that update process.

The INV/EXT/THISMX collaboration switch controls the processing of inventory (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 were 0, the inventory balance and cost values are updated. This is the default processing.

If the value of the INV/EXT/THISMX collaboration switch were 1, the default processing is bypassed. The 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.
## Inventory Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV</td>
<td>Update inventory.</td>
<td>0 Update inventory.</td>
<td>INVTRANS or MATUSERTRANS</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td>Used when creating issues, returns, or miscellaneous receipts or adjustments.</td>
<td>1 Do not update inventory.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inventory must exist in this system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVDEL</td>
<td>Delete inventory.</td>
<td>0 Delete item if it passes normal system validations.</td>
<td>“THISMX”</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td>Used when deleting externally owned inventory.</td>
<td>1 Delete inventory without any validations and delete INVBALANCES record for the item.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attention: If value is 1, the item-storeroom will still exist on related open PRs, POs, RFQs, work orders, and so on. This might result in problems receiving/approving these lines.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVISS</td>
<td>Enter item issues.</td>
<td>0 Allow material issues for the inventory.</td>
<td>MATUSERTRANS</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td>Used when issuing material.</td>
<td>1 Do not allow material issues for the inventory.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVISSR</td>
<td>Enter item issue returns.</td>
<td>0 Allow material returns.</td>
<td>MATUSERTRANS</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td>Used when returning material.</td>
<td>1 Do not allow issue returns for the material.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVISSWO</td>
<td>Update work order actual cost, equipment INVCOSTs.</td>
<td>0 Update work order actual material cost, equipment INVCOST.</td>
<td>MATUSERTRANS</td>
<td>WORKORDER</td>
</tr>
<tr>
<td></td>
<td>Used when processing issues or returns. Meant to handle system to system cases where these updates will be done separately.</td>
<td>1 Do not update work order actual material cost, equipment INVCOST.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVPHY</td>
<td>Enter external physical counts.</td>
<td>0 Allow physical count for the inventory.</td>
<td>INVTRANS</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td>Used when creating physical counts.</td>
<td>1 Do not allow physical count for the inventory.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Control ID</td>
<td>Description</td>
<td>Value and Action</td>
<td>Derivation of System ID 1</td>
<td>Derivation of System ID 2</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>------------------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>INVTR</td>
<td>Update the From storeroom on a transfer or the receipt of internal PO. Used when creating transfers or creating a receipt for an internal POs.</td>
<td>0 Update INVBALANCES in the From storeroom. 1 Do not update INVBALANCES in the From storeroom.</td>
<td>MATRECTRANS</td>
<td>LOCATIONS (storeroom for transfer; vendor for internal PO)</td>
</tr>
<tr>
<td>ITMDEL</td>
<td>Delete items. Used when deleting items that this system does not own. Attention: If value is 1, the item will still exist on related open PRs, POs, RFQs, work orders, and so on. This might result in problems receiving/approving these lines.</td>
<td>0 Delete item if it passes normal system validations. 1 Delete item without any validation. Also delete INVENTORY, INVBALANCES, and INVVENDOR records for the item.</td>
<td>“THISMX”</td>
<td>ITEM</td>
</tr>
</tbody>
</table>
## Invoice Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVILC</td>
<td>Update inventory last cost. Used when approving invoices.</td>
<td>0 Update inventory last cost.</td>
<td>INVOICE</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not update inventory last cost.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVINV</td>
<td>Update inventory average cost. Used when approving invoices.</td>
<td>0 Update inventory average cost.</td>
<td>INVOICE</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not update inventory average cost.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVMATCH</td>
<td>Use and validate invoice match. Used when approving invoices.</td>
<td>0 Validate match.</td>
<td>“THISMX”</td>
<td>INVOICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not validate any match provided.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVPO</td>
<td>Update POs and receipts. Used when approving invoices.</td>
<td>0 Update PO status and receipts.</td>
<td>INVOICE</td>
<td>PO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not update PO status or receipts.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVPRO</td>
<td>Check and prorate differences between invoice headers and lines. Used when approving invoices.</td>
<td>0 Prorate the difference between the header and the line total.</td>
<td>“THISMX”</td>
<td>INVOICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not prorate the difference between the header and line total.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVRC</td>
<td>Create service receipts for invoice lines without a PO reference. Used when approving invoices that contain a line without a PO reference.</td>
<td>0 Generate a service receipts for the invoice lines that do not have a PO reference.</td>
<td>“THISMX”</td>
<td>INVOICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not generate a service receipt for the invoice lines that do not have a PO reference.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Control ID</td>
<td>Description</td>
<td>Value and Action</td>
<td>Derivation of System ID 1</td>
<td>Derivation of System ID 2</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>IVRCY</td>
<td>Create service receipts for invoice lines with a PO reference and RECEIPTREQD = N. Used when approving invoices containing a line with a PO reference, when the corresponding POLINE is a service and RECEIPTREQD = N.</td>
<td>0  Generate a service receipt for the invoice line.  1  Do not generate a service receipt for the invoice line.</td>
<td>INVOICE</td>
<td>PO</td>
</tr>
<tr>
<td>IVTOL</td>
<td>Perform invoice tolerance checking validation.</td>
<td>0  Perform all tolerance checks on invoice.</td>
<td>“THISMX”</td>
<td>INVOICE</td>
</tr>
<tr>
<td></td>
<td>Used when approving invoices.</td>
<td>1  Do not perform tolerance checks on invoice.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVVLC</td>
<td>Update vendor last cost.</td>
<td>0  Update vendor last cost.</td>
<td>INVOICE</td>
<td>INVVENDOR</td>
</tr>
<tr>
<td></td>
<td>Used when approving invoices.</td>
<td>1  Do not update vendor last cost.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IVWO</td>
<td>Update work orders.</td>
<td>0  Update work order.</td>
<td>INVOICE</td>
<td>WORKORDER</td>
</tr>
<tr>
<td></td>
<td>Used when approving invoice.</td>
<td>1  Do not update work order.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Labor Transaction Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTSRC</td>
<td>Generate service receipts for POs. Used when creating labor transactions or changing status. PO must exist in this system.</td>
<td>0 Allow setting value of LABTRANS.GENAPPRSRVRECEIPT to Y; configurable in the Application Setup. Leave value of LABTRANS.GENAPPRSRVRECEIPT as N.</td>
<td>LABTRANS</td>
<td>PO</td>
</tr>
</tbody>
</table>

## Purchase Order Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PODEL</td>
<td>Delete POs. Used when deleting POs. Use only when deleting then subsequently re-adding a PO due to changes in the PO. If any PRLINES contain a reference to the PO, clear them. If necessary, reopen the PR. When the PO is re-added, the PRLINEs will be established again.</td>
<td>0 Do not delete PO. 1 Delete PO and PRLINEs; do not delete POSTATUS.</td>
<td>“THISMX”</td>
<td>PO</td>
</tr>
<tr>
<td>POINV</td>
<td>Do not allow unreferenced external inventory for internal POs. Used when adding or updating PO lines and changing the status of internal POs.</td>
<td>0 If the item-vendor combination not in INVENTORY table, error. 1 If item-vendor combination not found in INVENTORY table (where PO.VENDOR = LOCATIONS.LOCATION), ignore error.</td>
<td>PO</td>
<td>LOCATIONS, where vendor is the storeroom</td>
</tr>
<tr>
<td>Process Control ID</td>
<td>Description</td>
<td>Value and Action</td>
<td>Derivation of System ID 1</td>
<td>Derivation of System ID 2</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>POIVM</td>
<td>Create inventory vendor information for inventory. Used when approving POs.</td>
<td>0 Update or create INVVENDOR record. 1 Do not create INVVENDOR record.</td>
<td>PO</td>
<td>ITEM of POLINE</td>
</tr>
<tr>
<td>POPR</td>
<td>Update status of PRs. Used when copying PR lines to P0s, creating P0s from PRs, reopening PRs.</td>
<td>0 Change status of PR (auto close based on MAXVAR setting) or reopen when POLINE containing PR reference is deleted, or other instances of reopen. 1 Do not change PR status.</td>
<td>PO</td>
<td>PR</td>
</tr>
<tr>
<td>POREL</td>
<td>Create releases for blanket P0s. Used when approving PR and the PR lines contain a blanket references, and when a release is created directly from a P0 without a PR.</td>
<td>0 Generate PO release. (If PRLINE.AGREEMENTPONUM not in PO, do not generate PO release). 1 Do not regenerate PO releases.</td>
<td>PR</td>
<td>PO of the blanket</td>
</tr>
<tr>
<td>PORES</td>
<td>Process material reservations. Used when changing the status of internal P0s.</td>
<td>0 Generate inventory reservations. If item-vendor combination (where vendor is internal storeroom) not in INVENTORY table, do not generate PO reservations. This might happen if POINV is 0. 1 Do not generate inventory reservations.</td>
<td>PO</td>
<td>INVENTORY</td>
</tr>
</tbody>
</table>
### Purchase Requisition Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRDEL</td>
<td>Delete PRs. Used when deleting PRs. Use only when deleting then subsequently re-adding a PR due to changes in the PR. If WPMATERIAL or MRLINE contain references to the PR, clear them. They will be reestablished when you read PR.</td>
<td>0 Do not delete PR. 1 Delete the PR and PRLINES; do not delete PRSTATUS.</td>
<td>“THISMX”</td>
<td>PR</td>
</tr>
<tr>
<td>PRINV</td>
<td>Do not allow unreferenced external inventory on internal PRs. Used when storerooms are maintained in an external system. Items are in ITEM master in the system; storeroom is defined as a LOCATION; INVENTORY is not defined for item-storeroom combination because it is not owned by the system. The owner of the PR is the MXSYSID of the system that creates the PR. Validation occurs when an item-storeroom (INVENTORY) is validated on the PRLINE. The OWNERSYSID of the storeroom is compared with the OWNERSYSID of the PR, and the flag determines if the combination is allowed.</td>
<td>0 If the item-vendor combination is not in INVENTORY table, error. 1 If the item-vendor combination (vendor is the internal storeroom) is not in INVENTORY table, where PR.VENDOR = LOCATIONS.LOCATION, ignore error. LOCATIONS must exist; that is, pass standard validation for the location.</td>
<td>PR</td>
<td>LOCATIONS, where vendor is the internal storeroom</td>
</tr>
<tr>
<td>Process Control ID</td>
<td>Description</td>
<td>Value and Action</td>
<td>Derivation of System ID 1</td>
<td>Derivation of System ID 2</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>PRPAB</td>
<td>Do not allow unreferenced external purchase agreements / blankets. Used when adding or updating PR lines and changing the status of PRs.</td>
<td>0 If PRLINE.AGREEMENTPONUM is not in PO, error. 1 If PRLINE.AGREEMENTPONUM is not in PO, ignore error.</td>
<td>PR</td>
<td>“EXT”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Normally this would be from PO of the blanket, but in this case the blanket PO does not exist</td>
</tr>
</tbody>
</table>
# Receipt Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>Enter PO receipts. Used when creating receipts.</td>
<td>0</td>
<td>Allow receiving against the PO.</td>
<td>MATRECTRANS or SERVRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not allow receiving against the PO.</td>
<td>PO</td>
</tr>
<tr>
<td>RCILC</td>
<td>Update inventory last cost. Used when approving receipts.</td>
<td>0</td>
<td>Update inventory last cost.</td>
<td>MATRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not update inventory last cost.</td>
<td>INVENTORY</td>
</tr>
<tr>
<td>RCINV</td>
<td>Update inventory. Used when receiving, or approving receipts.</td>
<td>0</td>
<td>Update inventory if it exists.</td>
<td>MATRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not update inventory.</td>
<td>INVENTORY</td>
</tr>
<tr>
<td>RCIV</td>
<td>Generate invoices for PO receipts. Used when approving receipts.</td>
<td>0</td>
<td>Generate invoice if value of PayOnReceipt is set.</td>
<td>MATRECTRANS or SERVRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not generate invoice, even if value of PayOnReceipt is set.</td>
<td>PO</td>
</tr>
<tr>
<td>RCPO</td>
<td>Update external PO. Used when approving receipt.</td>
<td>0</td>
<td>Update PO.</td>
<td>MATRECTRANS or SERVRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not update PO.</td>
<td>PO</td>
</tr>
<tr>
<td>RCR</td>
<td>Enter PO receipt return. Used when creating receipt return.</td>
<td>0</td>
<td>Allow receipt returns for the PO.</td>
<td>MATRECTRANS or SERVRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not allow receipt returns for the PO.</td>
<td>PO</td>
</tr>
<tr>
<td>RCVLC</td>
<td>Update vendor last cost. Used when approving receipt or receiving PO line.</td>
<td>0</td>
<td>Update vendor last cost.</td>
<td>MATRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not update vendor last cost.</td>
<td>INVVENDOR</td>
</tr>
<tr>
<td>RCWO</td>
<td>Update work orders. Used when approving receipts.</td>
<td>0</td>
<td>Update work order.</td>
<td>MATRECTRANS or SERVRECTRANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Do not update work order.</td>
<td>WORKORDER</td>
</tr>
</tbody>
</table>
## Work Order Collaboration Switches

<table>
<thead>
<tr>
<th>Process Control ID</th>
<th>Description</th>
<th>Value and Action</th>
<th>Derivation of System ID 1</th>
<th>Derivation of System ID 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORES</td>
<td>Process material reservations. Used when changing the status of a work order. Inventory must exist in this system.</td>
<td>0 Generate inventory reservation.</td>
<td>WORKORDER</td>
<td>INVENTORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Do not generate inventory reservation.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply 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 Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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.

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.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.
Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
AIX
DB2
developerWorks
Everyplace
ibm.com
Lotus
Maximo
Notes
QuickPlace
Tivoli
WebSphere

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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.
Index

A

action 14
ACTION column, interface tables 103
adapters
  create 233
  modifying 236
alias column names 99
APIs 184
asynchronous inbound processing
  enterprise service class processing 26
  enterprise services 20
  message retrieval 24
  object creation 29
  object processing 31
  object processing rules 29
  object structure 24
  object structure multiplication 28
  object structure processing class 30
  object structure processing rules 29
  process initiation 21
queue messages 21
  Data Import 22
  data import cron task 24
  EJB 21
  HTTP 21
  interface tables 23
  user exit object processing 30
  user exit postprocessing 27
  user exit preprocessing 25
XSL Map 27
asynchronous outbound processing 48
  process initiation 49
  publish channels 48
    object structure 50
    object structure multiplication 51
    object structure processing rules 50
    publish channel class processing 52
    user exit postprocessing 53
    user exit processing 52
  XSL Map 54

B

baseLanguage attribute 261
basic configuration
  predefined components 130
  predefined queues 130
  prerequisite activities 130
  system properties 131
boolean controls 210
business object user exit processing customization 228

C

channels
  create 235
  invocation 235
  modifying 236
  publish 235
cluster configuration
  cron task 188
  global directory 195
  inbound message receipt 195
    enterprise beans 196
    HTTP servlet 196
    web services 196
  JMS queues 189
  message processing servers 195
  WebLogic continuous queue 193
  WebLogic sequential queue 194
  WebSphere continuous queue 190
  WebSphere sequential queue 192
CMDLINE handler
  definition 155
collaboration switches
  default 290
  format 289
  invoice 298
  labor transaction 300
  overview 289
  purchase order 300
  purchase requisition 302
  receipt 304
  viewing 292
  work order 305
column name length 99
columns
continuous queues
  configuring message beans 167
    WebLogic server 168
    WebSphere application server 167
  definition 165
  enabling message beans 166
  message caching 168
  message format 171
  performance 167
  queue errors 169
    WebLogic server 170
    WebSphere application server 169
  WebLogic cluster 193
  WebSphere cluster 190
cron task
  cluster configuration 188
  FLATFILECONSUMER parameters 135
  FLATFILECONSUMER processing properties 135
  IFACETABLECONSUMER 105, 160
  selectors 160
  XMLFILECONSUMER parameters 133
  XMLFILECONSUMER processing properties 134
Index

cross-reference controls 210

customization
  business object user exit processing 228
  conditions, evaluations 205
  EJB handler 220
  handlers 218
  HTTP handler 219
  inbound processing 223
  interface tables 223
  object rule 200
  objects, records 200
  outbound processing 214
  processing rule 201
  user exit postprocessing 217, 226
  user exit preprocessing 225
  web Service handler 221
  XSL mapping 218, 227

data exchange
  content 13
  data export 13
  data import 13
  enterprise services 11
  external systems 11
  invocation channels 10
  object structures 8
  publish channels 9
  web services 12

data formats
  interface tables 99

E

EAR file, rebuilding 146, 213
EJB
  customization 220
endpoint
  handler 145, 232
  MXCMDLINE 144
  MXFLATFILE 144
  MXIFACETABLE 144
  MXXMLFILE 144
enterprise bean access
  securing 179
enterprise bean handler
  CONTEXTFACTORY property 145
  definition 145
  EJBJEXIT property 145
  JNDINAME property 146
  METHODNAME property 146
  PROVIDEURL property 146
  USERNAME, PASSWORD property 147
enterprise services
  configuring 137
  data exchange 11
  integration controls 234
  interface tables 234
  object structure 233
  operation type 233
  processing 233

e enterprise services (continued)
  processing rules 234

error management
  configuration 118
  error causes 124
  interface table 117
  non-queue 127
  notification 120
  research 125

error management configuration
  external system properties 119
  system properties 118

external message ID
  message tracking 111

external systems
  configuring 136
  create 236
  data exchange 11
  enterprise services 237
  interface tables and configuring 106
  modifying 237
  publish channels 237
  queues 257

EXTSYSNAME column, interface tables 103

F

field transformation
  COMBINE action 203
  REPLACE action 203
  SET action 203
  SPLIT action 203

flat files
  alias column names 99
  duplicate columns 99

FLATFILE handler
  definition 147
  formatting 148
  naming convention 147
  properties 148

FLATFILECONSUMER cron task
  parameters 135
  processing properties 135

H

handlers
  CMDLINE 144, 155
  customization 218
  definition 232
  enterprise bean 145
  FLATFILE 144, 147
  HTTP 148
  IFACETABLE 144, 150
  JMS 151
  WEBSERVICE 153
  writing 155, 156
  XMLFILE 144, 154

HASLD column 100

HTTP handler 219
  CONNECTTIMEOUT property 150
  definition 148
HTTP handler (continued)
HTTPEXIT property 149
HTTPMETHOD property 150
READTIMEOUT property 150
URL property 150
USERNAME, PASSWORD property 150
HTTP servlet
definition 181
securing 181

IFACENAME column, interface tables 100
IFACETABLE handler
   DRIVER property 151
   ISREMOTE property 150
   URL property 151
   USERNAME, PASSWORD property 151
IFACETABLECONSUMER cron task 105, 160
IFACTABLE handler
   definition 150
inbound processing
   customization 223
integration components
   endpoints 137
   enterprise services 137
   external systems 136
   invocation channels 139
   object structures 137
   publish channels 138
integration controls
   create 211
   definition 209
   enterprise service 234
   levels 209
   types 210
integration framework
   for data exchange 8
   for operational management product integration 13
   for user interface integration 15
   overview 6
integration modules
   components 264
   definition 264
   endpoints 268
get service invoker utility method 278
getPropertyInvoker property map 275
invocation 273
   invocation channel
      implementation 272
      usage 269
invoke method 276
Java class
   implementation 271
   usage 269
Java methods 272
Java type 269
logical management operation associations 266
logical management operation enablement 267
operational management product associations 266
operational management product integration 14
operational management product service method 277
parameters 267
prerequisites 266
service utility method 274
tasks 268

integration queue
   definition 178
   J2EE restrictions 178
integration schemas 77
   key fields 77
   metadata schema 79
   namespace property 78
   object schemas 84
   object structure schemas 82
   schema directory and files 79
   schema generation 78
   service level schemas 86
   xml validation 78
integration xml content
   root element 63
interface queue tables 96
interface tables
   ACTION column 103
   alias column names 99
   column name lengths 99
   configuring external systems 106
   creating 97
   customization 223
   deleting 98
   duplicate columns 99
   enterprise service 234
   EXTSYSNAME column 103
   format 99
   HASLD field 100
   IFACENAME column 100
   IFACETABLECONSUMER cron task 160
   key columns 99
   location 96
   long description column 104
   multiple queue tables 161
   naming 96
   polling 105, 159
   processing columns 100
   queue tables 96
   regenerating 98
   restrictions 98
   security 185
   selectors 160
   TRANSID column 100
   TRANSSEQ column 102
invocation channels
   configuring 139
   data exchange 10
   endpoint 236
   integration modules 269
   object structures 235
   processing 235
invoice collaboration switches 298

J

Java class
   integration module implementation 271
Java integration modules
   definition 269
Java methods
   integration module processing 272
JMS handler
   CONFACCTOR JNDINAME property 152
   CONTEXTFACTORY property 152
   definition 151
JMS handler (continued)
  DESINTATIONTYPE property 152
  DESTJNDINAME property 152
  ISCOMPRESS property 152
  JMSEXIT property 153
  PROVIDERUSER, PROVIDERPASSWORD property 153
  PROVIDEURL property 153
  USERNAME, PASSWORD property 153

JMS queues
  asynchronous inbound, cron task 133
  cluster configuration 189
  configuring 132, 164
  continuous 165
  creating 164
  predefined queues 132
  properties 133, 164
  selectors 172
  sequential 165
  sequential, cron task 133
  utilities 173
  WebSphere MQ 174

K
  key columns, interface table 99

L
  labor transaction collaboration switches 300
  land in context
    user interface integration 16
  language tables 260
  languageEnabled attribute 261
  launch entries
    define 282
    user interface integration 15
  launch entry
    properties 282
    push button, associate 286
    signature option condition, associate 287
    signature option conditions 286
    signature option, associate 285
    toolbar menu item, associate 285
    user interface control, associate 285
  launch in context
    external applications 282
    launch entries 282
    operational management product properties 284
  list controls 210
  logical management operations
    definition 265
    enablement 267
    integration 14
    integration module associations 266
  long description column, interface table 104

M
  message details
    message tracking 110
  message events 113
    inbound and outbound 114
  message processing
    SKIP action 202
    SKIPCHILDREN action 202
    SKIPRECORD action 202
    STOP action 202
  message reprocessing
    critical errors 124
    error correction 122
    error details 122
    message deletion 124
    statuses 121
  message status 113
    inbound messages 113
    outbound messages 113
  message tracking
    configuration 111
    external message ID 111
    search ID 112
    stored messages 112
    message details 110
    statuses 113
    inbound 113
    outbound 113
  MicService 184
  multilanguage attributes 261
  multi-noun message
    external message ID 112
    search ID 112
  multiple languages 259
  multiple queue tables 161
  multiplication controls 211
  MXCMDLINE endpoint 144
  MXFLATFILE endpoint 144
  MXIFACETable endpoint 144
  MXXMLFILE endpoint 144

O
  object structure element 67
    additional considerations 76
    definition 245
    field attributes 68
      action attribute 70
      changed attribute 68
      glorder attribute 68
      langenabled attribute 69
      maxvalue attribute 70
      field comparison 245
      field selection 246
      key fields 67
      range selection 247
  object structure services
    object structure 234
    object structure processing 234
  object structures
    alternate keys 231
    build 230
    configuring 137
object structures (continued)
    create 230
    data exchange 8
    enterprise service 233
    flat files 232
    interface table 232
    nonpersistent fields 231
    persistent fields 231
    required fields 231
    required objects 230
operational management product integration
    action 14
    integration modules 14
    logical management operations 14
    process management products 13
operational management products
    definition 264
    integration 15
    integration module associations 266
    launch entry 284
    service method 277
Oracle databases and long description column 104
outbound processing
    customization 214

P
polling, interface tables 105, 159
predefined components 130
process management products
    operational management product integration 13
processing conditions
    evaluation categories 205
    evaluation types 206
    field evaluations 206
    specifications 205
processing rule
    action 201
    initiation 201
    sequence 204
publish channels
    configuring 138
    data exchange 9
    integration controls 235
    interface tables 235
    object structures 235
    processing rules 235
purchase order collaboration switches 300
purchase requisition collaboration switches 302

Q
query root element
    attributes 241
query services
    definition 240
    enterprise service, create 240
    operator 243
    root element 240
queue selectors 172
queue utilities 173

R
receipt collaboration switches 304
restrictions
    adapters 236
    changed attribute 68
    deletequeue utility 173
    FLATFILE handler 147, 150
    HASDL column 100
    interface tables 98
    predefined components 130

S
schemas, XML
    directories 79
search ID
    message tracking 112
security
    enterprise bean access 179
    HTTP servlet 181
    integration queue 178
    interface tables 185
    object-level authorization 185
    outbound router handler 185
    remote integration APIs 184
    web services 183
selectors, cron task 160
sequential queues
    cron task parameters 165
    definition 165
    overview 163
    WebLogic cluster 194
    WebSphere cluster 192
services
    create 233
    enterprise 233
    modifying 236
    object structure 234
    standard 234
    web 234
standard web services 253
stored messages
    message tracking 112
synchronous inbound processing
    enterprise services 32
        enterprise service class processing 35
        enterprise service response processing 39
        object creation 38
        object processing 39
        object processing rules 38
        object structure 33
        object structure multiplication 36
        object structure processing class 38
        object structure processing rules 37
        process initiation 32
        user exit object processing 39
        user exit postprocessing 35
        user exit preprocessing 34
        XSL Map 36
object structure services 41
    object creation 42
    object processing 43
    object structure 42
Index

object structure services (continued)
  object structure processing class 42
  process initiation 41
standard services 44
  object processing 45
  process initiation 44
synchronous outbound processing
integration modules 60
invocation channels 55
  object processing 60
  request class processing 57
  request object structure 56
  request user exit preprocessing 57
  request XSL Map 58
  response user exit preprocessing 59
  response XSL Map 60
  send data, endpoint 58
  process initiation 56
system properties 131

T

TRANSID column 100
transLanguage attribute 261
translatable columns 260
TRANSSEQ column 102

U

UDDI 256
user exit postprocessing 226
  customization 217
user exit preprocessing
  customization 225
user interface integration
  land in context 16
  launch entries 15

V

value controls 210

W

web services (continued)
  UDDI registration 256
  web service definition language generation 255
  XML schema generation 254
WebLogic
  continuous queue cluster 193
  message beans 168
  sequential cluster queue 194
WEBSERVICE handler
  definition 153
  HTTPCONNTIMEOUT property 154
  HTTPREADTIMEOUT property 154
  HTTPVERSION property 154
  MEP property 153
  SERVICENAME property 153
  SOAPACTION property 153
  USERNAME, PASSWORD property 154
  WSEXIT property 154
WebSphere
  continuous queue cluster 190
  message beans 167
  sequential cluster queue 192
WebSphere MQ
  configuring endpoints, handlers 174
  configuring queues, WebSphere MQ provider 175
work order collaboration switches 305
WSDL 196, 255

X

XML schema
  generation 254
XML schemas
  directories 79
XMLFILE handler
  definition 154
FILEDIR property 154
PRETTYPRINT property 154
XMLFILECONSUMER cron task
  parameters 133
  processing properties 134
XSL mapping
  customization 218, 227