

*IBM WebSphere Business Integration Collaborations
for Product Information Management Version 1.0.2
IBM WebSphere Business Integration Collaborations
for UCCnet Message Manager Version 4.3.2*



Solution Development Guide

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices and Trademarks” on page 63.

Seventh Edition (January 2004)

This edition applies to:

Version 4, Release 3, Modification 2 of the *IBM WebSphere Business Integration Collaborations for UCCnetMessage Manager* (5724-H63)

Version 1, Modification 2 of the *IBM WebSphere Business Integration Collaborations for Product Information Management* (5724-H64)

and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send them to the following address:

IBM Canada Ltd. Laboratory
Information Development
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002, 2004. All rights reserved.

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

Contents

Solution development guide 1

Introduction 1

Who should read the Solution development guide 1

How the Solution development guide is organized 1

Definitions of connector and adapter 3

Schema and DTD terminology 3

Definitions of NULL and BLANK 4

Processing a business object: an example NEW_ITEM workflow 4

Detailed workflow: receiving, filtering, and validating a business object 4

Detailed workflow: processing a business object with cascaded GLNs. 8

Detailed workflow: completing the WebSphere MQ Workflow process and merging updated information into a complete business object 10

Detailed workflow: synchronizing a business object to a back-end system 13

Filtering publication requests before business processing 15

Filtering based on the presence of attributes required by UCCnet 16

Filtering based on items belonging to approved supply-side trading partners. 18

Filtering based on items belonging to accepted categories 19

Complex field filtering based on multiple attributes 19

Filtering to eliminate processing of duplicate items 20

Validating an item before business processing 21

Validating an item against customized business policy rules 21

Validating an item by requiring data for specific attributes 22

Using a custom missing data retrieval process to collect data 28

Adding customized code to the ItemValidation collaboration template 28

Processing business objects with cascaded GLNs 29

Using the GLN Cascade Grouping File 29

Using a business process to review and approve an item 31

Printing an item before it is sent through the business review/approval process. 31

Mapping an item to the business review/approval process 32

Using WebSphere MQ Workflow containers 32

Returning data from the business review/approval process to an ItemCollector collaboration object. 34

Merging data into a complete item 34

Merging static data by using the X_COPY ATTRIBUTE configuration properties. 37

Merging missing data by using the MISSING_DATA_CHILD_ATTRIBUTE configuration property. 37

Processing an item after the business review/approval process completes 38

Synchronizing an item to a back-end application other than a file system 39

Synchronizing an item to multiple back-end applications 40

Sending responses to UCCnet 40

Using identifier, message, and item stores 42

Persisting or deleting an item to or from a local identifier store 43

Persisting, retrieving, or deleting an item to or from a local message store 44

Persisting, retrieving, updating, or deleting an item to or from a local item store 45

Generating database keys. 46

Controlling email 47

Alerting email recipients of item status or processing errors 47

Specifying message text, subjects, and recipients in external files 55

Specifying changing individual or multiple message recipients 55

Using substitution variables in message and subject text 56

Logging 56

Logging errors 57

Logging item status 57

Logging that mail is sent 58

Tracing 58

Handling solution processing errors 58

Diagnosing error conditions 59

Recovering from error conditions 60

Handling data from other data sources 61

Extending the solution to handle a single data source other than UCCnet 61

Extending the solution to handle multiple data sources 62

Notices and Trademarks 63

Notices 63

Programming interface information 64

Trademarks and service marks 65

Solution development guide

The Product Information Management for Retailers solution workflow is composed of business objects, collaboration objects, connectors, maps, and managed workflow. These basic components work together to enable a demand-side trading partner to receive item data from a supply-side trading partner through a global data registry such as UCCnet[®], and to process the data for completeness, approval, and eventual update to back-end systems. The solution also delivers appropriate responses to the UCCnet registry. The Solution development guide describes the internal processing of the Product Information Management for Retailers solution and offers some ideas on how the solution can be customized for different purposes.

Introduction

Who should read the Solution development guide

The Solution development guide is intended for programmers who design and implement workflows using the Product Information Management for Retailers solution and who might participate in designing customizations to this solution. It assumes that users are experienced programmers and that they understand the following concepts and have experience with the software associated with them:

- Developing collaboration objects, business objects, maps, and other related components
- Installing, configuring, and operating the Product Information Management for Retailers solution
- Modeling workflows using IBM[®] WebSphere[®] MQ Workflow

Programmers must also have experience with the respective operating system on which their implementation is installed.

How the Solution development guide is organized

The Solution development guide introduces the mechanics of the solution by first presenting a sample, high-level, step-by-step workflow of how the solution handles a Catalogue Item Notification NEW_ITEM (in systems supporting the UCCnet XML Schema Definition (XSD)) or a PUB_RELEASE_NEW_ITEM (in systems supporting the UCCnet Document Type Definition (DTD)) in the section “Processing a business object: an example NEW_ITEM workflow” on page 4. Many steps contain links to detailed conceptual information about the mechanics of the solution associated with those steps. This section is useful for obtaining an overall, conceptual understanding of solution processing.

Other sections describe in detail how solution processing operates, as follows:

- “Filtering publication requests before business processing” on page 15 details the preprocessing filtering performed by a UCCnetMessageReceive collaboration object, which ensures that items contain attributes required by UCCnet, that they are from supply-side trading partners or in categories accepted by the demand-side trading partner, that duplicate items are not processed, and that items pass a set of complex filters based on the interaction of multiple attributes.
- “Validating an item before business processing” on page 21 describes the item validation processes provided by an ItemValidation collaboration object, which

include validating a Retail_Item business object against customized business policy rules and evaluating an accepted business object against a customized list of required attribute data. It also details how to use a custom missing data retrieval process to obtain required attribute data.

- “Adding customized code to the ItemValidation collaboration template” on page 28 provides information on how to add custom logic to the ItemValidation collaboration template.
- “Processing business objects with cascaded GLNs” on page 29 explains how the solution handles items containing cascaded GLNs.
- “Using a business process to review and approve an item” on page 31 describes operations related to utilizing WebSphere MQ Workflow as a custom business review/approval process. The business review/approval process is used by the demand-side trading partner to review an item to determine whether to approve it or reject it.
- “Merging data into a complete item” on page 34 describes the operation of the an ItemCollector collaboration object, which builds a complete business object by merging partial business objects containing updated information into a complete copy of the business object retrieved from a local item store.
- “Processing an item after the business review/approval process completes” on page 38 details how a Process_Reviewed_Item collaboration object synchronizes an item to a back-end file system and calls a UCCnetMessageSend collaboration object to send a response to UCCnet. It also suggests ways of synchronizing an item to a customized back-end application other than a file system or to multiple back-end applications.
- “Sending responses to UCCnet” on page 40 explains how a UCCnetMessageSend collaboration object sends response messages to UCCnet based on an item’s command type, status value, and the values of particular collaboration object properties.
- “Using identifier, message, and item stores” on page 42 identifies how and why various collaboration objects persist, retrieve, update, and delete items to and from identifier, message, and item stores.
- “Controlling email” on page 47 details how various collaboration objects enable a Role_Email collaboration object to alert users of item status or if errors occur during processing. It also describes how to specify message text, subjects, and recipients in external files to eliminate updating individual collaboration objects each time one of these variables changes. Also covered is how to specify changing individual or multiple message recipients and how to use substitution variables in message and subject text.
- “Logging” on page 56 describes the special capabilities of various collaboration objects to log errors, item status, and when mail is sent.
- “Tracing” on page 58 outlines how problems that might occur in the solution workflow can be traced and identified.
- “Handling solution processing errors” on page 58 provides trouble-shooting tips that can be used to diagnose error conditions, identify where in the process flow errors might have occurred, and recover from error conditions.
- “Handling data from other data sources” on page 61 offers suggestions for extending the solution to handle single data sources other than UCCnet or multiple data sources.

Definitions of connector and adapter

The term "connector" used throughout refers to the runtime portion of an IBM WebSphere Business Integration Adapter. References to specific connectors are related to specific adapters, as follows:

- "iSoftConnector" refers specifically to the runtime portion of an IBM WebSphere Business Integration Adapter for iSoft.
- "TPICConnector" refers specifically to the runtime portion of an IBM WebSphere Business Integration Adapter for Trading Partner Interchange.
- "JMSConnector" refers specifically to the runtime portion of an IBM WebSphere Business Integration Adapter for Java™ Message Service (JMS).
- "JTextConnector" refers specifically to the runtime portion of an IBM WebSphere Business Integration Adapter for JText.
- "JTextISoftConnector" refers specifically to the runtime portion of an adapter based on the IBM WebSphere Business Integration Adapter for JText.
- "JTextTPICConnector" refers specifically to the runtime portion of an adapter based on the IBM WebSphere Business Integration Adapter for JText.
- "JTextJMSConnector" refers specifically to the runtime portion of an adapter based on the IBM WebSphere Business Integration Adapter for JText.
- "WebSphereMQWorkflowConnector" refers specifically to the runtime portion of an IBM WebSphere Business Integration Adapter for WebSphere MQ Workflow.

The way you connect to UCCnet determines the connector that you use to communicate with it. If you exchange messages with UCCnet using an AS2/EDIINT interface protocol, you can use a TPICConnector, an ISoftConnector, or you can use WebSphere Business Integration Connect in conjunction with a JMSConnector. Use the TPI connector if you communicate with UCCnet through Trading Partner Interchange servers. Use the ISoftConnector if you communicate with UCCnet through an iSoft Peer-to-Peer Agent. Use the JMS connector if you communicate with UCCnet through WebSphere Business Integration Connect. If you exchange messages through the UCCnet Command Line Utility (CLU) or are testing your installation, you can use either a JTextTPICConnector, a JTextISoftConnector, or a JTextJMSConnector.

Since the actual connector you use is dependent on your set up, this documentation uses "AS2 channel connector" as a general term for any of the TPICConnector, iSoftConnector, JMSConnector, JTextTPICConnector, JTextISoftConnector, and JTextJMSConnector.

Schema and DTD terminology

The information in the following sections outlines how the Product Information Management for Retailers solution handles publication information notifications (in systems supporting the UCCnet DTD) or Catalogue Item Notifications (in systems supporting the UCCnet XSD). Because the collaboration templates within the Product Information Management for Retailers solution are schema-based, in DTD-based systems, the solution maps incoming DTD-based publication information notifications into XSD-based Catalogue Item Notifications during processing, and converts outgoing XSD-based Catalogue Item Confirmation commands into DTD-based Authorization commands at the end of solution processing. For simplicity, the solution will be discussed throughout using schema terminology only.

Definitions of NULL and BLANK

The terms NULL and BLANK are defined as true responses when the attributes are tested using the business object methods `isNull()` and `isBlank()`, respectively. The method `isNull()` returns true when a value has never been set in an attribute. The method `isBlank()` returns true when the attribute contains a zero-length string. An attribute containing a space character is not considered BLANK by the `isBlank()` method.

Processing a business object: an example NEW_ITEM workflow

The information in the following sections outlines the Product Information Management for Retailers solution workflow in detail. It describes at a high level how the solution handles a PUB_RELEASE_NEW_ITEM flow (in systems supporting the UCCnet DTD) or a Catalogue Item Notification NEW_ITEM flow (in systems supporting the UCCnet XSD).

The example workflow follows an item from the point when it is retrieved from UCCnet, as it is taken through filtering, validation, and WebSphere MQ Workflow business review/approval processing, and is then synchronized to a back-end system. In the context of the Product Information Management for Retailers solution, this back-end system is a file system. Any response to the item is returned to UCCnet. The workflow is presented in four segments:

- The flow from when an item is received from UCCnet until it is sent to an ItemDispatcher collaboration object for optional handling of cascaded GLNs. See the section “Detailed workflow: receiving, filtering, and validating a business object” for more information.
- The flow from when an item is received by an ItemDispatcher collaboration object until it is sent to WebSphere MQ Workflow for business review/approval processing. See the section “Detailed workflow: processing a business object with cascaded GLNs” on page 8 for more information.
- The flow from when an item enters the WebSphere MQ Workflow business review/approval process until the updated information within it is merged into a complete item by an ItemCollector collaboration object. See the section “Detailed workflow: completing the WebSphere MQ Workflow process and merging updated information into a complete business object” on page 10 for more information.
- The flow from when an item exits an ItemCollector collaboration object until it is synchronized to a back-end system. See the section “Detailed workflow: synchronizing a business object to a back-end system” on page 13 for more information.

If an error occurs, its cause and its location in its overall process flow must be determined. Processing must then be restarted. See the section “Handling solution processing errors” on page 58 for detailed information on diagnosing error conditions, determining the point of failure, and recovering from error conditions. Tracing can also be enabled for all collaboration objects to record logical flaws and data processed. See the section “Tracing” on page 58 for detailed information.

Detailed workflow: receiving, filtering, and validating a business object

The following outlines the solution workflow as an item is received from UCCnet, passes through preprocessing filtering and validating processes, and is sent to an

ItemDispatcher collaboration object for optional processing of cascaded GLNs. For more detailed process logic behind the steps in this high-level flow, refer to the following:

- For steps 1 to 8, UCCnetMessageReceive collaboration template
- For steps 4 and 5, IdentifierStore collaboration template
- For steps 6 and 7, MessageStore collaboration template
- For steps 9 to 14, ItemValidation collaboration template
- For steps 12 and 13, ItemStore collaboration template

Also, refer to the Installation guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

1. A worklist is requested from UCCnet. The AS2 channel connector receives the worklist from the AS2 channel server (iSoft Peer-to-Peer Agent, TPI server or WebSphere Business Integration Connect). This worklist can contain one or more publication information notifications (if you are supporting UCCnet DTDs) or one or more Catalogue Item Notifications (if you are supporting UCCnet XSDs).
2. The AS2 channel connector sends the worklist to the IBM WebSphere Business Integration Data Handler for XML, which converts it into an application specific business object of the form UCCnetXXX_envelope (if iSoft connectivity is used), UCCnetTPIXXX_envelope (if TPI connectivity is used), or UCCnetJMSXXX_envelope (if WebSphere Business Connect is used with JMS). This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.

Note: In this and the following steps, the variable XXX specifies the XML definition type used (DTD or XSD).

3. The business object is converted to a UCCnetGBO_envelope business object by passing through an input map of the form UCCnetXXX_envelope_to_UCCnetGBO_envelope (if iSoft connectivity is used), UCCnetTPIXXX_envelope_to_UCCnetGBO_envelope (if TPI connectivity is used) or UCCnetJMSXXX_envelope_to_UCCnetGBO_envelope (if WebSphere Business Connect is used with JMS). The DTD forms of these maps convert incoming ASBOs containing DTD-based publication information notifications into UCCnetGBO_envelope GBOs containing XSD-based Catalogue Item Notifications for ongoing processing by the solution. The AS2 channel connector checks for subscriptions to the UCCnetGBO_envelope business object by collaboration objects. Collaboration objects based on the UCCnetMessageReceive collaboration template subscribe to it. Therefore, a UCCnetMessageReceive collaboration object is passed the business object through its *FromAS2* port.
4. The UCCnetMessageReceive collaboration object parses the message, separating each instance of a Catalogue Item Notification. Because this is an example of a Catalogue Item Notification with a NEW_ITEM command, the UCCnetMessageReceive collaboration object accommodates persisting the UCCnetGBO_envelope business object to a local identifier store by first converting it to a UCCnetGBO_identifier business object and then passing it with a Create verb to an IdentifierStore collaboration object through its *ToIdentifier_Store* port.
5. The IdentifierStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, stores the identifier in the identifier store. If the item is already found in the identifier store, it is rejected. The section

“Persisting or deleting an item to or from a local identifier store” on page 43 describes in detail how a business object is persisted to the identifier store. The section “Filtering to eliminate processing of duplicate items” on page 20 describes how a UCCnetMessageReceive collaboration object prevents duplicate items from being processed.

6. The UCCnetMessageReceive collaboration object accommodates persisting the UCCnetGBO_envelope business object to a local message store by first converting it to a UCCnetGBO_storable business object and then passing it with a Create verb to a MessageStore collaboration object through its *ToMessage_Store* port.
7. The MessageStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, stores the message in the message store. The section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 describes in detail how a message is persisted to the message store.
8. The UCCnetMessageReceive collaboration object can filter the business object to ensure that it contains attributes required by UCCnet, that it came from an approved set of supply-side trading partners, that it belongs to an approved set of item categories, and that it passes a set of complex filters that are based on the interaction of multiple attributes. Information in the section “Filtering publication requests before business processing” on page 15 details how the collaboration object performs these functions. Based on whether the business object passes or fails this analysis, the collaboration object directs processing, as follows:
 - a. If the business object fails analysis, the collaboration object converts the UCCnetGBO_envelope business object into a Retail_Item business object containing the individual Catalogue Item Notification and its data. It attaches a status to the business object preconfigured in its FILTER_FAIL_RESPONSE property and sends it to its *ToRetail_Response* port to be passed to a UCCnetMessageSend collaboration object for transmittal to UCCnet.
 - b. If the business object passes analysis, the collaboration object converts the UCCnetGBO_envelope business object into a Retail_Item business object containing the individual Catalogue Item Notification and its data. It attaches a status of Review to the business object and sets the command type according to the original message type received from UCCnet. It then sends this business object to its *ToRetail_Processing* port to be passed to an ItemValidation collaboration object.

For the purposes of this example, assume this business object passed analysis.

9. The ItemValidation collaboration object accepts the Retail_Item business object on its *From* port and executes any code added to the template to evaluate it according to customized business policy rules. The section “Validating an item against customized business policy rules” on page 21 details how the collaboration object performs this function. The business policy logic code must change the business object status to a value of either Review or Rejected, which directs further processing by the collaboration object, as follows:
 - a. If the Retail_Item business object status is *not* set to Rejected by the business policy logic, the ItemValidation collaboration object continues to process it.
 - b. If the Retail_Item business object status is set to Rejected by the business policy logic, the collaboration object aborts any further processing and returns the business object to the caller through its *From* port.

For the purposes of this example, assume that the `Retail_Item` business object's status was *not* set to `Rejected` by the business policy logic.

10. The `ItemValidation` collaboration object checks that those particular attributes of the business object that the user has specified must contain data are not `NULL` or `BLANK` (see the section “Definitions of `NULL` and `BLANK`” on page 4 for a definition of these terms in the context of the `Product Information Management for Retailers` solution). Simple or complex filtering can be used to conditionally specify which attributes must contain data. The section “Validating an item by requiring data for specific attributes” on page 22 details how the collaboration object performs this function. Based on the results of this check, the collaboration object handles the `Retail_Item` business object, as follows:
 - a. If all required attribute data is present, the `ItemValidation` collaboration object continues to process it.
 - b. If the business object is missing data for any specified `Retail_Item` attribute, the collaboration object adds the attribute name to another `Retail_Item` business object list attribute, which is specified in the collaboration object's `CUST_DATA_MISS_ATTR` configuration property (by default, `internals.customer_data_missing_attributes`) and continues to process it.
11. The `ItemValidation` collaboration object executes any customized code the user has added to the template for the command associated with the business object. The section “Adding customized code to the `ItemValidation` collaboration template” on page 28 details how the collaboration object performs this function.
12. If configured to do so, the `ItemValidation` collaboration object enables persistence of the business object to a local item store by sending it with a `Create` verb to its `LocalItemStore` port to be passed to an `ItemStore` collaboration object.
13. The `ItemStore` collaboration object receives the business object on its `From` port and, through a series of interactions with the `IBM WebSphere Business Integration Data Handler for XML`, stores it in the item store database. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is persisted to the item store.
14. The `ItemValidation` collaboration object sets the business object's status to `Review`. It processes the business object according to whether it is missing data for any specified `Retail_Item` attribute, as follows:
 - a. If the object is missing required attribute data, the collaboration object passes it to its `ToMissingData` port to trigger a process for obtaining the missing data. The `ToMissingData` port can be connected to another collaboration object or to an asynchronous process for collecting the missing attribute values. This custom missing data retrieval process is described in the section “Using a custom missing data retrieval process to collect data” on page 28.
 - b. If all required attribute data is present, the collaboration object passes it to its `To` port to trigger an `ItemDispatcher` collaboration object.

For the purposes of this example, to be continued in the section “Detailed workflow: processing a business object with cascaded GLNs” on page 8, assume that all required data is present in the business object.

An `ItemValidation` collaboration object can be configured to initiate notification if errors are detected during processing. It connects through its `Notify` port to a

Role_Email collaboration object, which actually controls the email. See the section “Controlling email” on page 47 for more information.

An ItemValidation collaboration object can also log the business object being processed when an error occurs (in addition to the error) and when item status values are Review and Rejected. A Role_Email collaboration object can log when errors occur and each time an email message is sent. See the section “Logging” on page 56 for detailed information.

Detailed workflow: processing a business object with cascaded GLNs

The following outlines the solution workflow as an item is accepted and processed by an ItemDispatcher collaboration object. Based on whether the item has any cascaded GLNs within it, one or more items are then sent to a WebSphere MQ Workflow process for review/approval.

For more detailed process logic behind the steps in this high-level flow, refer to the following:

- For steps 1 to 15, ItemDispatcher collaboration template
- For steps 6, 7, 10, and 11, MessageStore collaboration template
- For steps 8, 9, 14, and 15, ItemStore collaboration template

Also, refer to the Installation guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

1. An ItemDispatcher collaboration object is triggered by the receipt of a Retail_Item business object on its *From* port.
2. The ItemDispatcher collaboration object examines the item to see if it contains cascaded GLNs and then directs processing, as follows:
 - If no cascaded GLNs exist, the collaboration object sends the item through its *To* port to the WebSphere MQ Workflow process, and processing by the ItemDispatcher collaboration object ends.
 - If cascaded GLNs do exist, the collaboration object continues processing the item.

For the purposes of this example, assume that cascaded GLNs do exist.

3. The ItemDispatcher collaboration object checks if a filename exists in its GLN_CASCADE_GROUPING_FILE property and directs processing, as follows:
 - If this property is empty, the collaboration object uses the value of the GLN_CASCADE_GROUPING_DEFAULT configuration property to guide on-going processing.
 - If a filename exists in this property, the collaboration object verifies that the file exists and parses correctly. Based on the results of this check, it guides processing, as follows:
 - If the file exists and parses correctly, the collaboration object continues processing the item.
 - If the file does not exist or does not parse correctly, the collaboration object raises an exception, sends the item to its *Notify* port with appropriate error attributes filled in, and processing ends.

For the purposes of this example, assume that a filename and file exist and that the file parses correctly.

4. The ItemDispatcher collaboration object checks if the RetailUtility class file specified in its UTILITY_CLASS property exists. It then directs processing, as follows:
 - If a file exists, the collaboration object continues processing the item.
 - If a file does not exist, the collaboration object raises an exception, sends the item to its *Notify* port with appropriate error attributes filled in, and processing ends.

For the purposes of this example, assume that a RetailUtility class file exists.

5. The ItemDispatcher collaboration object uses the information in the file specified in its GLN_CASCADE_GROUPING_FILE property to determine how to group the cascaded GLNs. See the section “Using the GLN Cascade Grouping File” on page 29 for more information about the content of this file.
6. The ItemDispatcher collaboration object reads the value of the `internals.correlationID` attribute from the `Retail_Item`, creates a `UCCnetGBO_storable` business object with this information, and sends the `UCCnetGBO_storable` business object with a Retrieve verb through its *LocalMessageStore* port in order to request the message store information from a MessageStore collaboration object.
7. The MessageStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, retrieves the message from the message store database and returns it with a Create verb to the ItemDispatcher collaboration object over its *LocalMessageStore* port. The section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 describes in detail how a message is retrieved from the message store.
8. For each cascaded GLN identified in Step 5, which will be sent to the WebSphere MQ Workflow process, the ItemDispatcher collaboration object creates one copy of the `Retail_Item` business object. It removes the extra cascaded GLNs from each copy and sends each to its *LocalItemStore* port with a Create verb to an ItemStore collaboration object. The key used for each new item is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value of the newly created business object.
9. The ItemStore collaboration object receives each new business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, stores each in the item store database. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is persisted to the item store.
10. For each cascaded GLN identified in Step 5, which will be sent to the WebSphere MQ Workflow process, the ItemDispatcher collaboration object creates one copy of the retrieved message store information. It removes the extra cascaded GLNs from each item and sends each to its *LocalMessageStore* port with a Create verb to a MessageStore collaboration object. The key used for each new message is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value of the newly created business object.
11. The MessageStore collaboration object receives each new business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, stores each message in the

message store. The section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 describes in detail how a message is persisted to the message store.

12. The collaboration object creates a new copy of the triggering Retail_Item business object for each group, removing extra cascaded GLNs that do not belong in the group from each Retail_Item.
13. The new Retail_Item business objects are sent through the *To* port to the WebSphere MQ Workflow process for approval.
14. The original Retail_Item business object is updated with the number of cascaded items sent to the WebSphere MQ Workflow process and is then sent to the *LocalItemStore* port with an Update verb to an ItemStore collaboration object.
15. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, updates the item store database. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is updated in the item store.

This workflow is continued in the section “Detailed workflow: completing the WebSphere MQ Workflow process and merging updated information into a complete business object.”

An ItemDispatcher collaboration object can be configured to initiate notification if errors are detected during processing. It connects through its *Notify* port to a Role_Email collaboration object, which actually controls the email. See the section “Controlling email” on page 47 for more information.

An ItemDispatcher collaboration object can also log when errors occur. See the section “Logging” on page 56 for detailed information.

Detailed workflow: completing the WebSphere MQ Workflow process and merging updated information into a complete business object

The following outlines the solution workflow as a business object passes through the WebSphere MQ Workflow business review/approval process and is delivered to an ItemCollector collaboration object, where the new information is merged into a complete business object. For more detailed process logic behind the steps in this high-level flow, refer to the following:

- For steps 1 to 5, IBM WebSphere MQ Workflow documentation
- For steps 6, 7, 9, and 11, ItemStore collaboration template
- For steps 6 to 11, ItemCollector collaboration template
- For step 11, MessageStore collaboration template

Also, refer to the Installation guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

1. The WebSphere MQ Workflow business review/approval process begins when an ItemDispatcher collaboration object passes a Retail_Item to the WebSphereMQWorkflowConnector over its *To* port through the Retail_Item_to_MQWF_Retail_Item map. At this point, the flow becomes asynchronous. A Retail_Item can contain multiple GLNs.

2. The WebSphereMQWorkflowConnector builds a WebSphere MQ Workflow container and invokes a specific WebSphere MQ Workflow process. The sections “Mapping an item to the business review/approval process” on page 32 and “Using WebSphere MQ Workflow containers” on page 32 describe how business objects are mapped into WebSphere MQ Workflow and how process definitions and containers are used in the context of the Product Information Management for Retailers solution.
3. The approver(s) responsible for approving the item set the item status to Approved, Rejected, Accepted, or Review through a customized user interface with WebSphere MQ Workflow. If an item contains multiple GLNs, the approver must set the status for each GLN in the item.
4. The WebSphere MQ Workflow process sends the container with the updated status to the WebSphereMQWorkflowConnector. At this point, the WebSphere MQ Workflow business review/approval process is complete and the flow becomes synchronous again.
5. The WebSphereMQWorkflowConnector maps the container back into a partial Retail_Item business object by passing it through the MQWF_Retail_Item_to_Retail_Item map and invokes an ItemCollector collaboration object. The section “Returning data from the business review/approval process to an ItemCollector collaboration object” on page 34 describes how updated information in this partial object is returned to the ItemCollector collaboration object.
6. The ItemCollector collaboration object accepts the partial Retail_Item business object on its *From* port and retrieves the complete copy of the Retail_Item that was stored by the ItemValidation collaboration object in the item store database before the review/approval process was started. It retrieves the item by sending the business object and a Retrieve verb to its *local_store* port to be passed to an ItemStore collaboration object.
7. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, retrieves the item from the item store database and returns it to the ItemCollector collaboration object. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is retrieved from the item store.
8. The collaboration object checks if the original Retail_Item retrieved from the item store database contained cascaded GLNs. Assume for the purposes of this example flow that cascaded GLNs are present. (See the section “Merging data into a complete item” on page 34 for a detailed description of how the solution handles business objects without cascaded GLNs.)
9. The ItemCollector collaboration object loops through the GLNs, processing each one, as follows:
 - a. It retrieves the copy of the Retail_Item that was stored by the ItemDispatcher collaboration object from the item store database by sending the business object and a Retrieve verb to its *local_store* port to be passed to an ItemStore collaboration object. The key used to access the item is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value for the GLN being processed.
 - b. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, retrieves the item from the item store database and returns it to the ItemCollector collaboration object.

- c. The ItemCollector collaboration object merges the new data received from the WebSphere MQ Workflow process into the Retail_Item business object retrieved from the item store. The section “Merging data into a complete item” on page 34 describes in detail how the ItemCollector collaboration object merges the data into a complete item.
 - d. The ItemCollector collaboration object then handles the business object according to its status value. Assume for the purposes of this example flow that the status value of the `internals.cascadedGlns.item_status` attribute is Approved. In this case, the ItemCollector collaboration object routes the merged Retail_Item business object to a Process_Reviewed_Item collaboration object via its *To* port. See the section “Merging data into a complete item” on page 34 for information on how the collaboration object handles items with status values of Accepted, Rejected, and Review.
10. The ItemCollector collaboration object calculates how many total cascaded GLNs have completed processing (i.e., they have Approved, Rejected, or Error status). If all of the cascaded GLNs from the original message that was sent to the ItemDispatcher collaboration object have completed processing, it does the following:
- a. Deletes the entry from the message store database that was stored by the UCCnetMessageReceive collaboration object by converting the Retail_Item to a UCCnetGBO_storable business object and sending it with a Delete verb to its *message_store* port to be passed to an MessageStore collaboration object. The key used to access the item is the `internals.correlationID` attribute value of the original triggering business object.
 - b. The MessageStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, deletes the message from the message store.
 - c. Deletes the copy of the Retail_Item that was stored by the ItemValidation collaboration object from the item store database by sending the business object and a Delete verb to its *local_store* port to be passed to an ItemStore collaboration object. The key used to access the item is the `internals.correlationID` attribute value of the original triggering business object.
 - d. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, deletes the item from the item store database.

This workflow is continued in the section “Detailed workflow: synchronizing a business object to a back-end system” on page 13.

An ItemCollector collaboration object can be configured to initiate notification if errors are detected during processing. It connects through its *email* port to a Role_Email collaboration object, which actually controls the email. See the section “Controlling email” on page 47 for more information.

An ItemCollector collaboration object can also log the business object being processed when an error occurs (in addition to the error). A Role_Email collaboration object can log when errors occur and each time an email message is sent. See the section “Logging” on page 56 for detailed information.

Detailed workflow: synchronizing a business object to a back-end system

The following outlines the solution workflow as a business object passes from an ItemCollector collaboration object to a Process_Reviewed_Item collaboration object, which synchronizes the business object to the back-end system and initiates sending a response to UCCnet through a UCCnetMessageSend collaboration object. For more detailed process logic behind the steps in this high-level flow, refer to the following:

- For steps 1 to 6, Process_Reviewed_Item collaboration template
- For steps 2, 3, 5, and 6, ItemStore collaboration template
- For steps 7 to 15, UCCnetMessageSend collaboration template
- For steps 7, 8, 11, 12, MessageStore collaboration template

Also, refer to the Installation guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

1. An ItemCollector collaboration object passes the Retail_Item business object to a Process_Reviewed_Item collaboration object, which receives it on its *From* port.
2. If configured to do so, the Process_Reviewed_Item collaboration object enables deletion of the item from the item store by passing the business object with a Delete verb to its *local_store* port to be passed to an ItemStore collaboration object.
3. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, deletes the item from the item store. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is deleted from the item store.
4. The Process_Reviewed_Item collaboration object directs processing according to the status of the business object. For the purposes of this example, assume that the status of the business object is Approved. (See the section “Processing an item after the business review/approval process completes” on page 38 for a detailed description of how the collaboration object handles processing for business objects with a status of Review, Accepted, reprocess, Rejected, or Error.) Since, in this example, the Retail_Item business object status is Approved, the following operations occur:
 - a. The business object is routed to the JTextConnector via the Process_Reviewed_Item collaboration object’s *Sync* port. The JTextConnector writes the Retail_Item business object to a file system. If this is successful, the item status is changed to Synchronized; if not, the item status is changed to Error.
 - b. The business object is sent to the *respond_to* port to be passed to a UCCnetMessageSend collaboration object. Alternatively, the Retail_Item can be sent to a single application other than the JTextConnector or to multiple back-end applications. For more information on these topics, see the sections “Synchronizing an item to a back-end application other than a file system” on page 39 and “Synchronizing an item to multiple back-end applications” on page 40, respectively.
5. Optionally, the Process_Reviewed_Item collaboration object rewrites the Retail_Item to the local item store by sending the business object and a Create verb to the *local_store* port to be passed to an ItemStore collaboration object.

6. The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, stores the item in the item store. The section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 describes in detail how a business object is persisted to the item store.
7. The UCCnetMessageSend collaboration object receives the Retail_Item business object on its *FromRetail* port. It then retrieves the layer information from the local message store by converting the Retail_Item to a UCCnetGBO_storable business object and passing it with a Retrieve verb to a MessageStore collaboration object through its *ToMessage_Store* port.
8. The MessageStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, retrieves the message from the message store database and returns it to the UCCnetMessageSend collaboration object. The section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 describes in detail how a message is retrieved from the message store.
9. The UCCnetMessageSend collaboration object extracts the Catalogue Item Notification command from the UCCnetGBO_storable business object and handles the object according to its original command type, its status value, and the values of certain properties. For the purposes of this example, assume that the original Catalogue Item Notification was a NEW_ITEM and the status is Approved. See the section “Sending responses to UCCnet” on page 40 for a detailed description of how the collaboration object handles business objects with other types of notification, status, and property values. The UCCnetMessageSend collaboration object builds a Catalogue Item Confirmation command message, composes the UCCnetGBO_envelope business object by first converting the Retail_Item and UCCnetGBO_storable business objects into a UCCnetGBO_RI_S business object, and then passing this object through the map specified in its TOAS2_RESPONSE_MAP configuration property. The resulting UCCnetGBO_envelope business object contains all of the information needed for the UCCnet Catalogue Item Confirmation message.
10. The UCCnetMessageSend collaboration object passes the UCCnetGBO_envelope business object to the AS2 channel connector on its *ToAS2_Response* port. In the connector controller portion of the connector, it is converted to an application specific business object of the form UCCnetXXX_envelope (if iSoft connectivity is used), UCCnetTPIXXX_envelope (if TPI connectivity is used), or UCCnetJMSXXX_envelope (if WebSphere Business Connect is used with JMS). The UCCnetGBO_envelope business object is converted by passing through either the UCCnetGBO_envelope_to_UCCnetXXX_envelope map (if iSoft connectivity is used), the UCCnetGBO_envelope_to_UCCnetTPIXXX_envelope map (if TPI connectivity is used), or the UCCnetGBO_envelope_to_UCCnetJMSXXX_envelope map (if WebSphere Business Connect is used with JMS). The DTD forms of these maps convert outgoing UCCnetGBO_envelope GBOs containing XSD-based Catalogue Item Confirmation commands into ASBOs containing DTD-based Authorization commands.

Note: The variable XXX specifies the XML definition type used (DTD or XSD).

11. The UCCnetMessageSend collaboration object enables the deletion of the entry from the message store by converting the Retail_Item to a

UCCnetGBO_storable business object and passing it with a Delete verb to a MessageStore collaboration object through its *ToMessage_Store* port.

12. The MessageStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML, deletes the message from the message store. The section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 describes in detail how a message is deleted from the message store.
13. The AS2 channel connector calls the IBM WebSphere Business Integration Data Handler for XML to produce the XML message.
14. The AS2 channel connector passes this message to the iSoft Peer-to-Peer Agent, the TPI server or to WebSphere Business Integration Connect.
15. An AUTHORIZED authorization command for DTDs or SYNCHRONISED Catalogue Item Confirmation command for XSDs is sent to UCCnet.

A Process_Reviewed_Item collaboration object can be configured to initiate notification if an item has a status of Approved, Accepted, or Rejected or if errors are detected during processing. It connects through its *mail* port to a Role_Email collaboration object, which actually controls the email. See the section “Controlling email” on page 47 for more information.

A Process_Reviewed_Item collaboration object can also log the business object being processed when an error occurs (in addition to the error) and when item status values are Approved, Accepted, and Rejected. A Role_Email collaboration object can log when errors occur and each time an email message is sent. See the section “Logging” on page 56 for detailed information.

Filtering publication requests before business processing

Catalogue Item Notifications that arrive from UCCnet can be filtered before business processing occurs to ensure that certain attributes required by UCCnet are present, that items are from specified supply-side trading partners, that items are in categories accepted by the demand-side trading partner, or that items pass complex filters that are based on multiple attributes.

Catalogue Item Notifications with NEW_ITEM, DATA_CHANGE, WITHDRAW, and DE_LIST commands can also be checked against an existing identifier store to eliminate processing of duplicate items. All of these filtering operations are performed by a UCCnetMessageReceive collaboration object.

The following sections describe the filtering operations in more detail:

- “Filtering based on the presence of attributes required by UCCnet” on page 16
- “Filtering based on items belonging to approved supply-side trading partners” on page 18
- “Filtering based on items belonging to accepted categories” on page 19
- “Complex field filtering based on multiple attributes” on page 19
- “Filtering to eliminate processing of duplicate items” on page 20

For detailed information on the collaboration objects and business objects mentioned in these sections, see the following:

- IdentifierStore collaboration template
- UCCnetMessageReceive collaboration template
- UCCnetGBO_envelope business object

- UCCnetGBO_identifier business object

Filtering based on the presence of attributes required by UCCnet

To enable a UCCnetMessageReceive collaboration object to filter Catalogue Item Notifications for the presence of attributes required by UCCnet, create an external text file and list those UCCnetGBO_envelope business object attributes that must be present and not contain NULL or BLANK values. Specify the path and name of this file in the UCCnetMessageReceive collaboration object's REQUIRED_ATTRIBUTE_FILE configuration property.

This external text file can contain one of the following:

- A simple list of the fully qualified attributes in the UCCnetXSD_envelope_notification child business object of the UCCnetGBO_envelope business object that must be present (one attribute per line).
- Complex filters that conditionally specify the list of attributes.

The first line of the file determines whether simple or complex filtering is performed. If the first line contains a single fully qualified business object attribute name, simple filtering is performed on the rest of the attributes listed in the file. If the first line of the file contains a separator of FILTER or FIELDS, complex filtering is assumed.

To provide complex filtering, the file must contain any number of sections, each section containing FILTER and FIELDS subsections. "Filterless" FIELDS subsections can also be included.

Each FILTER subsection filters on one or more business object attributes. A FILTER subsection contains the following:

- A fully qualified business object attribute name on a single line.
- On the next single line, a comma-delimited list of valid values for the attribute.
- If filtering on multiple attributes, on the next single line, a value of AND.
- On the next single line, another fully qualified attribute name.
- On the next single line, a comma-delimited list of valid values for that attribute.

Any number of attributes can be filtered in this way.

Each FILTER subsection must be followed by a FIELDS subsection. A FIELDS subsection must contain the required attributes for the filter above it, each attribute on a separate single line. A FIELDS subsection can be followed by a FILTER subsection, another FIELDS subsection without a filter, or an optional END separator at the end of the file. If a FIELDS subsection is specified without a FILTER subsection in front of it, the list of attributes in that FIELDS subsection is always required.

The UCCnetMessageReceive collaboration object reads through the text file examining the first line. If the line contains a fully qualified business object attribute name, the collaboration object checks all attributes named in the specified file. If any of the listed attributes are missing from the business object or are present in the business object but missing data, the collaboration object logs the business object and handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property.

If the first line of the file contains a FILTER or FIELDS separator, the collaboration object reads the file until it finds a FILTER subsection that is satisfied by the attribute values contained in the business object being processed. The collaboration object then checks all attributes in the business object against the list of attributes included in the FIELDS subsection related to the satisfied filter. If any of the attributes in the FIELDS subsection are missing from the business object or are present in the business object but missing data, the collaboration object logs the business object and handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property.

If no file is specified in the REQUIRED_ATTRIBUTE_FILE property, or if no attributes are indicated within the file, all items are accepted for further processing by the collaboration object.

An example file that demonstrates complex filtering and its interpretation follow:

```

FILTER
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
00011112222333
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemUnitDescriptor
CASE,PALLET
FIELDS
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.netWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.grossWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.height
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.width
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.depth
FILTER
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
00011112222333
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemUnitDescriptor
BASE_UNIT_OR_EACH
FIELDS
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.netWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.grossWeight
END

```

The collaboration object checks if the business object being processed has a gln attribute value of 00011112222333 and a tradeItemUnitDescriptor attribute value of CASE or PALLET. If it doesn't, the collaboration proceeds to the next FILTER. If it does, the collaboration object then checks if the business object being processed contains the attributes netWeight, grossWeight, height, width, and depth, and that these attributes are not NULL or BLANK. If the business object passes this filter,

collaboration object processing continues. If any of the attributes in the FIELDS subsection are missing from the business object or are present in the business object but missing data, the collaboration object logs the business object and handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property.

The collaboration object then proceeds to the next FILTER subsection. It checks if the business object has a gln value of 00011112222333 and a tradeItemUnitDescriptor attribute value of BASE_UNIT_OR_EACH. If it does, the collaboration object then checks if the business object being processed contains the attributes netWeight and grossWeight, and that these attributes are not NULL or BLANK. If the business object passes this filter, collaboration object processing continues. If any of the attributes in the FIELDS subsection are missing from the business object or are present in the business object but missing data, the collaboration object logs the business object and handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property.

Filtering based on items belonging to approved supply-side trading partners

To enable a UCCnetMessageReceive collaboration object to filter Catalogue Item Notifications to ensure that they come from supply-side trading partners approved by the demand-side trading partner, create an external text file and list those trading partners from whom items are accepted. Specify the path and name of this file in the UCCnetMessageReceive collaboration object's VENDOR_FILE configuration property.

Each entry in this file must have the following structure and be followed by a line return:

```
vendorGLN,other_data,other_data,...
```

In this structure, *vendorGLN* is the GLN (assigned by UCCnet) of an accepted supply-side trading partner. This value is found in the `entityIdentification.globalLocationNumber.gln` field of the `UCCnetXSD_envelope_notification` child business object of the `UCCnetGBO_envelope` business object. The variables called *other_data* are other data entries that can be linked to the vendor GLN. The vendor GLN must be the first entry in the line. Any number of associated attributes can be appended to the line as long as they are separated by commas (.). Do not place a comma before the vendor GLN. The following is an example of a valid entry:

```
00011112222333,TestVendor,(111)111-1111,contact@TestVendor.com
```

The collaboration object checks the `entityIdentification.globalLocationNumber.gln` field of the `UCCnetXSD_envelope_notification` child business object. If the supply-side trading partner specified in this business object attribute is also listed in the file specified by the collaboration object's VENDOR_FILE property, the collaboration object processes the business object normally. If the supply-side trading partner specified in the business object attribute is not listed in this file, the collaboration object logs it and handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property. If no file is specified in the VENDOR_FILE property, all items are accepted for further processing by the collaboration object.

Filtering based on items belonging to accepted categories

To enable a UCCnetMessageReceive collaboration object to filter Catalogue Item Notifications to ensure that they belong to categories accepted by the demand-side trading partner, create an external text file and list those categories from which items are accepted. Specify the path and name of this file in the UCCnetMessageReceive collaboration object's CATEGORY_FILE configuration property.

Each line of this file must contain only one category value. Each entry can include embedded special characters, such as periods (.) and commas (,), as long as the characters are valid within the category. Do not place characters in the line other than those specified in the category. The following is an example of a valid entry:

```
0001.0001.001
```

The collaboration object checks the `notificationDetail.catalogueItemNotification.catalogueItem.tradeItem.tradeItemInformation.classificationCategoryCode.additionalClassification.additionalClassificationCategoryCode` field of the UCCnetXSD_envelope_notification child business object of the UCCnetGBO_envelope business object. If the value specified in this business object attribute is also listed in the file specified in the collaboration object's CATEGORY_FILE property, the collaboration object processes it normally. If the value listed in the business object attribute is not from a category listed in this file, the collaboration object handles it according to the value specified in its FILTER_FAIL_RESPONSE configuration property. If no file is specified in the CATEGORY_FILE property, all items are accepted for further processing by the collaboration object.

Complex field filtering based on multiple attributes

To enable a UCCnetMessageReceive collaboration object to filter Catalogue Item Notifications based on a complex set of field criteria, create an external text file that includes the complex filtering information. Specify the path and name of the file in the UCCnetMessageReceive collaboration object's COMPLEX_FILTER_FILE configuration property. This file can contain any number of sections on which the collaboration will filter, each section containing a fully qualified business object attribute and a comma-delimited list of valid values for the attribute. The collaboration object can search on multiple attribute name/value pairs combined with an AND separator, or on mutually exclusive groups of attribute name/value pairs separated by a NEXT separator. A file must be composed as follows:

- The first line contains a fully qualified attribute name on a single line.
- On the next single line, a comma-delimited list of valid values for the attribute.
- On the next single line, one of the following values:
 - AND – A value of AND is followed by another fully qualified attribute name on a single line followed by a comma-delimited list of valid values for it on the following single line. Any number of attributes can be checked together in this way.
 - NEXT – A value of NEXT is followed by another set of attributes and values combined with an AND separator.
 - END – A value of END signifies the end of the list of complex filters.

The UCCnetMessageReceive collaboration object reads through the Complex Filter File until it finds an attribute name/value pair that is satisfied by the attribute name/value contained in the business object being processed. If the business object

passes complex filtering, the collaboration object processes it normally. If the business object does not pass complex filtering, the collaboration handles it according to the value specified in the `FILTER_FAIL_RESPONSE` configuration property. If no file is specified in the `COMPLEX_FILTER_FILE` property, all items are accepted for further processing by the collaboration object. If an item satisfies the conditions of multiple filters, only the first filter in the file is ever considered.

An example file and its interpretation follow:

```
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
00011112222333
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.classificationCategoryCode.additionalClassification[0]. \
additionalClassificationCategoryCode
0001.001.001
NEXT
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
00011112222334
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.classificationCategoryCode.additionalClassification[0]. \
additionalClassificationCategoryCode
0001.001.005
END
```

The collaboration object checks if the business object being processed has a `gln` attribute value of 00011112222333 and an `additionalClassificationCategoryCode` attribute value of 0001.001.001. If it does, the business object is accepted and collaboration object processing continues.

If these first filter conditions were not satisfied, the collaboration object now proceeds to the attribute name/value following the `NEXT` separator. It checks if the business object has a `gln` value of 00011112222334 and an `additionalClassificationCategoryCode` attribute value of 0001.001.005. If it does, the business object is accepted and collaboration object processing continues. If it does not, the collaboration ends processing.

Filtering to eliminate processing of duplicate items

A `UCCnetMessageReceive` collaboration object filters Catalogue Item Notifications with `NEW_ITEM`, `DATA_CHANGE`, `WITHDRAW`, or `DE_LIST` commands to ensure that they are not duplicate items.

During normal item processing, a `UCCnetMessageReceive` collaboration object enables persistence of the `UCCnetGBO_envelope` business object to a local identifier store by converting it to a `UCCnetGBO_identifier` business object and passing it to its `ToIdentifier_Store` port. This port is connected to an `IdentifierStore` collaboration object. The `IdentifierStore` collaboration object performs the actual storage operation. See the section “Persisting or deleting an item to or from a local identifier store” on page 43 for more information on how items are persisted to an identifier store.

Before sending the object to this port, the collaboration object checks the value of its `FILTER_DUPLICATE` configuration property. If the value is `true` (which is the default value), the collaboration object checks if an item with identical key attribute values exists in the identifier store. If you are processing DTDs the keys are the `gtin`, `version`, and `topic` attributes. If you are processing XSDs, the keys are the

gtin, topic, dataRecipientGLN, dataSourceGLN, targetMarket and uniqueCreatorID attributes. If an identical item does exist, the second entry with identical information is logged as a duplicate and further processing of it ends. If an identical item does not exist, the item is added to the local identifier store and processed normally.

Validating an item before business processing

A Retail_Item business object created from an Catalogue Item Notification can be subjected to validation processes provided by code in the ItemValidation collaboration template. A collaboration object based on this template is used to accept or reject a Retail_Item business object based on customized business policy rules, to evaluate an accepted business object based on a customized list of required attribute data, and to direct the business object to the appropriate port based on the results of the evaluation. After an item completes validation successfully, the collaboration object changes its status to Review and sends the item to its To port. This port is connected to an ItemDispatcher collaboration object. The following sections describe the validation operations in more detail:

- “Validating an item against customized business policy rules”
- “Validating an item by requiring data for specific attributes” on page 22
- “Using a custom missing data retrieval process to collect data” on page 28

For detailed information on the collaboration objects and business object mentioned in these sections, see the following:

- ItemCollector collaboration template
- ItemStore collaboration template
- ItemValidation collaboration template
- Process_Reviewed_Item collaboration template
- UCCnetMessageSend collaboration template
- Retail_Item business object

Validating an item against customized business policy rules

To accept or reject an item based on existing business policies, insert customized code into the ItemValidation collaboration template’s Business Policy Processing subdiagram. Enable the logic in this subdiagram to execute for specific commands attached to a Retail_Item by specifying those commands in an ItemValidation collaboration object’s BUSINESS_POLICY_CMDS configuration property.

The pre-existing subdiagram logic examines the value for the Retail_Item business object’s attribute named in the ItemValidation collaboration object’s ITEM_COMMAND_ATTRIBUTE configuration property against the values specified in its BUSINESS_POLICY_CMDS configuration property. If the value for the attribute in the ITEM_COMMAND_ATTRIBUTE property is specified in the BUSINESS_POLICY_CMDS property, the ItemValidation collaboration object executes the customized code in the Business Policy Processing subdiagram.

The rules in the customized code must specify whether the Retail_Item is accepted or rejected for subsequent processing by changing the value of its attribute named in the ItemValidation collaboration object’s ITEM_STATUS_ATTRIBUTE configuration property. If the item is accepted, the code must change the value of the business object attribute to Review; if the item is rejected, it must change the value to Rejected. The pre-existing subdiagram logic uses this attribute value to determine whether to return the item to the caller or to continue processing it. If

the item is rejected, the business policy subdiagram raises an exception to abort the item processing and returns it to the main scenario, where the item rejection processing is handled.

Validating an item by requiring data for specific attributes

A standard message from UCCnet might not contain all of the data required by an implementation. In this case, certain Retail_Item business object attributes must be identified for which data must exist (the attributes must not be NULL or BLANK) before an item is accepted for processing. Pre-existing code in the ItemValidation template's File Missing Attribute Logic subdiagram enables this type of item validation.

To use this type of validation, do the following:

1. Create a text file that contains one of the following:
 - A simple list of the fully qualified attributes in the Retail_Item business object that must be present (one attribute per line).
 - Complex filters that conditionally specify the list of attributes.

The first line of the file determines whether simple or complex filtering is performed. If the first line contains a single fully qualified business object attribute name, simple filtering is performed on the rest of the attributes listed in the file. If the first line of the file contains a separator of FILTER or FIELDS, complex filtering is assumed. See the section "Performing simple filtering" on page 23 for instructions on creating a file for simple filtering; the section "Performing complex filtering" on page 24 for instructions on creating a file for complex filtering.
2. Specify the fully qualified name of this text file in the REQUIRED_ATTRIBUTE_FILE configuration property.
3. Enable the logic in the File Missing Attribute Logic subdiagram to execute for specific commands attached to a Retail_Item by specifying those commands in the REQUIRED_ATTRIBUTE_CMDS configuration property. Pre-existing logic examines the value of the Retail_Item business object's attribute, named in the configuration property ITEM_COMMAND_ATTRIBUTE, against the values specified in the REQUIRED_ATTRIBUTE_CMDS configuration property. If the value of the attribute named in the ITEM_COMMAND_ATTRIBUTE property is specified in the REQUIRED_ATTRIBUTE_CMDS property, the ItemValidation collaboration object executes the code.
4. Since the data collection process can be asynchronous or the tool used to perform it (such as a WebSphere MQ Workflow container) might be limited in size and not able to persist all of the attributes of a Retail_Item business object, save a complete copy of the Retail_Item business object in an item store before it is passed to the ToMissingData port for processing. This can be accomplished by setting the value of the ItemValidation collaboration object's RETAIN_ITEM_IN_LOCAL_STORE property to true. The collaboration object then calls an ItemStore collaboration object, which actually performs the storage operation.

Note: The value of the ItemValidation collaboration object's RETAIN_ITEM_IN_LOCAL_STORE property must be set to true for the Product Information Management for Retailers solution to operate properly.

Performing simple filtering

For simple filtering, the text file specified in the `REQUIRED_ATTRIBUTE_FILE` property must contain a list of the fully qualified attributes in the `Retail_Item` business object that must be present and not NULL or BLANK. It is composed, as follows:

- A fully qualified business object attribute name on a single line.
- On the next single line, another fully qualified attribute name.

Any number of attributes can be included in this way.

The File Missing Attribute logic reads the attribute names from this file and employs an external Java™ class called `RetailUtility` in the Java package `com.ibm.wbi.retail.utils` to determine if the required attributes listed in the file contain data in the `Retail_Item` business object being processed (see the section “Using the `RetailUtility` external Java class” on page 26 for more information). It then adds the names of any required attributes missing data to a `Retail_Item` business object attribute named in the configuration property `CUST_DATA_MISS_ATTR`. This attribute consists of a multiple cardinality array of `Retail_Missing_Attributes` business objects. If there are any names in this missing attribute child business object array, the `ItemValidation` collaboration object passes the `Retail_Item` business object to the `ToMissingData` port to start a process for obtaining the missing data.

The following examples show how the missing attribute list is populated:

- The attribute `customer_data.vendorAddress` is included in the required attribute file. The `Retail_Item` business object that triggers the `ItemValidation` collaboration object does not contain an instance of the `customer_data` child business object attribute. Therefore, the missing attribute code does not include `customer_data.vendorAddress` in the missing attribute list.
- The attribute `customer_data.vendorAddress` is included in the required attribute file. The `Retail_Item` business object that triggers the `ItemValidation` collaboration object does contain an instance of the `customer_data` child business object attribute. The `vendorAddress` attribute is NULL. Therefore, the missing attribute code includes `customer_data.vendorAddress` in the missing attribute list.
- The attribute `item.catalogueItem.catalogueItemChildItemLink[].catalogueItem.tradeItem.tradeItemIdentification.gtin` is included in the required attribute file. The `Retail_Item` business object that triggers the `ItemValidation` collaboration object contains two instances of the `catalogueItemChildItemLink` business object attribute, each of which contains an instance of the `catalogueItem.tradeItem.tradeItemIdentification` child business object. In each instance, the `gtin` attribute is NULL. Therefore, the missing attribute code adds the following entries to the missing attribute list:

```
item.catalogueItem.catalogueItemChildItemLink[0].catalogueItem.tradeItem.  \
tradeItemIdentification.gtin
item.catalogueItem.catalogueItemChildItemLink[1].catalogueItem.tradeItem.  \
tradeItemIdentification.gtin
```

The missing attribute code includes specific references to all required attributes that exist, but are NULL or BLANK.

- The attribute `item.catalogueItem.catalogueItemChildItemLink[].catalogueItem.tradeItem.tradeItemIdentification.gtin` is included in the required attribute file. The `Retail_Item` business object that triggers the `ItemValidation` collaboration object contains no instances of the `catalogueItemChildItemLink` business object

attribute. Therefore, the missing attribute code includes no references to the `item.catalogueItem.catalogueItemChildItemLink[].catalogueItem.tradeItem.tradeItemIdentification.gtin` attribute in the missing attribute list.

Performing complex filtering

For complex filtering, the text file specified in the `REQUIRED_ATTRIBUTE_FILE` property must contain the fully qualified attributes in the `Retail_Item` business object that must be present and not NULL or BLANK, conditionally specified by complex filters.

The first line of the file must contain a separator of `FILTER` or `FIELDS`. The file can contain any number of sections, each section containing `FILTER` and `FIELDS` subsections. “Filterless” `FIELDS` subsections can also be included.

Each `FILTER` subsection filters on one or more business object attributes. A `FILTER` subsection contains the following:

- A fully qualified business object attribute name on a single line.
- On the next single line, a comma-delimited list of valid values for the attribute.
- If filtering on multiple attributes, on the next single line, a value of `AND`.
- On the next single line, another fully qualified attribute name.
- On the next single line, a comma-delimited list of valid values for that attribute.

Any number of attributes can be filtered in this way.

Each `FILTER` subsection must be followed by a `FIELDS` subsection. A `FIELDS` subsection must contain the required attributes for the filter above it, each attribute on a separate single line. A `FIELDS` subsection can be followed by a `FILTER` subsection, another `FIELDS` subsection without a filter, or an optional `END` separator at the end of the file. If a `FIELDS` subsection is specified without a `FILTER` subsection in front of it, the list of attributes in that `FIELDS` subsection is always required.

The collaboration object reads the file until it finds a `FILTER` subsection that is satisfied by the attribute values contained in the business object being processed. The File Missing Attribute logic employs an external Java class called `RetailUtility` in the Java package `com.ibm.wbi.retail.utils` to determine if the required attributes listed in the `FIELDS` subsection of the satisfied filter contain data in the `Retail_Item` business object being processed (see the section “Using the `RetailUtility` external Java class” on page 26 for more information). If any of the attributes in the `FIELDS` subsection are missing from the business object or are present in the business object but missing data, the collaboration object adds their names to a `Retail_Item` business object attribute named in the configuration property `CUST_DATA_MISS_ATTR`. This attribute consists of a multiple cardinality array of `Retail_Missing_Attributes` business objects. If there are any names in this missing attribute child business object array, the `ItemValidation` collaboration object passes the `Retail_Item` business object to the `ToMissingData` port to start a process for obtaining the missing data.

If no file is specified in the `REQUIRED_ATTRIBUTE_FILE` property, or if no attributes are indicated within the file, all items are accepted for further processing by the collaboration object.

An example file that demonstrates complex filtering and its interpretation follow:

```
FILTER
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
```

```

0001111222333
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemUnitDescriptor
CASE,PALLET
FIELDS
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.netWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.grossWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.height
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.width
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.depth
FILTER
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.informationProviderOfTradeItem.informationProvider.gln
0001111222333
AND
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemUnitDescriptor
BASE_UNIT_OR_EACH
FIELDS
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.netWeight
notificationDetail.catalogueItemNotification.catalogueItem.tradeItem. \
tradeItemInformation.tradingPartnerNeutralTradeItemInformation. \
tradeItemMeasurements.grossWeight
END

```

The collaboration object checks if the business object being processed has a gln attribute value of 0001111222333 and a tradeItemUnitDescriptor attribute value of CASE or PALLET. If it doesn't, the collaboration proceeds to the next FILTER. If it does, the collaboration object then checks if the business object being processed contains the attributes netWeight, grossWeight, height, width, and depth, and that these attributes are not NULL or BLANK. If any of the attributes in the FIELDS subsection are missing from the business object or are present in the business object but missing data, the collaboration object adds their names to a Retail_Item business object attribute named in the configuration property CUST_DATA_MISS_ATTR and handles the business object as described above.

The collaboration object then proceeds to the next FILTER subsection. It checks if the business object has a gln value of 0001111222333 and a tradeItemUnitDescriptor attribute value of BASE_UNIT_OR_EACH. If it does, the collaboration object then checks if the business object being processed contains the attributes netWeight and grossWeight, and that these attributes are not NULL or BLANK. If any of the attributes in the FIELDS subsection are missing from the business object or are present in the business object but missing data, the collaboration object again adds their names to the Retail_Item business object attribute named in the configuration property CUST_DATA_MISS_ATTR and handles the business object as described above.

Specifying attribute names

Each child in a hierarchical business object structure described by the attribute name is separated by periods. Any multiple-cardinality child business objects (BusObjArray) in the structure are represented in the element by appending front- and end-bracket symbols ([]) to the name of the child attribute. Single-cardinality child business objects must not contain brackets ([]) in the attribute name. The brackets ([]) characters are required to tell the code that parses through the child business objects in the attribute whether the object is a single business object (BusObj) or a business object array (BusObjArray).

Some example file entries corresponding to Retail_Item business object attributes include the following:

```
customer_data.vendorAddress
item.catalogueItem.catalogueItemChildItemLink[].catalogueItem.tradeItem. \
tradeItemIdentification.gtin
```

Notice that since the catalogueItemChildItemLink business object is a multiple-cardinality business object array, the [] characters are appended to its name in the fully qualified attribute.

Note: If an attribute is part of a hierarchy of parent and child business objects, and any of the attribute's parent business objects does not exist in the Retail_Item, then the missing attribute code does *not* include the attribute in the missing attribute list. Also, the Retail_Item business object must contain an instance of its customer_data child business object for the data check code to successfully check the customer_data child business object attributes.

The ItemValidation collaboration object always converts mixed-case Strings to lowercase characters by using the command toLowerCase() before testing their values against known values, with the following exception:

- Attributes that are used by Java to reference a business object remain in mixed-case characters. An example is an attribute name that is inserted in the multiple cardinality array of Retail_Missing_Attributes business objects in the internals.customer_data_missing_attributes attribute.

Using the RetailUtility external Java class

A collaboration object based on the ItemValidation collaboration template checks that required attributes exist and contain data by using an external Java class called RetailUtility, located in the Java package com.ibm.wbi.retail.utils. It recursively parses each fully qualified attribute string to determine if the value of the attribute is NULL or BLANK.

The class contains four methods:

```
public boolean checkRequiredAttribute(BusObj busObj, Vector attrs, Vector err) \
throws Exception
public boolean checkRequiredAttribute(BusObj busObj, String attr, Vector err) \
throws Exception
public boolean checkComplexFilters(BusObj busObj, Vector complexFilters) \
throws Exception
public boolean checkComplexFilter(BusObj busObj, HashMap complexFilter) \
throws Exception
```

The first method, checkRequiredAttribute, takes as input the Retail_Item business object and two Vectors. The first Vector contains the list of fully qualified attribute names to be checked for NULL or BLANK values. On return, the second Vector

contains the list of missing attributes (those containing NULL or BLANK values). The method returns true if no attribute data is missing; it returns false if any required attribute data is missing.

The second method, also named `checkRequiredAttribute`, operates the same as the first, except it takes as input a single fully qualified attribute name.

The third method, `checkComplexFilters`, takes as input the `Retail_Item` business object and one Vector. The Vector contains HashMaps of complexFilters to be checked. If any of the complex filters evaluate to true, then true is returned. If none of them evaluate to true, then false is returned.

The fourth method, `checkComplexFilter`, operates the same as the third, except that it takes as input a HashMap of a single complex filter.

At runtime, if the collaboration object's missing attribute data check logic is enabled (the value of the `Retail_Item` business object's attribute named in the configuration property `ITEM_COMMAND_ATTRIBUTE` exists in the `REQUIRED_ATTRIBUTE_CMDS` property), and the required attribute Vector contains elements, this class must be added to a directory or to a jar file contained in one of the following:

- On Windows® 2000 systems, the ICS CLASSPATH, which is set up during the `start_server.bat` file prior to starting the ICS.
- On AIX® and Solaris™ systems, the CWCLASSES path, which is set up during the `CWSharedEnv.sh` file prior to starting the ICS.

Access to this external Java class via the ICS CLASSPATH or CWCLASSES path is also required in order to successfully compile the `ItemValidation` collaboration template.

Using the Custom Missing Attribute Logic

The `ItemValidation` collaboration template also contains a subdiagram called Custom Missing Attribute Logic. This subdiagram holds customized code for user-defined methods for identifying the required attributes of the `Retail_Item` business object that are missing data. This code can be executed instead of, or in addition to, the code in the File Missing Attribute Logic subdiagram. Before customization, the Custom Missing Attribute Logic subdiagram contains examples of the following methods to enhance the algorithm for identifying required attributes of the `Retail_Item` business object that are missing data.

Note: These examples do not execute as long as the TEST configuration property is set to false. These examples should not be executed as part of production code.

- Add fully qualified attribute names to the Required Attribute Vector, which is processed later in the algorithm.
- Add fully qualified attribute names to the Missing Attribute Vector, which is processed later in the algorithm.
- Modify the name of the file that is normally set from the `REQUIRED_ATTRIBUTE_FILE` configuration property.
- Create and add instances of `Retail_Missing_Attributes` business objects directly to the attribute named in the `CUST_DATA_MISS_ATTR` configuration property (by default, `internals.customer_data_missing_attributes`).

Using a custom missing data retrieval process to collect data

The custom process used to collect missing data can be another collaboration object, a user interface, a WebSphere MQ Workflow process, or even an email function. Regardless of the method used, it must set each missing attribute value it acquires into the `internals.customer_data_missing_attributes.attributeValue` attribute of the `Retail_Item` business object. In the context of the Product Information Management for Retailers solution, it must then call an `ItemCollector` collaboration object, which assembles the complete `Retail_Item` business object by merging the values of each missing attribute (`internals.customer_data_missing_attributes.attributeValue`) into its proper place in a previously stored, complete `Retail_Item` business object retrieved from the item store. See the section “Merging data into a complete item” on page 34 for more information on how the `ItemCollector` collaboration object merges missing information.

Note: If the custom missing data retrieval process is not asynchronous, or can persist a complete `Retail_Item` business object, an `ItemCollector` collaboration object does not have to be used. The custom process can reuse code in the `ItemCollector` collaboration template to merge the collected data into the `Retail_Item` business object and then pass the merged `Retail_Item` business object to a `Process_Reviewed_Item` collaboration object. The `DELETE_FROM_LOCAL_STORE` property of the `Process_Reviewed_Item` collaboration object would need to be set to `false`. It is recommended that the custom missing data retrieval process *not* be connected to an `ItemValidation` collaboration object because of error flows. If an error occurs, the `Process_Reviewed_Item` collaboration object calls a `UCCnetMessageSend` collaboration object to send a response back to the original sender (`UCCnet` in the context of the Product Information Management for Retailers solution).

Adding customized code to the `ItemValidation` collaboration template

The `ItemValidation` collaboration template contains a subdiagram called `Message Type Processing` that can be customized to contain any desired logic. After including code in this subdiagram, enable it to execute for specific commands attached to a `Retail_Item` by specifying those commands in the `ItemValidation` collaboration object’s `MESSAGE_TYPE_PROCESSING_CMDS` configuration property.

The subdiagram logic examines the value for the `Retail_Item` business object’s attribute named in the `ItemValidation` collaboration object’s `ITEM_COMMAND_ATTRIBUTE` configuration property against the values specified in its `MESSAGE_TYPE_PROCESSING_CMDS` configuration property. If the value for the attribute in the `ITEM_COMMAND_ATTRIBUTE` property is specified in the `MESSAGE_TYPE_PROCESSING_CMDS` property, the `ItemValidation` collaboration object executes the customized code in the `Message Type Processing` subdiagram.

Note: The code contained in the `Message Type Processing` subdiagram does *not* replace the code in the `File Missing Attribute Logic` or `Custom Missing Attribute Logic` subdiagrams.

For detailed information on the collaboration object and business object mentioned in this section, see the following:

- `ItemValidation` collaboration template

- Retail_Item business object

Processing business objects with cascaded GLNs

The ItemDispatcher collaboration template is used to control how approval requests generated by an ItemValidation collaboration object are sent to a WebSphere MQ Workflow process. If a Retail_Item business object that requires approval contains cascaded GLNs, the ItemDispatcher collaboration object breaks up the request for approval into multiple smaller messages, each containing one or more of the individual GLNs. For example, assume that you are working with a single object that has a cascaded GLN containing six individual GLNs. GLNs 1, 2, and 3 are the responsibility of approver A. GLNs 4 and 5 are the responsibility of approver B. GLN 6 can be ignored. You can use the ItemDispatcher collaboration to create one copy of the Retail_Item business object that contains only GLNs 1, 2, and 3, and a second copy that contains only GLNs 4 and 5. GLN 6 is disregarded. These two copies can then be directed to the correct approvers through WebSphere MQ Workflow.

How the collaboration object handles any particular GLN is determined by a user-defined configuration file. See the section “Using the GLN Cascade Grouping File” for more information on the structure of this file. Items without cascaded GLNs are passed through the collaboration object without processing.

Note: You can implement the Product Information Management for Retailers solution without using an ItemDispatcher collaboration object. However, you must use the ItemDispatcher collaboration object if cascaded GLNs will be received.

Using the GLN Cascade Grouping File

The GLN Cascade Grouping File is a user-created text file, which provides information the ItemDispatcher collaboration object needs to determine how to group the GLNs for a given item in messages sent to WebSphere MQ Workflow for approval. It can contain any number of sections, each section containing FILTER and GLN_GROUPS subsections. A final optional “filterless” subsection, containing only a GLN_GROUPS subsection, can also be included.

Each FILTER subsection filters on one or more Retail_Item business object attributes. A FILTER subsection contains the following:

- A fully qualified Retail_Item attribute on a single line.
- On the next single line, a comma-delimited list of valid values for the attribute.
- If filtering on multiple attributes, on the next single line, a value of AND.
- On the next single line, another fully qualified Retail_Item attribute.
- On the next single line, a comma-delimited list of valid values for that attribute.

Any number of attributes can be filtered in this way.

Each FILTER subsection must be followed by a GLN_GROUPS subsection. A GLN_GROUPS subsection must contain at least one comma-delimited list of cascaded GLN values grouped together on a single line. Additional groups of cascaded GLNs can be defined simply by adding more lines of comma-delimited GLN lists to this subsection. A GLN_GROUPS subsection can be followed by a FILTER subsection, another GLN_GROUPS subsection without a filter, or an optional END separator at the end of the file.

The ItemDispatcher collaboration object reads through the GLN Cascade Grouping File specified in the GLN_CASCADE_GROUPING_FILE configuration property until it finds a FILTER subsection that is satisfied by the attribute values contained in the Retail_Item being processed. The collaboration object then groups the GLNs according to the GLN_GROUPS subsection associated with the satisfied FILTER subsection. If an item satisfies the conditions of multiple filters, only the first filter in the file is ever considered.

An example file and its interpretation follow:

```

FILTER
internals.fromGln
0001000000001,0001000000002
AND
item.catalogueItem.tradeItem.tradeItemInformation.classificationCategoryCode. \
additionalClassification[].additionalClassificationCategoryCode
0001.001.001,0001.001.002,0001.001.003
GLN_GROUPS
0005000000001,0005000000002,0005000000003
0005000000004,0005000000005,0005000000006
FILTER
internals.fromGln
0001000000005
GLN_GROUPS
0005000000001,0005000000002
0005000000003,0005000000004
0005000000005,0005000000006
GLN_GROUPS
0005000000001
0005000000002
0005000000003
0005000000004
0005000000005
0005000000006

```

If the Retail_Item business object being processed has a fromGln attribute value of 0001000000001 or 0001000000002 and one of the following additionalClassificationCategoryCode attribute values:

- 0001.001.001
- 0001.001.002
- 0001.001.003

then the GLNs are grouped as follows:

Group 1

Contains GLNs 0005000000001, 0005000000002, and 0005000000003, if they exist.

Group 2

Contains GLNs 0005000000004, 0005000000005, and 0005000000006, if they exist.

Any GLNs that are not in the GLN_GROUPS subsection are disregarded. Each group of GLNs is sent separately to WebSphere MQ Workflow for approval, so a maximum of two messages are sent.

If the first filter conditions were not satisfied, the collaboration object now checks to see if the Retail_Item business object has a fromGln value of 0001000000005. If it does, then the GLNs are grouped as follows:

Group 1

Contains GLNs 0005000000001 and 0005000000002, if they exist.

Group 2

Contains GLNs 0005000000003 and 0005000000004, if they exist.

Group 3

Contains GLNs 0005000000005 and 0005000000006, if they exist.

Any GLNs that are not in the GLN_GROUPS subsection are disregarded. Each group of GLNs is sent separately to WebSphere MQ Workflow for approval, so a maximum of three messages are sent.

If none of the preceding filters has been satisfied, the filterless GLN_GROUPS subsection is assumed to apply. Each of the six listed GLNs will be sent to WebSphere MQ Workflow in its own message. Unlisted GLNs are disregarded, so a maximum of six messages are sent.

If no FILTER subsection is satisfied and there is no filterless GLN_GROUPS subsection, then the behavior is determined by the value of the GLN_CASCADE_GROUPING_DEFAULT property.

Using a business process to review and approve an item

The business review/approval process is a custom application used by the demand-side trading partner to review an item to determine whether to accept it. In the context of the Product Information Management for Retailers solution, a WebSphere MQ Workflow process represents it. The process can also be a general user interface or an automated process, such as another collaboration object, that examines specific item attributes and makes a decision based on that data. The following sections describe operations related to using WebSphere MQ Workflow as the business review/approval process:

- “Printing an item before it is sent through the business review/approval process”
- “Mapping an item to the business review/approval process” on page 32
- “Using WebSphere MQ Workflow containers” on page 32
- “Returning data from the business review/approval process to an ItemCollector collaboration object” on page 34

For more information on WebSphere MQ Workflow, see IBM WebSphere MQ Workflow documentation. For detailed information on the collaboration objects and business object mentioned in these sections, see the following:

- ItemCollector collaboration template
- ItemStore collaboration template
- UCCnetMessageReceive collaboration template
- UCCnetMessageSend collaboration template
- Retail_Item business object

Printing an item before it is sent through the business review/approval process

To produce a hard copy of an item that arrives from UCCnet, create a second UCCnetMessageReceive collaboration object. Connect its *FromAS2* port to an AS2 channel connector, its *ToRetail_Processing* port to a JTextConnector, and its *ToRetail_Response*, *ToMessage_Store*, and *ToIdentifier_Store* ports to the PortConnector.

This second UCCnetMessageReceive collaboration object sends the item to the JTextConnector, which writes it to a file system. From the file system, the file can be printed by using any local print function.

Mapping an item to the business review/approval process

In the context of the Product Information Management for Retailers solution, the business review/approval process begins when an ItemDispatcher collaboration object passes the Retail_Item business object to the WebSphereMQWorkflowConnector over its *To* port. The WebSphereMQWorkflowConnector maps the Retail_Item business object into a WebSphere MQ Workflow container by passing it through the Retail_Item_to_MQWF_Retail_Item map. Examine this map in the Map Designer tool to view how the Retail_Item business object is mapped into the MQWF_Retail_Item application specific business object. At this point, the flow becomes asynchronous.

Note: The case of a business object's String attributes is normally not important. Product Information Management for Retailers solution collaboration objects always convert mixed-case Strings to lowercase characters by using the command `toLowerCase()` before testing their values against known values, with the following exception:

- Attributes that are used by Java to reference a business object remain in mixed-case characters. An example is an attribute name that is inserted in the multiple cardinality array of Retail_Missing_Attributes business objects in the `internals.customer_data_missing_attributes` attribute.

The Retail_Item_to_MQWF_Retail_Item map also invokes a specific WebSphere MQ Workflow process definition based on the command identified in the `internals.item_command` attribute of the Retail_Item business object. There are three WebSphere MQ Workflow process definitions in the Product Information Management for Retailers solution, which are located in the `Retail.fdl` file:

- Retail_ItemCreate
- Retail_ItemUpdate
- Retail_ItemDelete

These process definitions are determined by the relationship definitions contained in the `CMDTOWPN` file for the mapping of the `internals.item_command` attribute to `ProcessTemplateName`. They allow users to select a container attribute to be modified.

A customized solution might require a different set of WebSphere MQ Workflow processes. In this case, modify the process definitions by loading them into WebSphere MQ Workflow Buildtime. Alternatively, provide your own process definitions to match the item commands that are specified in the Retail_Item_to_MQWF_Retail_Item map. Customize the map's custom move code to enable different WebSphere MQ Workflow process definition routings. The custom move code includes examples of how to generate definition names by either using a relationship or by hard coding them. The name of the WebSphere MQ Queue Manager might have to be changed to match a particular implementation.

Using WebSphere MQ Workflow containers

A WebSphere MQ Workflow container must contain the item data that is needed to make the business process decision. It must also contain certain internal attributes

from the Retail_Item business object so they can be returned to succeeding collaboration objects that require them. The following list includes those attributes that must be mapped into the WebSphere MQ Workflow container:

internals.item_status

This attribute holds the item status value when there are no cascaded GLNs in the `internals.cascadedGlns` child business object. The approver(s) responsible for approving items must set the value of this attribute to Approved, Rejected, Accepted, or Review through a customized user interface with WebSphere MQ Workflow, so that the succeeding collaboration objects can determine the processing path for the item. If the status remains Review, the item is routed back to the ItemValidation collaboration object and then back into business review/approval processing.

internals.cascadedGlns

This attribute is an N-cardinality child business object that holds the cascaded GLNs and the status for each GLN. When cascaded GLNs are present in the `internals.Glns` child business object, the approver(s) responsible for approving items must set the value of the `internals.cascadedGlns.item_status` attribute for each cascaded GLN to Approved, Rejected, Accepted, or Review through a customized user interface with WebSphere MQ Workflow, so that the succeeding collaboration objects can determine the processing path for the item. If the status remains Review, the item is routed back to the ItemValidation collaboration object and eventually back into business review/approval processing.

Note: If you are using DTD XML definitions, all GLNs, whether cascaded or not, are carried to the WebSphere MQ Workflow in this attribute.

internals.item_command

This attribute holds the item command value.

internals.correlationID

This attribute holds the identifying key used to retrieve the item from the local item store. Even if a local item store is not used, the value for this attribute must be returned since it is used by the UCCnetMessageReceive and UCCnetMessageSend collaboration objects.

internals.fromGln

This attribute holds the GLN of the supplier publishing the Retail_Item.

internals.toGln

This attribute holds the top-level retailer GLN to which the item is being published.

Other useful data to map into the WebSphere MQ Workflow container include the following attributes:

internals.date_processed

This attribute holds the date that the item is processed by the approver(s).

internals.time_processed

This attribute holds the time that the item is processed by the approver(s).

internals.responder_name

This attribute holds the name of the approver or a list of approvers.

If the WebSphere MQ Workflow business process needs additional data from the Retail_Item business object that is not mapped into the container, a customized program that reads the local item store must be developed. For example, a collaboration template could be developed that invokes an ItemStore collaboration object to read the store. The custom collaboration object could pass the Retail_Item business object `internals.correlationID` attribute to the ItemStore collaboration object as the key to the item in the store. See the section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 for more information about retrieving item information from the item store.

Returning data from the business review/approval process to an ItemCollector collaboration object

When the business review/approval process completes, the WebSphere MQ Workflow process sends the container with the updated status to the WebSphereMQWorkflowConnector. At this point, the flow becomes synchronous again. The WebSphereMQWorkflowConnector maps the WebSphere MQ Workflow container back into a partial Retail_Item business object by passing it through the MQWF_Retail_Item_to_Retail_Item map and then invokes an ItemCollector collaboration object. Values for the following attributes must be mapped back into the Retail_Item business object from the WebSphere MQ Workflow container:

- `internals.item_status`
- `internals.item_command`
- `internals.correlationID`
- `internals.cascadedGlns[].gln`
- `internals.cascadedGlns[].item_status`

The data mapped back into the Retail_Item from the WebSphere MQ Workflow container must also be matched with configuration property settings for the ItemCollector collaboration object, which determine which updated attribute values to copy back into a complete Retail_Item business object. See the section “Merging static data by using the X_COPY ATTRIBUTE configuration properties” on page 37 for details.

Merging data into a complete item

The Retail_Item business object passed to an ItemCollector collaboration object might not be complete. The updated data contained in this incomplete item must be merged into the complete Retail_Item business object that was saved in the local item store before the item was sent to business review/approval or missing data retrieval processing. The ItemCollector collaboration object retrieves the copy of the Retail_Item that was stored by the ItemValidation collaboration object from the local item store. It retrieves the item by sending the business object and a Retrieve verb to its *local_store* port to be passed to an ItemStore collaboration object, which actually retrieves the item and returns it to the ItemCollector collaboration object. See the section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 for more information on retrieving items from the item store.

The collaboration object checks if the original Retail_Item retrieved from the item store database contains cascaded GLNs. It then directs processing, as follows:

- If the Retail_Item does not contain cascaded GLNs, the collaboration object merges the new data received from the WebSphere MQ Workflow process into the Retail_Item business object that was read from the item store. See the sections “Merging static data by using the X_COPY ATTRIBUTE configuration

properties” on page 37 and “Merging missing data by using the MISSING_DATA_CHILD_ATTRIBUTE configuration property” on page 37 for information on how merging is accomplished.

Note: If the business object had not had all required data present, causing the ItemValidation collaboration object to invoke a custom missing data retrieval process, it is at this point that this retrieval process would call an instance of an ItemCollector collaboration object to merge the retrieved data into the complete Retail_Item business object.

The merged Retail_Item business object is routed to a Process_Reviewed_Item collaboration object via the ItemCollector collaboration object’s *To* port.

- If the Retail_Item does contain cascaded GLNs, the collaboration object does the following:
 1. It loops through the GLNs, processing each one, as follows:
 - a. It retrieves the copy of the Retail_Item that was stored by the ItemDispatcher collaboration object from the item store database by sending the business object and a Retrieve verb to its *local_store* port to be passed to an ItemStore collaboration object. The key used to access the item is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value for the GLN being processed. The copy is returned to the ItemCollector collaboration object on its *local_store* port.
 - b. The ItemCollector collaboration object merges the new data received from the WebSphere MQ Workflow process into the Retail_Item business object retrieved from the item store. See the sections “Merging static data by using the X_COPY ATTRIBUTE configuration properties” on page 37 and “Merging missing data by using the MISSING_DATA_CHILD_ATTRIBUTE configuration property” on page 37 for information on how merging is accomplished.

Note: If the business object had not had all required data present, causing the ItemValidation collaboration object to invoke a custom missing data retrieval process, it is at this point that this retrieval process would call an instance of an ItemCollector collaboration object to merge the retrieved data into the complete Retail_Item business object.

- c. The ItemCollector collaboration object then handles the business object according to its status value, as follows:
 - If the status value of the `internals.cascadedGlns.item_status` attribute is either Approved or Rejected, the merged Retail_Item business object is routed to a Process_Reviewed_Item collaboration object via the ItemCollector collaboration object’s *To* port.
 - If the status value of the `internals.cascadedGlns.item_status` attribute is Review or Accepted, the `internals.cascadedGlns` child business object is added to an array, as follows:
 - If the status is Review, it is added to an array of review cascaded GLNs.
 - If the status is Accepted, it is added to an array of accepted cascaded GLNs.

Note: If the cascaded GLN is not the first GLN in the respective array, the ItemCollector collaboration object does the following:

- 1) Deletes the entry from the message store database that was stored by the ItemDispatcher collaboration object by converting the Retail_Item to a UCCnetGBO_storable business object and sending it with a Delete verb to its *message_store* port to be passed to an MessageStore collaboration object, which performs the deletion. The key used to access the item is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value of the newly created business object.
 - 2) Deletes the copy of the Retail_Item that was stored by the ItemDispatcher collaboration object from the item store database by sending the business object and a Delete verb to its *local_store* port to be passed to an ItemStore collaboration object, which performs the deletion. The key used to access the item is the concatenation of the `internals.correlationID` attribute value of the original triggering business object and the `internals.cascadedGlns.gln` attribute value of the newly created business object.
2. After all of the cascaded GLNs are processed, all the GLNs with a status of Review are sent as a group by copying the array of review cascaded GLNs into the `internals.cascadedGlns` attribute, to a Process_Reviewed_Item collaboration object via the ItemCollector collaboration object's *To* port. Similarly, all GLNs with a status of Accepted are sent as a group by copying the array of accepted cascaded GLNs into the `internals.cascadedGlns` attribute, to a Process_Reviewed_Item collaboration object via the ItemCollector collaboration object's *To* port.
 3. The ItemCollector collaboration object calculates how many total cascaded GLNs have completed processing (i.e., they have Approved, Rejected, or Error status). If all of the cascaded GLNs from the original message that was sent to the ItemDispatcher collaboration object have completed processing, it does the following:
 - a. Deletes the entry from the message store database that was stored by the UCCnetMessageReceive collaboration object by converting the Retail_Item to a UCCnetGBO_storable business object and sending it with a Delete verb to its *message_store* port to be passed to an MessageStore collaboration object, which performs the deletion. The key used to access the item is the `internals.correlationID` attribute value of the original triggering business object.
 - b. Deletes the copy of the Retail_Item that was stored by the ItemValidation collaboration object from the item store database by sending the business object and a Delete verb to its *local_store* port to be passed to an ItemStore collaboration object, which performs the deletion. The key used to access the item is the `internals.correlationID` attribute value of the original triggering business object.

The following sections describe the merging operations in greater detail:

- "Merging static data by using the X_COPY ATTRIBUTE configuration properties" on page 37
- "Merging missing data by using the MISSING_DATA_CHILD_ATTRIBUTE configuration property" on page 37.

For detailed information on the collaboration objects and business object mentioned in this and the following sections, see the following:

- ItemCollector collaboration template
- ItemDispatcher collaboration template
- ItemStore collaboration template
- Retail_Item business object

Merging static data by using the X_COPY ATTRIBUTE configuration properties

An ItemCollector collaboration template can be configured with properties that indicate which attributes to copy from a triggering business object received on the *From* port into the complete item previously stored in the item store. These configuration properties are of the form X_COPY_ATTRIBUTE (where X is an integer, such as in 1_COPY_ATTRIBUTE). Four properties with default values are supplied with the collaboration template:

- 1_COPY_ATTRIBUTE
- 2_COPY_ATTRIBUTE
- 3_COPY_ATTRIBUTE
- 4_COPY_ATTRIBUTE

Any number of configuration properties of the form X_COPY_ATTRIBUTE can be created to copy any number of attributes.

For each of the attributes identified in these properties, the collaboration object first verifies that the attribute exists in the Retail_Item business object and, if it does, copies the updated attribute value from the Retail_Item business object received on the *From* port to the Retail_Item retrieved from the local item store. If the attribute does not exist in the Retail_Item business object, the collaboration object generates an informational message in the log and continues processing.

When using an unmodified Retail_Item business object, at a minimum, set 1_COPY_ATTRIBUTE equal to the value `internals.item_status`, so that the status of the item is copied. Remove any properties of the form X_COPY_ATTRIBUTE that are not used. A blank for the property value, or an invalid attribute name, cause a warning message to be logged by the collaboration object.

Merging missing data by using the MISSING_DATA_CHILD_ATTRIBUTE configuration property

Within a triggering Retail_Item business object, an ItemCollector collaboration object checks if the attribute specified in its property MISSING_DATA_CHILD_ATTRIBUTE (by default, the `internals.customer_data_missing_attributes` child business object) contains a list of missing attributes.

Note: It is assumed that previous processing, such as that performed by an ItemValidation collaboration object, identified which attribute names were required and missing from the item and that previous processing (such as a custom collaboration object) collected the values for those missing attributes and set the values in the Retail_Missing_Attributes child business object of the Retail_Item business object. See the sections “Validating an item by requiring data for specific attributes” on page 22 and “Using a custom missing data retrieval process to collect data” on page 28 for more information.

Each `Retail_Missing_Attributes` business object contains two attributes. The first attribute contains the name of a `Retail_Item` attribute that was determined during previous processing to be required and missing its data. The second attribute contains the value of the `Retail_Item` attribute that was obtained through a custom missing data retrieval process prior to the `ItemCollector` collaboration object being invoked.

The `ItemCollector` collaboration object needs to reference each of the two elements of the `Retail_Missing_Attributes` child business object. This is accomplished by using two configuration properties, `MISSING_DATA_NAME_ATTRIBUTE` and `MISSING_DATA_VALUE_ATTRIBUTE`:

MISSING_DATA_NAME_ATTRIBUTE

This property holds the attribute name as defined in the `Retail_Missing_Attributes` child business object. By default, this value is `attribute_name`. If this default value is changed, the change must be reflected in the value of the `MISSING_DATA_NAME_ATTRIBUTE` as well.

MISSING_DATA_VALUE_ATTRIBUTE

This property holds the attribute value as defined in the `Retail_Missing_Attributes` child business object. By default, this value is `attributeValue`. If this default value is changed, the change must be reflected in the value of the `MISSING_DATA_NAME_ATTRIBUTE` as well.

For each `Retail_Missing_Attributes` instance in the missing attribute array, the collaboration object first verifies that the attribute is a valid `Retail_Item` attribute. If so, the attribute's value is copied from the `Retail_Missing_Attributes` instance to its correct position in the `Retail_Item` business object obtained from the local item store. If the attribute is not valid for the `Retail_Item`, the collaboration object generates an informational message in the log and continues processing.

Processing an item after the business review/approval process completes

After the `ItemCollector` collaboration object merges data into a complete `Retail_Item` business object, it passes this `Retail_Item` to a `Process_Reviewed_Item` collaboration object. The `Process_Reviewed_Item` collaboration object receives this object on its *From* port. It then performs several tasks, as follows:

1. If the collaboration object's `DELETE_FROM_LOCAL_STORE` configuration property is set to `true`, it calls an `ItemStore` collaboration object to delete the item from the item store. See the section "Persisting, retrieving, updating, or deleting an item to or from a local item store" on page 45 for more information.
2. It examines the status of the item and directs processing, as follows:
 - If the item status is `Approved`, the collaboration object routes the `Retail_Item` business object to the `JTextConnector` via its *Sync* port. The `JTextConnector` writes the `Retail_Item` business object to a file system. If this is successful, the item status is changed to `Synchronised`; if not, the item status is changed to `Error`. The item is also sent to the *respond_to* port to be passed to a `UCCnetMessageSend` collaboration object. Alternatively, the `Retail_Item` can be sent to a single application other than the `JTextConnector` or to multiple back-end applications. For more information, see the sections "Synchronizing an item to a back-end application other than a file system" on page 39 and "Synchronizing an item to multiple back-end applications" on page 40, respectively.

- If the item status is Rejected, the collaboration object routes the Retail_Item to the JTextConnector via its *Sync* port. The JTextConnector writes the Retail_Item business object to a file system. If this is not successful, the item status is changed to Error. The item is then sent to the *respond_to* port to be passed to a UCCnetMessageSend collaboration object.
- If the item status is Accepted or Review, the collaboration object routes the Retail_Item business object back to a ItemValidation collaboration object via its *reprocess* port. The Process_Reviewed_Item collaboration object then waits for a response. When the item is returned on the *reprocess* port from the ItemValidation collaboration object, the Process_Reviewed_Item collaboration object processes it, as follows:
 - If the status of the returned item is Review or reprocess, it indicates that the item was successfully processed by the ItemValidation collaboration object and sent to the WebSphere MQ Workflow business review/approval process by the ItemDispatcher collaboration object. If the original status for the item when it was received on the *From* port was Accepted, the item with Accepted status is sent to the *respond_to* port to be passed to a UCCnetMessageSend collaboration object. Otherwise, the item is not sent to the *respond_to* port. At this point, processing performed by the Process_Reviewed_Item collaboration object ends. When the WebSphere MQ Workflow process triggered by the ItemDispatcher collaboration object completes, the Retail_Item will again be sent to a Process_Reviewed_Item collaboration object's *From* port.
 - If the status of the returned item is Rejected or Error, the collaboration object sends the Retail_Item to the *respond_to* port to be passed to a UCCnetMessageSend collaboration object.
- 3. Optionally, the collaboration object calls an ItemStore collaboration object to rewrite the Retail_Item to the local item store for back-up or future validation purposes. See the section "Persisting, retrieving, updating, or deleting an item to or from a local item store" on page 45 for more information.

For detailed information about the collaboration objects and business object mentioned in this and the following sections, see the following:

- ItemCollector collaboration template
- ItemDispatcher collaboration template
- ItemStore collaboration template
- ItemValidation collaboration template
- Process_Reviewed_Item collaboration template
- UCCnetMessageSend collaboration template
- Retail_Item business object

Synchronizing an item to a back-end application other than a file system

In the context of the Product Information Management for Retailers solution, a Process_Reviewed_Item collaboration object's *Sync* port is connected to the JTextConnector, which writes the item to a file system. By setting the JTextConnector's *BOPrefix* property, the filename can be prefixed to match the requirements of any back-end application. Also, the *Sync* port can be attached to any other connector that supports a different back-end application, such as a legacy system. IBM WebSphere technology adapters can provide interfaces to different types of back-end systems such as databases or queue interfaces.

Synchronizing an item to multiple back-end applications

Synchronize a Retail_Item business object to more than one back-end application by doing one of the following:

- Modify the Process_Reviewed_Item collaboration template to include several *Sync* ports or write another custom collaboration template and attach a collaboration object based on it to a Process_Reviewed_Item collaboration object's *Sync* port. A collaboration object based on this custom collaboration template can pass the Retail_Item business object to multiple output ports (i.e., each back-end application's connector).
- Connect a Process_Reviewed_Item collaboration object's *Sync* port to a collaboration object based on the ItemSync collaboration template. The Retail_Item business object must be added to the list of supported business objects for a collaboration object based on this template. The advantage of using an ItemSync collaboration object is that it provides the base capabilities to do the following:
 - Convert a Create verb to an Update verb if the item currently exists in the back-end application.
 - Convert an Update verb to a Create verb if the item does not exist in the back-end application (a Process_Reviewed_Item collaboration object provides only the Create verb).
 - Filter the item based on its contents and not synchronize it to the back-end application.

Sending responses to UCCnet

In the Product Information Management for Retailers solution, a Process_Reviewed_Item collaboration object is connected through its *respond_to* port to a UCCnetMessageSend collaboration object, which builds and sends appropriate responses to UCCnet. Response messages must be returned to UCCnet for each NEW_ITEM, DATA_CHANGE, or NEW_ITEM with reload request. Process_Reviewed_Item collaboration objects pass Retail_Item business objects to UCCnetMessageSend collaboration objects when they have been assigned Synchronised, Rejected, Accepted, or Error status. UCCnetMessageReceive collaboration objects may also pass Retail_Item business objects with Review or Error status to UCCnetMessageSend collaboration objects for the purpose of sending a response to UCCnet.

When a triggering Retail_Item business object is received, a UCCnetMessageSend collaboration object first retrieves the layer information from the local message store by converting the Retail_Item to a UCCnetGBO_storable business object and passing it to its ToMessage_Store port. This port is bound to the appropriate persistence mechanism, such as a MessageStore collaboration object, which performs the retrieval.

It then extracts the Catalogue Item Notification command and handles the object according to its original command type, its status value, and the values of certain properties, as follows:

- If the original command to the demand-side trading partner was a DE_LIST, WITHDRAW, or CORRECTION, the collaboration object stops processing.
- If the original command to the demand-side trading partner was a NEW_ITEM, DATA_CHANGE, or NEW_ITEM with reload, the UCCnetMessageSend collaboration object does the following:
 1. Checks the value of the `internals.item_status` attribute of the Retail_Item:

- If the status of the Retail_Item is Synchronised, the collaboration object builds a Catalogue Item Confirmation message with a state of Synchronised.
- If the status is Rejected and the collaboration object's SEND_REJECT property is set to true, it builds a Catalogue Item Confirmation message with a state of Rejected.
- If the status is Review and the collaboration object's SEND_REVIEW property is set to true, it builds a Catalogue Item Confirmation message with a state of Review.

Note: A UCCnetMessageReceive collaboration object can initiate sending a Catalogue Item Confirmation message with a state of Review when an item is first received and sent to the business review/approval process. Since this process can be long-running, it might be necessary to send the Catalogue Item Confirmation message with a state of Review back to originator of the request. Normally, UCCnet does not want Catalogue Item Confirmation messages with a state of Review to be sent, so it is recommended that the SEND_REVIEW property be set to false.

- If the status is Accepted and the collaboration object's SEND_ACCEPT property is set to true, it builds a Catalogue Item Confirmation message with a state of Accepted.
2. Composes a UCCnetGBO_envelope business object by first converting the Retail_Item and UCCnetGBO_storable business objects into a UCCnetGBO_RI_S business object and then passing this object through the map specified in its TOAS2_RESPONSE_MAP configuration property. The resulting UCCnetGBO_envelope business object contains all of the information needed for the UCCnet Catalogue Item Confirmation message.
 3. Passes the UCCnetGBO_envelope business object to the AS2 channel connector on its *ToAS2_Response* port. In the connector controller portion of the connector, it is converted to an application specific business object of the form UCCnetXXX_envelope (if iSoft connectivity is used), UCCnetTPIXXX_envelope (if TPI connectivity is used), or UCCnetJMSXXX_envelope (if WebSphere Business Connect is used with JMS). The UCCnetGBO_envelope business object is converted by passing through either the UCCnetGBO_envelope_to_UCCnetXXX_envelope map (if iSoft connectivity is used), the UCCnetGBO_envelope_to_UCCnetTPIXXX_envelope map (if TPI connectivity is used), or the UCCnetGBO_envelope_to_UCCnetJMSXXX_envelope map (if WebSphere Business Connect is used with JMS). The DTD forms of these maps convert the outgoing UCCnetGBO_envelope GBO containing an XSD-based Catalogue Item Confirmation command into an ASBO containing a DTD-based Authorization command.

Note: The variable XXX specifies the XML definition type used (DTD or XSD).

4. Deletes the entry from the message store by converting the business object to a UCCnetGBO_storable business object and passing this object to its ToMessage_Store port. This port is bound to the appropriate persistence mechanism, such as a MessageStore collaboration object, which performs the actual deletion.
- Regardless of the original command type, if the status of the item is Error, the UCCnetMessageSend collaboration object does the following:

- If the value of its `FILTER_DUPLICATE` configuration property is true, it maintains the identifier store by converting the business object to a `UCCnetGBO_identifier` business object and calling an `IdentifierStore` collaboration object to remove the entry from the identifier store. The key attributes for DTD processing are `gtin`, `version`, and `topic`. The key attributes for XSD processing are `gtin`, `topic`, `dataRecipientGLN`, `dataSourceGLN`, `targetMarket`, and `uniqueCreatorID`. See the sections “Filtering to eliminate processing of duplicate items” on page 20 and “Persisting or deleting an item to or from a local identifier store” on page 43 for more information.
- It calls a `MessageStore` collaboration object to delete the entry from the message store. See the section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 for more information.
- It stops further processing of the Catalogue Item Confirmation command so that no response is returned to UCCnet.

For detailed information about the collaboration objects and business objects mentioned in this section, see the following:

- `IdentifierStore` collaboration template
- `MessageStore` collaboration template
- `Process_Reviewed_Item` collaboration template
- `UCCnetMessageReceive` collaboration template
- `UCCnetMessageSend` collaboration template
- `Retail_Item` business object
- `UCCnetDTD_envelope` business object
- `UCCnetGBO_envelope` business object
- `UCCnetGBO_RI_S` business object
- `UCCnetTPIDTD_envelope` business object
- `UCCnetTPIXSD_envelope` business object
- `UCCnetXSD_envelope` business object
- `UCCnetJMSDTD_envelope` business object
- `UCCnetJMSXSD_envelope` business object
-

Using identifier, message, and item stores

Three local stores, or database tables, are used in the Product Information Management for Retailers solution. The stage of item processing determines when these stores are accessed, by which collaboration objects, and which actions (Create, Retrieve, Delete, Update) are performed on them.

Identifier store

The identifier store stores unique identifying information for each item, which is used by a `UCCnetMessageReceive` collaboration object to prevent duplicate Catalogue Item Notifications with `NEW_ITEM`, `DATA_CHANGE`, `WITHDRAW`, or `DE_LIST` commands from being processed. This collaboration object triggers an `IdentifierStore` collaboration object, which performs the actual Create action on the store. When error conditions are encountered, the `UCCnetMessageSend` collaboration object can also trigger an `IdentifierStore` collaboration object to delete the item from the identifier store. See the section “Persisting or deleting an item to

or from a local identifier store” for more information on how these collaboration objects interact with each other and with the identifier store.

Message store

The message store holds item information that is used by UCCnetMessageReceive, UCCnetMessageSend, ItemCollector, and ItemDispatcher collaboration objects to build appropriate messages to be sent back to UCCnet. These collaboration objects trigger a MessageStore collaboration object, which performs the actual Create, Retrieve, and Delete actions on the store. See the section “Persisting, retrieving, or deleting an item to or from a local message store” on page 44 for more information on how these collaboration objects interact with each other and with the message store.

Item store

The item store saves a complete version of the Retail_Item business object. A complete version of the item might have to be stored and later merged with updated information for some of its attributes for the following reasons:

- Processes to be performed on the item are long-running, disjoint, and asynchronous, or cannot persist the entire Retail_Item business object.
- A complete version of the item is needed for validation or back-up.
- Complete information might be needed by business review /approval processes.

ItemValidation, ItemCollector, ItemDispatcher, and Process_Reviewed_Item collaboration objects trigger an ItemStore collaboration object, which performs the actual Create, Retrieve, Update, and Delete actions on the store. See the section “Persisting, retrieving, updating, or deleting an item to or from a local item store” on page 45 for more information on how these collaboration objects interact with each other and with the item store.

Persisting or deleting an item to or from a local identifier store

Various collaboration objects used in the Product Information Management for Retailers solution can initiate the storage or deletion of a business object to or from an identifier store, as follows:

- For a Catalogue Item Notification with a NEW_ITEM, DATA_CHANGE, WITHDRAW, or DE_LIST command, a UCCnetMessageReceive collaboration object initiates persisting the UCCnetGBO_envelope business object by sending it with a Create verb to its *ToIdentifier_Store* port.
- When error conditions are encountered, a UCCnetMessageSend collaboration object initiates deleting the UCCnetGBO_envelope business object by sending it with a Delete verb to its *ToIdentifier_Store* port.

In either case, before sending the UCCnetGBO_envelope business object to the port, the collaboration object first converts it to a UCCnetGBO_identifier business object by passing it through the map specified in its TOIDENTIFIER_STORE_MAP property. The *ToIdentifier_Store* port can then be bound to a persistence mechanism. In the context of the Product Information Management for Retailers solution, this mechanism is an IdentifierStore collaboration object.

The IdentifierStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML (which are detailed per command in IdentifierStore

collaboration template), stores or deletes the identifier. The key to this record in the identifier store is made up of the attributes `gtin`, `version`, and `topic` of the `UCCnetGBO_identifier` business object for DTD processing, and the attributes `gtin`, `topic`, `dataRecipientGLN`, `dataSourceGLN`, `targetMarket`, and `uniqueCreatorID` of the `UCCnetGBO_identifier` business object for XSD processing. The `UCCnetMessageReceive` collaboration object can also check if a duplicate item already exists in the identifier store to prevent processing of duplicate items. See the section “Filtering to eliminate processing of duplicate items” on page 20 for more information on this feature.

For detailed information about components used in persisting or deleting items to or from an identifier store, see the following:

- `IdentifierStore` collaboration template
- `UCCnetMessageReceive` collaboration template
- `UCCnetMessageSend` collaboration template
- `UCCnetGBO_envelope` business object
- `UCCnetGBO_identifier` business object

Persisting, retrieving, or deleting an item to or from a local message store

Various collaboration objects used in the Product Information Management for Retailers solution can initiate the storage, retrieval, or deletion of a business object to or from a message store, as follows:

- A `UCCnetMessageReceive` collaboration object initiates persisting a business object for use in a return message by sending it with a `Create` verb to its `ToMessage_Store` port.
- A `UCCnetMessageSend` collaboration object initiates retrieving or deleting a business object by passing it with either a `Retrieve` or `Delete` verb to a `MessageStore` collaboration object through its `ToMessage_Store` port.
- An `ItemCollector` collaboration object initiates deleting a business object by passing it with a `Delete` verb to a `MessageStore` collaboration object through its `message_store` port.
- An `ItemDispatcher` collaboration object initiates persisting or retrieving a business object by passing it with a `Create` or `Retrieve` verb to a `MessageStore` collaboration object through its `LocalMessageStore` port.

In each case, before persisting, retrieving, or deleting the business object, the collaboration object first converts it to a `UCCnetGBO_storable` business object. The `UCCnetGBO_storable` business object is a combination of one instance of an `Catalogue Item Notification`, together with the message header and routing information for the worklist message in which the item was contained. The outgoing port on each collaboration object can then be bound to a persistence mechanism. In the context of the Product Information Management for Retailers solution, this mechanism is a `MessageStore` collaboration object.

The `MessageStore` collaboration object receives the business object on its `From` port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML (which are detailed per command in `MessageStore` collaboration template), stores, retrieves, or deletes the message. When it retrieves a message, it returns it to the calling collaboration object. The key to this record in the message store is saved in the `correlationID` attribute of the `UCCnetGBO_storable` business object.

For detailed information about components used in persisting, retrieving, and deleting items to and from a message store, see the following:

- ItemCollector collaboration template
- ItemDispatcher collaboration template
- MessageStore collaboration template
- UCCnetMessageReceive collaboration template
- UCCnetMessageSend collaboration template
- UCCnetGBO_envelope business object
- UCCnetGBO_storable business object

Persisting, retrieving, updating, or deleting an item to or from a local item store

Various collaboration objects used in the Product Information Management for Retailers solution can initiate the storage, retrieval, updating, or deletion of a business object to or from an item store, as follows:

- ItemValidation, ItemDispatcher, and Process_Reviewed_Item collaboration objects initiate persisting a business object. An ItemValidation collaboration object initiates storing the item prior to its being sent to a process for obtaining missing information or for business review/approval. An ItemDispatcher collaboration object initiates storing an item with cascaded GLNs. A Process_Reviewed_Item collaboration object initiates storing the item after it is synchronized to a back-end system, so the item is available to be used later for validation or back-up. To enable persisting the item in the local item store, set an ItemValidation or Process_Reviewed_Item collaboration object's configuration property `RETAIN_ITEM_IN_LOCAL_STORE` to true; an ItemDispatcher collaboration object persists by default. ItemValidation and ItemDispatcher collaboration objects then send the business object with a Create verb to their *LocalItemStore* ports; the ItemDispatcher collaboration object to its *LocalItemStore* port; and the Process_Reviewed_Item collaboration object, to its *local_store* port.
- An ItemCollector collaboration object initiates retrieving and deleting a Retail_Item business object. The ItemCollector collaboration object sends the item with a Retrieve or Delete verb to its *local_store* port.
- An ItemDispatcher collaboration object initiates updating a Retail_Item business object. The ItemDispatcher collaboration object sends the item with an Update verb to its *LocalItemStore* port.
- A Process_Reviewed_Item collaboration object initiates deleting a business object. To enable deletion of the business object, set a Process_Reviewed_Item collaboration object's configuration property `DELETE_FROM_LOCAL_STORE` to true. It then passes the item with a Delete verb to its *local_store* port.

The ItemValidation and ItemDispatcher collaboration objects' *LocalItemStore* ports, and the ItemCollector and Process_Reviewed_Item collaboration objects' *local_store* ports, are all bound to a persistence mechanism. In the context of the Product Information Management for Retailers solution, this mechanism is an ItemStore collaboration object.

The ItemStore collaboration object receives the business object on its *From* port and, through a series of interactions with the IBM WebSphere Business Integration Data Handler for XML (which are detailed per command in ItemStore collaboration template), stores, retrieves, updates, or deletes the item. When the collaboration

object retrieves an item, it returns it to the calling collaboration object. The key to this record in the item store is saved in the `internals.correlationID` attribute of the `Retail_Item` business object.

Note: If the item has not been stored by another collaboration object and therefore does not have to be deleted, set the `Process_Reviewed_Item` collaboration object's property `DELETE_FROM_LOCAL_STORE` to `false`.

For detailed information about components used in persisting, retrieving, updating, or deleting items to and from an item store, see the following:

- `ItemCollector` collaboration template
- `ItemDispatcher` collaboration template
- `ItemStore` collaboration template
- `ItemValidation` collaboration template
- `Process_Reviewed_Item` collaboration template
- `Retail_Item` business object

Generating database keys

A `UCCnetMessageReceive` collaboration object generates keys to items in each of the stores. A key is guaranteed to be unique for each item and for each type of command. The key for the item store is saved in the `Retail_Item` business object `internals.correlationID` attribute. Do not modify the value of this attribute.

Table 1. Database keys

Database table	Key
item store	<ul style="list-style-type: none"> • For systems with cascaded GLNs: <code>messageHeader.messageIdentifier + notification.sequenceId + internals.cascadedGLns.gln</code> • For systems without cascaded GLNs: <code>messageHeader.messageIdentifier + notification.sequenceId</code>
message store	<ul style="list-style-type: none"> • For systems with cascaded GLNs: <code>messageHeader.messageIdentifier + notification.sequenceId + internals.cascadedGLns.gln</code> • For systems without cascaded GLNs: <code>messageHeader.messageIdentifier + notification.sequenceId</code>
identifier store	<ul style="list-style-type: none"> • For systems supporting XSD XML definition: <code>gtin</code>, <code>topic</code>, <code>dataRecipientGLN</code>, <code>dataSourceGLN</code>, <code>targetMarket</code>, <code>uniqueCreatorID</code> attributes of <code>UCCnetGBO_identifier</code> business object • For systems supporting DTD XML definition: <code>gtin</code>, <code>version</code>, <code>topic</code> attributes of <code>UCCnetGBO_identifier</code> business object

Entries in the item store and message store can possess identical keys since the stores are different tables within the database.

Controlling email

ItemValidation, ItemCollector, ItemDispatcher, and Process_Reviewed_Item collaboration objects can be configured to initiate sending email to alert recipients of item status or when processing errors occur. These collaboration objects link to a Role_Email collaboration object, which actually controls sending the email.

The Role_Email collaboration object allows the contents of the Retail_Item attributes that specify email message text, subject text, and recipients to contain the names of files. These files contain the actual email message text, subject text, and addresses, and can be easily modified without modifying the using collaboration objects. This feature permits messages, subjects, and recipients to be shared among multiple collaboration objects. Also, more than one recipient can be specified to receive email through use of a comma-delimited list. Plus, email message and subject text can be constants that contain variables. The Role_Email collaboration object substitutes data from the processing state or business object into these variables dynamically.

For more information on these topics, see the following sections:

- “Alerting email recipients of item status or processing errors”
- “Specifying message text, subjects, and recipients in external files” on page 55
- “Specifying changing individual or multiple message recipients” on page 55
- “Using substitution variables in message and subject text” on page 56

Alerting email recipients of item status or processing errors

ItemValidation, ItemCollector, ItemDispatcher, and Process_Reviewed_Item collaboration objects can be configured to initiate sending email to alert recipients of item status or when processing errors occur, as follows:

- Alerting of Approved item status — Process_Reviewed_Item collaboration objects only
- Alerting of Accepted item status — Process_Reviewed_Item collaboration objects only
- Alerting of Rejected item status — ItemValidation and Process_Reviewed_Item collaboration objects only
- Alerting of processing errors — ItemValidation, ItemCollector, ItemDispatcher, and Process_Reviewed_Item collaboration objects

Common configuration properties control whether email is sent and delineate the contents of the message text, message subject, and the mail recipients. These collaboration objects link to a Role_Email collaboration object, which actually controls sending the email through use of the sendEmail API. This collaboration object formulates the sendEmail API parameters from the triggering business object (Retail_Item in the Product Information Management for Retailers solution) passed to it from the various collaboration objects.

The following sections detail the use of configuration properties to configure email notification.

Alerting of Approved item status

A Process_Reviewed_Item collaboration object contains the following properties that can be configured to enable email to be sent if an item contains a status of Approved:

SEND_MAIL_ON_APPROVAL

This property controls whether the Retail_Item is sent to the port

connected to the Role_Email collaboration object. In essence, its value of true or false controls whether email is sent. This property must be configured by the user.

APPROVED_EMAIL_ROLE

This property specifies the Retail_Item attribute containing the recipients that are to receive the message. It must be configured by the user. If this property and the SEND_MAIL_ON_APPROVAL property exist, and the Retail_Item has a status of Approved, the value of APPROVED_EMAIL_ROLE is stored in the Retail_Item attribute specified by the Process_Reviewed_Item collaboration object's EMAIL_ROLE_ATTRIBUTE property. By default, this Retail_Item attribute is internals.message_recipient_role. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the Process_Reviewed_Item collaboration object's EMAIL_ROLE_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_RECIPIENT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual recipient or list of recipients or a filename containing this value. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify email recipients by using filenames and the benefits of specifying email recipients in this way.

Note: If the APPROVED_EMAIL_ROLE property does not exist, a value of unknown is set in the Retail_Item attribute specified in the collaboration object's EMAIL_ROLE_ATTRIBUTE.

APPROVED_EMAIL_MSG

This property specifies the Retail_Item attribute containing the appropriate message text to be sent. If the SEND_MAIL_ON_APPROVAL and APPROVED_EMAIL_ROLE properties exist, and the Retail_Item has a status of Approved, the value of APPROVED_EMAIL_MSG is stored in the Retail_Item attribute specified by the Process_Reviewed_Item collaboration object's EMAIL_MSG_ATTRIBUTE property. By default, this Retail_Item attribute is internals.message_text. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the Process_Reviewed_Item collaboration object's EMAIL_MSG_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_TEXT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual message text or a filename containing the message text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify message text by using filenames and the benefits of specifying message text in this way.

APPROVED_EMAIL_SUBJECT

This property specifies the Retail_Item attribute containing the appropriate message subject. If the SEND_MAIL_ON_APPROVAL and APPROVED_EMAIL_ROLE properties exist, and the Retail_Item has a status of Approved, the value of APPROVED_EMAIL_SUBJECT is stored in the Retail_Item attribute specified by the Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property. By default, this Retail_Item attribute is internals.message_subject. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property must match

the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_SUBJECT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual subject text or a filename containing the subject text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify subject text by using filenames and the benefits of specifying subject text in this way.

The collaboration object sends the business object to its *mail* port and the `Role_Email` collaboration object receives it on its *From* port. The `Role_Email` collaboration object then formulates the `sendEmail` API parameters from the values of the `EMAIL_ROLE_ATTRIBUTE`, `EMAIL_MSG_ATTRIBUTE`, and `EMAIL_SUBJECT_ATTRIBUTE` properties passed to it.

For detailed information about the collaboration objects and business object mentioned in this section, see the following:

- `Process_Reviewed_Item` collaboration template
- `Role_Email` collaboration template
- `Retail_Item` business object

Alerting of Accepted item status

A `Process_Reviewed_Item` collaboration object contains the following properties that can be configured to enable email to be sent if an item contains a status of Accepted:

SEND_MAIL_ON_ACCEPTED

This property controls whether the `Retail_Item` is sent to the port connected to the `Role_Email` collaboration object. In essence, its value of true or false controls whether email is sent. This property must be configured by the user.

ACCEPTED_EMAIL_ROLE

This property specifies the `Retail_Item` attribute containing the recipients that are to receive the message. It must be configured by the user. If this property and the `SEND_MAIL_ON_ACCEPTED` property exist, and the `Retail_Item` has a status of Accepted, the value of `ACCEPTED_EMAIL_ROLE` is stored in the `Retail_Item` attribute specified by the `Process_Reviewed_Item` collaboration object's `EMAIL_ROLE_ATTRIBUTE` property. By default, this `Retail_Item` attribute is `internals.message_recipient_role`. To successfully use the `Role_Email` collaboration object as the notifying collaboration object, the `Retail_Item` attribute specified in the `Process_Reviewed_Item` collaboration object's `EMAIL_ROLE_ATTRIBUTE` property must match the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_RECIPIENT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual recipient or list of recipients or a filename containing this value. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify email recipients by using filenames and the benefits of specifying email recipients in this way.

Note: If the `ACCEPTED_EMAIL_ROLE` property does not exist, a value of unknown is set in the `Retail_Item` attribute specified in the collaboration object's `EMAIL_ROLE_ATTRIBUTE`.

ACCEPTED_EMAIL_MSG

This property specifies the `Retail_Item` attribute containing the appropriate

message text to be sent. If the `SEND_MAIL_ON_ACCEPTED` and `ACCEPTED_EMAIL_ROLE` properties exist, and the `Retail_Item` has a status of `Accepted`, the value of `ACCEPTED_EMAIL_MSG` is stored in the `Retail_Item` attribute specified by the `Process_Reviewed_Item` collaboration object's `EMAIL_MSG_ATTRIBUTE` property. By default, this `Retail_Item` attribute is `internals.message_text`. To successfully use the `Role_Email` collaboration object as the notifying collaboration object, the `Retail_Item` attribute specified in the `Process_Reviewed_Item` collaboration object's `EMAIL_MSG_ATTRIBUTE` property must match the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_TEXT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual message text or a filename containing the message text. See the section “Specifying message text, subjects, and recipients in external files” on page 55 for more information on how to specify message text by using filenames and the benefits of specifying message text in this way.

ACCEPTED_EMAIL_SUBJECT

This property specifies the `Retail_Item` attribute containing the appropriate message subject. If the `SEND_MAIL_ON_ACCEPTED` and `ACCEPTED_EMAIL_ROLE` properties exist, and the `Retail_Item` has a status of `Accepted`, the value of `ACCEPTED_EMAIL_SUBJECT` is stored in the `Retail_Item` attribute specified by the `Process_Reviewed_Item` collaboration object's `EMAIL_SUBJECT_ATTRIBUTE` property. By default, this `Retail_Item` attribute is `internals.message_subject`. To successfully use the `Role_Email` collaboration object as the notifying collaboration object, the `Retail_Item` attribute specified in the `Process_Reviewed_Item` collaboration object's `EMAIL_SUBJECT_ATTRIBUTE` property must match the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_SUBJECT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual subject text or a filename containing the subject text. See the section “Specifying message text, subjects, and recipients in external files” on page 55 for more information on how to specify subject text by using filenames and the benefits of specifying subject text in this way.

The collaboration object sends the business object to its *mail* port and the `Role_Email` collaboration object receives it on its *From* port. The `Role_Email` collaboration object then formulates the `sendEmail` API parameters from the values of the `EMAIL_ROLE_ATTRIBUTE`, `EMAIL_MSG_ATTRIBUTE`, and `EMAIL_SUBJECT_ATTRIBUTE` properties passed to it.

For detailed information about the collaboration objects and business object mentioned in this section, see the following:

- `Process_Reviewed_Item` collaboration template
- `Role_Email` collaboration template
- `Retail_Item` business object

Alerting of Rejected item status

`ItemValidation` and `Process_Reviewed_Item` collaboration objects contain the following properties that can be configured to enable email to be sent if an item contains a status of `Rejected`:

SEND_MAIL_ON_REJECTION

This property controls whether the `Retail_Item` is sent to the port

connected to the Role_Email collaboration object. In essence, its value of true or false controls whether email is sent. This property must be configured by the user.

REJECT_EMAIL_ROLE

This property specifies the Retail_Item attribute containing the recipients that are to receive the message. It must be configured by the user. If this property and the SEND_MAIL_ON_REJECTION property exist (and, in the ItemValidation collaboration object, are not BLANK), and the Retail_Item has a status of Rejected, the value of REJECT_EMAIL_ROLE is stored in the Retail_Item attribute specified by the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_ROLE_ATTRIBUTE property. By default, this Retail_Item attribute is internals.message_recipient_role. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_ROLE_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_RECIPIENT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual recipient or list of recipients or a filename containing this value. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify email recipients by using filenames and the benefits of specifying email recipients in this way.

Note: In the Process_Reviewed_Item collaboration object, if the REJECT_EMAIL_ROLE property does not exist, a value of unknown is set in the Retail_Item attribute specified by the EMAIL_ROLE_ATTRIBUTE property. In the ItemValidation collaboration object, if the REJECT_EMAIL_ROLE property exists but is BLANK, no notification is executed (no default mail recipient exists).

REJECT_EMAIL_MSG

This property specifies the Retail_Item attribute containing the appropriate message text to be sent. If the SEND_MAIL_ON_REJECTION and REJECT_EMAIL_ROLE properties exist (and, in the ItemValidation collaboration object, are not BLANK), and the Retail_Item has a status of Rejected, the value of REJECT_EMAIL_MSG is stored in the Retail_Item attribute specified by the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_MSG_ATTRIBUTE property. By default, this Retail_Item attribute is internals.message_text. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_MSG_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_TEXT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual message text or a filename containing the message text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify message text by using filenames and the benefits of specifying message text in this way.

Note: In the ItemValidation collaboration object, if the REJECT_EMAIL_MSG property exists but is BLANK, a default value is placed in the Retail_Item attribute specified by the EMAIL_MSG_ATTRIBUTE property.

REJECT_EMAIL_SUBJECT

This property specifies the Retail_Item attribute containing the appropriate message subject. If the SEND_MAIL_ON_REJECTION and REJECT_EMAIL_ROLE properties exist (and, in the ItemValidation collaboration object, are not BLANK), and the Retail_Item has a status of Rejected, the value of REJECT_EMAIL_SUBJECT is stored in the Retail_Item attribute specified by the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property. By default, this Retail_Item attribute is internal.s.message_subject. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the ItemValidation or Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_SUBJECT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual subject text or a filename containing the subject text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify subject text by using filenames and the benefits of specifying subject text in this way.

Note: In the ItemValidation collaboration object, if the REJECT_EMAIL_SUBJECT property exists but is BLANK, a default value is placed in the Retail_Item attribute specified by the EMAIL_SUBJECT_ATTRIBUTE property.

The ItemValidation collaboration object sends the object to its *Notify* port; the Process_Reviewed_Item collaboration object, to its *mail* port. The Role_Email collaboration object receives it on its *From* port and then formulates the sendEmail API parameters from the values of the EMAIL_ROLE_ATTRIBUTE, EMAIL_MSG_ATTRIBUTE, and EMAIL_SUBJECT_ATTRIBUTE properties passed to it.

For detailed information about the collaboration objects and business object mentioned in this section, see the following:

- ItemValidation collaboration template
- Process_Reviewed_Item collaboration template
- Role_Email collaboration template
- Retail_Item business object

Alerting of processing errors

ItemValidation, ItemCollector, ItemDispatcher, and Process_Reviewed_Item collaboration objects contain the following properties that can be configured to enable email to be sent if errors are detected during processing:

SEND_MAIL_ON_ERROR

This property controls whether the Retail_Item is sent to the port connected to the Role_Email collaboration object. In essence, its value of true or false controls whether email is sent. This property must be configured by the user.

ERROR_EMAIL_ROLE

This property specifies the Retail_Item attribute containing the recipients that are to receive the message. It must be configured by the user. If an error occurs and this property and the SEND_MAIL_ON_ERROR property exist (and, in the ItemValidation collaboration object, are not BLANK), the

value of `ERROR_EMAIL_ROLE` is stored in the `Retail_Item` attribute specified by the `ItemValidation`, `ItemCollector`, `ItemDispatcher`, or `Process_Reviewed_Item` collaboration object's `EMAIL_ROLE_ATTRIBUTE` property. By default, this `Retail_Item` attribute is `internals.message_recipient_role`. To successfully use the `Role_Email` collaboration object as the notifying collaboration object, the `Retail_Item` attribute specified in the `ItemValidation`, `ItemCollector`, `ItemDispatcher`, or `Process_Reviewed_Item` collaboration object's `EMAIL_ROLE_ATTRIBUTE` property must match the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_RECIPIENT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual recipient or list of recipients or a filename containing this value. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify email recipients by using filenames and the benefits of specifying email recipients in this way.

Note: In the `Process_Reviewed_Item` collaboration object, if the `ERROR_EMAIL_ROLE` property does not exist, a value of unknown is set in the `Retail_Item` attribute specified by the `EMAIL_ROLE_ATTRIBUTE` property. In the `ItemValidation` collaboration object, if the `ERROR_EMAIL_ROLE` property exists but is `BLANK`, no notification is executed (no default mail recipient exists).

**ERROR_EMAIL_MSG (ItemValidation, ItemDispatcher, and Process_Reviewed_Item collaboration objects only),
ERROR_RETRIEVE_EMAIL_MSG (ItemCollector collaboration object only),
ERROR_SEND_EMAIL_MSG (ItemCollector collaboration object only)**

These properties specify the `Retail_Item` attribute containing the appropriate message text to be sent. If an error occurs and the `SEND_MAIL_ON_ERROR` and `ERROR_EMAIL_ROLE` properties exist (and, in the `ItemValidation` collaboration object, are not `BLANK`), the value of `ERROR_EMAIL_MSG`, `ERROR_RETRIEVE_EMAIL_MSG`, or `ERROR_SEND_EMAIL_MSG` is stored in the `Retail_Item` attribute specified by the `ItemValidation`, `ItemCollector`, `ItemDispatcher`, or `Process_Reviewed_Item` collaboration object's `EMAIL_MSG_ATTRIBUTE` property. By default, this `Retail_Item` attribute is `internals.message_text`. To successfully use the `Role_Email` collaboration object as the notifying collaboration object, the `Retail_Item` attribute specified in the `ItemValidation`, `ItemCollector`, `ItemDispatcher`, or `Process_Reviewed_Item` collaboration object's `EMAIL_MSG_ATTRIBUTE` property must match the `Retail_Item` attribute specified in the `Role_Email` collaboration object's `MSG_TEXT_ATTRIBUTE` property. This `Retail_Item` attribute can contain either the actual message text or a filename containing the message text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify message text by using filenames and the benefits of specifying message text in this way.

Note: In the `ItemValidation` collaboration object, if the `ERROR_EMAIL_MSG` property exists but is `BLANK`, a default value is placed in the `Retail_Item` attribute specified by the `EMAIL_MSG_ATTRIBUTE` property. In the `ItemCollector` collaboration object, if the `ERROR_RETRIEVE_EMAIL_MSG` or `ERROR_SEND_EMAIL_MSG` properties exist but are `BLANK`, a default value is placed in the `Retail_Item` attribute specified by the

EMAIL_MSG_ATTRIBUTE property. In the ItemDispatcher collaboration object, if the ERROR_EMAIL_MSG property exists but is BLANK, the exception that occurred in the collaboration object is used as the message text.

ERROR_EMAIL_SUBJECT

This property specifies the Retail_Item attribute containing the appropriate message subject. If an error occurs and the SEND_MAIL_ON_ERROR and ERROR_EMAIL_ROLE properties exist (and, in the ItemValidation collaboration object, are not BLANK), the value of ERROR_EMAIL_SUBJECT is stored in the Retail_Item attribute specified by the ItemValidation, ItemCollector, ItemDispatcher, or Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property. By default, this Retail_Item attribute is `internals.message_subject`. To successfully use the Role_Email collaboration object as the notifying collaboration object, the Retail_Item attribute specified in the ItemValidation, ItemCollector, ItemDispatcher, or Process_Reviewed_Item collaboration object's EMAIL_SUBJECT_ATTRIBUTE property must match the Retail_Item attribute specified in the Role_Email collaboration object's MSG_SUBJECT_ATTRIBUTE property. This Retail_Item attribute can contain either the actual subject text or a filename containing the subject text. See the section "Specifying message text, subjects, and recipients in external files" on page 55 for more information on how to specify subject text by using filenames and the benefits of specifying subject text in this way.

Note: In the ItemValidation collaboration object, if the ERROR_EMAIL_SUBJECT property exists but is BLANK, a default value is placed in the Retail_Item attribute specified by the EMAIL_SUBJECT_ATTRIBUTE property.

The ItemValidation and ItemDispatcher collaboration objects send the object to their *Notify* ports; the ItemCollector collaboration object, to its *email* port; the Process_Reviewed_Item collaboration object, to its *mail* port. The Role_Email collaboration object receives it on its *From* port and then formulates the `sendEmail` API parameters from the values of the EMAIL_ROLE_ATTRIBUTE, EMAIL_MSG_ATTRIBUTE, and EMAIL_SUBJECT_ATTRIBUTE properties passed to it.

Using the QUALIFICATION_FAILED_EMAIL_MSG property

(Process_Reviewed_Item collaboration object only): Set this attribute to the appropriate message text to be sent if the status of the item returned on the Process_Reviewed_Item collaboration object's *reprocess* port from the ItemValidation collaboration object is Rejected.

For detailed information about the collaboration objects and business object mentioned in this section, see the following:

- ItemCollector collaboration template
- ItemDispatcher collaboration template
- ItemValidation collaboration template
- Process_Reviewed_Item collaboration template
- Role_Email collaboration template
- Retail_Item business object

Specifying message text, subjects, and recipients in external files

A Role_Email collaboration object allows the contents of the Retail_Item attributes that specify email message text, subject text, and recipients to contain the names of files. These files contain the actual email message text, subject text, and addresses, and can be easily modified without modifying the using collaboration objects. This feature permits messages, subjects, and recipients to be shared among multiple collaboration objects. A solution's messages, subjects, and recipients can all be contained in one easily modifiable directory.

The Role_Email collaboration object uses the following configuration properties to identify the Retail_Item attributes containing the email message text, subject text, and recipients:

MSG_TEXT_ATTRIBUTE

Identifies the Retail_Item attribute containing the message text, by default, `internals.message_text`.

MSG_SUBJECT_ATTRIBUTE

Identifies the Retail_Item attribute containing the subject text, by default, `internals.message_subject`.

MSG_RECIPIENT_ATTRIBUTE

Identifies the Retail_Item attribute containing the recipient or list of recipients, by default, `internals.message_recipient_role`.

The collaboration object distinguishes whether the content of the Retail_Item attributes are actual values or filenames through use of its configuration property `FILE_NAME_PREFIX`. If the value of the Retail_Item attribute specified in the `MSG_TEXT_ATTRIBUTE`, `MSG_SUBJECT_ATTRIBUTE`, or `MSG_RECIPIENT_ATTRIBUTE` property is prefixed with the String specified in `FILE_NAME_PREFIX`, the Role_Email collaboration object interprets the rest of the value as a filename. The collaboration object reads the value of the file into a String variable in preparation for further processing. Files must be identified by their fully qualified names.

For instance, if the filename containing the email recipient(s) is `c:\Email_Files\CategoryManagerRole.txt` and the value of `FILE_NAME_PREFIX` is `@`, set the value of the Retail_Item attribute identified by the `MSG_RECIPIENT_ATTRIBUTE` property, as follows:

```
@c:\Email_Files\CategoryManagerRole.txt
```

If the values of the Retail_Item attributes specified in the `MSG_TEXT_ATTRIBUTE`, `MSG_SUBJECT_ATTRIBUTE`, and `MSG_RECIPIENT_ATTRIBUTE` properties do not start with the String specified in `FILE_NAME_PREFIX`, the Role_Email collaboration object obtains the email values directly from the attributes. If the Retail_Item attributes for message text or subject text contain no values, the Role_Email collaboration object supplies default values for them.

Specifying changing individual or multiple message recipients

A Role_Email collaboration object allows all email messages to be routed to an administrator or to a specific role in an organization (like a Category Manager), without the need to maintain the email recipient's fully qualified email address in every collaboration object that might send email. By placing the email address in an external file, if the address changes, the file can be modified without having to reconfigure the using collaboration objects. More than one recipient can be

specified to receive the email through use of a comma-delimited list. The comma-delimited list can be specified in the business object attribute or in the external file pointed to by the attribute.

Using substitution variables in message and subject text

Email message and subject text can be constants that contain variables. A Role_Email collaboration object substitutes data from the processing state or business object into these variables dynamically. For instance, in the message

An item with GTIN `${item.catalogueItem.tradeItem.itemInformation. \ tradeItemIdentification.gtin}` was approved and synchronized to the back end system

the GTIN value represented by `${item.catalogueItem.tradeItem.itemInformation.tradeItemIdentification.gtin}` is filled in automatically during the generation of this message.

Variables to be substituted must be enclosed in prefix and suffix characters. The substitution variable characters are defined in the values of the Role_Email collaboration object's configuration properties `SUBSTITUTION_VARIABLE_PREFIX` and `SUBSTITUTION_VARIABLE_SUFFIX`. In the following example, these properties are set as follows:

```
SUBSTITUTION_VARIABLE_PREFIX = ${
SUBSTITUTION_VARIABLE_SUFFIX = }
```

As a result, the substitution variables in the email message and subject text must appear as:

`${variable_name}`

Note: These characters might have to be changed to meet National Language requirements.

The supported values for *variable_name*, along with the values that the collaboration object actually inserts in the text, are as follows:

ROOT

Substitutes the entire triggering business object.

Date Substitutes the current date.

getName

Substitutes the name of the triggering business object.

getVerb

Substitutes the verb of the triggering business object.

Any attribute name

Substitutes the value of the named attribute from the triggering business object. If the value for *variable_name* does not match one of the specific values above, the collaboration object interprets it as the name of a business object attribute.

Logging

ItemValidation, ItemCollector, Process_Reviewed_Item, and Role_Email collaboration objects can log error situations and, in some cases, item status. A Role_Email collaboration object can also log each time an email message is sent.

Logging errors

If error conditions are encountered during any stage of processing, ItemValidation, ItemCollector, and Process_Reviewed_Item collaboration objects do the following:

- Set the Retail_Item business object's attribute value named in the configuration property ITEM_STATUS_ATTRIBUTE to Error.
- Log the error in the configured log destination.
- Return the object to the calling collaboration object through the *From* port.

If any errors are detected when email is sent, the Role_Email collaboration object logs them in the log destination.

Note: For error logging to occur in ItemCollector and Process_Reviewed_Item collaboration objects, tracing must be enabled. For error logging to occur in a Role_Email collaboration object, its LOG_ERROR configuration property must be set to true.

Logging item status

The values of configuration properties in ItemValidation, ItemCollector, and Process_Reviewed_Item collaboration objects control the logging of item status. A value of true in the following properties enables the business object to be logged in the configured log destination, as follows:

LOG_ERROR_ITEM

This property specifies whether the collaboration object logs the business object being processed (in addition to the error) when an error is detected during processing.

LOG_REVIEW_ITEM (ItemValidation collaboration objects only)

This property specifies whether the collaboration object logs the business object being processed when the object is successfully processed. The Review item status means that the item continues to be processed for required missing attribute data review or approval.

LOG_APPROVED_ITEM (Process_Reviewed_Item collaboration objects only)

This property specifies whether the collaboration object logs the business object being processed when the object status is Approved.

LOG_ACCEPTED_ITEM (Process_Reviewed_Item collaboration objects only)

This property specifies whether the collaboration object logs the business object being processed when the object status is Accepted.

LOG_REJECTED_ITEM (ItemValidation and Process_Reviewed_Item collaboration objects only)

This property specifies whether the collaboration objects log the business object being processed when the object status is Rejected.

Solution requirements dictate the type of logging that occurs, as well as when it occurs. It is recommended that the LOG_ERROR_ITEM property always be set to true so that any item that fails is logged for diagnostic purposes. To log an item only after it completes the business review/approval process, set the LOG_APPROVED_ITEM and LOG_REJECTED_ITEM properties of the Process_Reviewed_Item collaboration object to true and the LOG_REVIEW_ITEM and LOG_REJECTED_ITEM properties of the ItemValidation collaboration object and the LOG_ACCEPTED_ITEM property of the Process_Reviewed_Item collaboration object to false. To log an item both before and after it enters the business review/approval process, set all logging collaboration properties of ItemValidation and Process_Reviewed_Item collaboration objects to true.

Note: Use separate files for logging and tracing. Use logging files to maintain persistent records of processed data. Use tracing files to diagnose problems and to show the flow of an item through the Product Information Management for Retailers solution. The Log Viewer tool has log and trace file filters that enable users to view the log or trace records for a particular business object or collaboration object.

Logging that mail is sent

Set a Role_Email collaboration object's configuration property LOG_ALL_MAIL to true to place an entry in the configured log destination each time an email message is generated.

Tracing

All collaboration objects based on collaboration templates included in the Product Information Management for Retailers solution provide tracing capabilities to record logical flows and data processed. Users can enable tracing for a particular collaboration object by selecting the collaboration object in the System Manager, displaying its properties, and, on the **Collaboration General Properties** tab, selecting a trace level greater than 0 from the **System trace level** field.

Enable tracing for one or more collaboration objects when a reproducible problem occurs. If a problem occurs only once during processing, leave the tracing function enabled continually so that the first occurrence of the failure is captured. However, leaving the tracing function enabled continually can degrade performance. Clear the trace file periodically to simplify viewing and filtering it.

Note: Use separate files for tracing and logging. Use tracing files to diagnose problems and to show the flow of an item through the Product Information Management for Retailers solution. Use logging files to maintain persistent records of processed data. The Log Viewer tool has trace and log file filters that enable users to view the trace or log records for a particular business object or collaboration object.

Handling solution processing errors

The Product Information Management for Retailers solution is designed so that its component collaboration objects gather as much information as possible about an error situation, notify the appropriate users that an error has occurred, and then continue with item processing. Only severe system errors cause processing by a collaboration object to end as an unfinished flow. This design prevents situations in which a restarted flow might not have all of the resources available to it to reprocess the item or might initiate tasks that should not be repeated. The following examples describe some of the problems that can occur if item processing is restarted:

- If an item is approved in the business process but the synchronization fails on the back end, if the process is restarted, the item will not be found in the item store (it has already been deleted).
- If an item is synchronized to the back end but an error occurs when sending the response to UCCnet, if the process is restarted, the item will be synchronized to the back end again.

The following lists a few of the error situations that can occur when processing an item:

- A UCCnetMessageReceive collaboration object cannot parse the incoming message into a specific item request.
- A database access failure occurs during processing by ItemValidation, ItemCollector, ItemDispatcher, or Process_Reviewed_Item collaboration objects.
- The business review/approval process ends in error.
- An item cannot be synchronized to the back-end system.
- A response cannot be sent back to UCCnet.

Where an error occurs in the solution processing determines the type of recovery that is needed.

Diagnosing error conditions

The following sections suggest some procedures to follow to determine how an item failed in processing and where in the processing flow it failed.

Determining the problem from email messages

The Product Information Management for Retailers solution uses email routed to specific user IDs to alert users when errors occur during item processing. The email provides basic information about the errors. Email messages can be configured or modified. Refer to the section “Controlling email” on page 47 for more details on configuring and controlling email.

Determining the problem from log or trace files

Use the log file to determine how far an item progressed through the solution flow before a problem occurred. Filter the log file to examine specific collaboration object or business object events. If the log does not provide enough information about the failure, activate tracing for one or more collaboration objects and examine the trace file for more details. Filter the trace file to search for specific collaboration object or business object events. See the sections “Logging” on page 56 and “Tracing” on page 58 for more information.

Determining the point of failure

The point of failure in solution processing determines the method used to restart item processing. The following sections outline check points that can help pinpoint where in item processing an error occurred.

Check that the item exists in the message store: Use database user interface methods to examine the message store. Search for a message that correlates to the item in error. Refer to the key of the message store to correlate the item.

Check that the item completed validation checks: Examine the log file for messages originating from the ItemValidation collaboration object that indicate that the item completed validation checks.

Check that the item reached the business review/approval process: Use the user interface for the business review/approval process to search for the item in question. If the item reached the business review/approval process and the business review/approval process used is WebSphere MQ Workflow, the item appears as a new task in the workflow.

Check that the item completed the business review/approval process: The log information related to the ItemCollector collaboration object indicates the state of the item as it arrives from the business review/approval process. If the error occurred in this process, the status of the item passed to the ItemCollector collaboration object is Error. In this case, review the section “Check that the item reached the business review/approval process” for further diagnostic procedures.

If the error resulted from the ItemCollector collaboration object not being able to read the item from the item store, information in the log or trace files can identify the failure. In this case, follow the procedures in the section “Recovering from error conditions” to remove the item from the stores and restart the item from UCCnet.

Check that item synchronization to the back end occurred: In the context of the Product Information Management for Retailers solution, the JTextConnector synchronizes the item to a back-end file system by writing it to a flat file. Examine the directory containing the item files to see if the failed item was written to a file. If it was, the item was synchronized properly and the error occurred at a later stage.

If the item was not written to a file, the error occurred prior to processing performed by the Process_Reviewed_Item collaboration object. Most likely, no response was sent to UCCnet.

Check that a response was sent to UCCnet: Verify that a response was sent to UCCnet by examining the logging and tracing information related to the UCCNetMessageSend collaboration object or the AS2 channel output directory.

If the response sent to UCCnet was Synchronised (for systems using the XSD XML definition) or Authorized (for systems using the DTD XML definition), the item was synchronized to the back-end system before the error occurred.

If the response sent was Rejected, most likely the item was rejected. The error might have occurred after the response was sent. Normally, a UCCNetMessageSend collaboration object is configured to *not* send responses if errors occur during item processing.

If no response was sent for the item, the item can be reprocessed.

Recovering from error conditions

After determining the nature of the error and correcting it, the existing item must be removed from the process that ended in error so it can be reprocessed. Unless the item is to be restarted by using a test connector (see the section “Removing an item from processing” for more information), the item must be removed from every store and from the business review/approval process before processing can be restarted.

Removing an item from processing

Perform these steps to remove the Retail_Item business object from processing:

1. Remove the item from the business review/approval process (WebSphere MQ Workflow in the context of the Product Information Management for Retailers solution) by referring to WebSphere MQ Workflow documentation that describes how to remove the instance of the WebSphere MQ Workflow process that ended in error.
2. Delete the item from the item store by finding the value of its `internals.correlationID` attribute in the log file. Use this value as the key to the item store and find and delete the Retail_Item record.
3. Delete the item from the message store finding the value of the `correlationID` attribute of the related UCCnetGBO_storable business object in the log file. Use this value as the key to the message store and find and delete the record.
4. If the original request was a NEW_ITEM, DATA_CHANGE, WITHDRAW, or DE_LIST, delete the item from the identifier store by finding in the log file the

values of the `gtin`, `version`, and `topic` attributes of the related `UCCnetGBO_identifier` business object for DTD processing or the `gtin`, `topic`, `dataRecipientGLN`, `dataSourceGLN`, `targetMarket`, and `uniqueCreatorID` attributes of the related `UCCnetGBO_identifier` business object for XSD processing. Use these values as the key to the identifier store and find and delete the record.

Restarting item processing

To restart item processing, do one of the following:

- Regenerate the message from UCCnet by obtaining the GTIN and trading partner GLN from the log entries related to the item and using the UCCnet Graphical User Interface (GUI) to request a republication of the item. This action causes a message to arrive for the item and the item is processed normally.
- Regenerate the message from the UCCnet backup area. The original message containing the failed item can be copied from the UCCnet staging directory to the connector processing directory.
- Use a test connector to restart the item from the item store if the following conditions exist:
 - The item failed in the back-end process
 - The item was synchronized
 - No message was sent to UCCnet
 - The item still exists in the item, identifier, and message stores

To use a test connector to restart the item, do the following:

1. Do not delete the item from the item store, message store, or identifier store.
2. Start a test connector and connect it to an `ItemCollector` collaboration object flow. This requires configuring a set of collaboration objects and connectors to imitate a normal back-end flow.
3. Create a `Retail_Item` business object in the test connector and set the value of its `internals.correlationID` attribute equal to the value of the `internals.correlationID` attribute for the item found in the log.
4. Send the item. The `ItemCollector` collaboration object receives it and processes it normally.

Handling data from other data sources

The Product Information Management for Retailers solution can be extended to handle different item sources in addition to UCCnet, including the World Wide Retail Exchange (WWRE), the Universal Data Exchange (UDEX), or a user interface. Items from various sources can differ significantly in the architecture of the data that comprises them.

Extending the solution to handle a single data source other than UCCnet

To handle data from a source other than UCCnet, do the following:

1. Replace the `UCCnetMessageReceive` and `UCCnetMessageSend` collaboration templates with custom collaboration templates that generate collaboration objects that can parse incoming messages from and produce appropriate response messages to the new data source.
2. Modify the `Retail_Item` business object `item` attribute type (by default, `Retailer_item`) into a different child business object that represents the new

architecture of the item data. It is recommended that this child business object closely follow the architecture of the data source (i.e., UCCnet publication, WWRE item, etc.).

3. Modify all solution maps that convert Retail_Item business objects to instead convert items with the new data architecture.
4. Configure all the configuration properties that point to attributes in the Retail_Item business object to instead point to the appropriate attributes in the new data architecture.

Very few, if any, changes need to be made to individual collaboration objects.

Extending the solution to handle multiple data sources

To handle items from multiple data sources, the front end of the solution must include collaboration objects that can parse each type of new item. These collaboration objects must be based on custom collaboration templates similar to the UCCnetMessageReceive collaboration template. Each of these front-end collaboration objects must be attached to an ItemValidation collaboration object. The ItemValidation collaboration objects operate identically for each front-end parsing collaboration object, routing the different types of Retail_Item business objects to ItemDispatcher collaboration objects. If the various data sources use the same business review/approval process, they can use a common data container. Appropriate maps must be created that convert the different types of Retail_Item business objects to the common data container.

When the business review/approval process completes, the common data container is sent to the WebSphere MQ Workflow queue. The existing Product Information Management for Retailers solution uses the MQWF_Retail_Item_to_Retail_Item map to convert the WebSphere MQ Workflow container into a Retail_Item business object. However, this map, which is used by the WebSphereMQWorkflowConnector, cannot produce different types of Retail_Item business objects. Since the incoming Retail_Item business objects are from different sources, the Retail_Item business objects going to the back end are going to be different as well.

A collaboration template (similar to the ItemCollector collaboration template) must be created that develops and builds these different business objects. A collaboration object based on this custom collaboration template can possibly use the call_map API and have a different To port for each data source producing a Retail_Item business object.

Notices and Trademarks

Proprietary Information

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

Notices

IBM may not offer the products, services, or features discussed in this document in all 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 give 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.

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 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 Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
AS/400e
CrossWorlds
DB2
DB2 Universal Database
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
OS/400
Tivoli
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Solaris, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UCC and UCCnet are trademarks of Uniform Code Council, Inc., UCCnet, Inc. or both, in the United States, other countries, or both.

UCCnet Messaging is a product and/or trademark of UCCnet and is used with permission.

Other company, product, or service names may be trademarks or service marks of others.

IBM WebSphere InterChange Server Version 4.2.2

IBM WebSphere Business Integration Toolset Version 4.2.2

IBM WebSphere Business Integration Adapters Version 2.4



