IBM WebSphere Business Integration Express for Item Synchronization

Solution Development Guide

Version 4.3

Note!
Before using this information and the product it supports, read the general information under "Notices and Trademarks" on page 59.

(10October2003)

This edition of this document applies to *IBM® WebSphere® Business Integration Express for Item Synchronization*Version 4.3, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you.

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.

Contents

Solution Development Guide

The IBM® WebSphere® Business Integration Express for Item Synchronization product workflow is composed of business objects, collaboration objects, connectors, and maps. These basic components work together to enable supply-side trading partners to automatically add items to, update or delist items within, or withdraw items from UCCnet® when item updates are made in their Enterprise Resource Planning (ERP) applications. When an update is made in a supplier's ERP system, item data is automatically validated, reformatted, and sent to the UCCnet standard registry. Suppliers can also communicate new or updated item information to subscribing trading partners via UCCnet. Thus, enterprise data is synchronized with item data sent outside the enterprise.

About this document

The IBM WebSphere Business Integration Express for Item Synchronization product includes InterChange Server Express, the associated Toolset Express product, the Item Synchronization Collaboration, and a set of software integration adapters. Together they provide business process integration and connectivity among leading e-business technologies and enterprise applications as well as integration with the UCCnet GLOBALregistry.

The *Solution Development Guide* describes the internal processing of the WebSphere Business Integration Express for Item Synchronization product.

Audience

This document is intended for programmers who design and implement workflows using the product and who might participate in designing customizations to solutions that are based upon this product. 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 WebSphere Business Integration Express for Item Synchronization product.

Programmers must also have experience with the respective operating systems on which their implementations are installed.

Related documents

The complete set of documentation describes the features and components common to all WebSphere Business Integration Express for Item Synchronization installations and includes reference material on specific components.

The Solution Development Guide does not contain an overview of all of the WebSphere Business Integration Express for Item Synchronization product components or how to install them. For additional information on the individual components of the product, see the IBM® WebSphere® Business Integration Express for Item Synchronization documentation library in the InfoCenter.

New in this release

This is the first release of WebSphere Business Integration Express for Item Synchronization V4.3.

How the Solution Development Guide is organized

The Solution Development Guide introduces the mechanics of the WebSphere Business Integration Express for Item Synchronization product by first presenting sample, high-level, step-by-step workflows of how the product handles the following scenarios:

- ItemAdd — Described in the sections "ItemAdd workflow: adding a new item to UCCnet (DTD support)" on page 4 and "ItemAdd workflow: adding a new item to UCCnet (schema support)" on page 22.
- ItemPublicationAdd — Described in the section "ItemPublicationAdd workflow: making a new item available to trading partners and processing their responses to it" on page 7 and used with DTD support. (ItemAdd and ItemPublicationAdd workflows normally occur sequentially.)
- CatalogueItemNotification_Add — Described in the section "CatalogueItemNotification_Add workflow" on page 24 and used with schema support. (ItemAdd and CatalogueItemNotification_Add workflows normally occur sequentially.)
- ItemChange — Described in the sections "ItemChange workflow: updating item information in UCCnet (DTD support)" on page 11 and "ItemChange workflow: updating item information in UCCnet (schema support)" on page 26.
- ItemPublicationChange — Described in the section "ItemPublicationChange workflow: making updated item information available to trading partners and processing their responses to it" on page 14. (ItemChange and ItemPublicationChange workflows normally occur sequentially.)
- CatalogueItemNotification_Change — Described in the section "CatalogueItemNotification_Change workflow: making updated item information available to trading partners and processing their responses to it" on page 28and used with schema support. (ItemAdd and CatalogueItemNotification_Change workflows normally occur sequentially.)
- ItemDelist — Described in the sections "ItemDelist workflow: making an item permanently unavailable to trading partners (DTD support)" on page 19 and "ItemDelist workflow: making an item permanently unavailable to trading partners (schema support)" on page 30.
- ItemWithdrawal — Described in the sections "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (DTD support)" on page 20 and "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (schema support)" on page 32.

Many steps contain links to detailed conceptual information about the mechanics of the product 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:

- "Checking that item data exists for fields required by UCCnet" on page 33 details how a UCCnet_ItemSync collaboration object ensures that the business object to be passed to UCCnet contains data in all of the fields for which UCCnet requires data.
- "Using the PROCESSED_GTIN table" on page 34 describes how the product populates and maintains data in the provided PROCESSED_GTIN relational

table, which permits a UCCnet_processWorklist collaboration object to process incoming INITIAL_ITEM_LOAD_REQUEST commands without the need to communicate with the back-end ERP system.

- "Using the audit_log table" on page 35 provides information on how the product populates and maintains data in the provided audit_log relational table, used to track events associated with UCCnet activities to support complete end-to-end accountability.
- "Using the trading_partner table" on page 36 identifies how the product maintains data in the provided trading_partner relational table, which maintains the complete list of trading partners.
- "Polling UCCnet for worklists" on page 37 describes how the product obtains worklists from UCCnet.
- "Using subdiagrams" on page 38 details the logic behind the subdiagrams contained in a UCCnet_processWorklist collaboration object.
- "Sending email" on page 43 describes how product collaboration objects alert email recipients of processing errors, and how subdiagrams within a UCCnet_processWorklist collaboration object process email for different processing circumstances.
- "Logging" on page 47 describes the capabilities of various collaboration objects to log errors.
- "Tracing" on page 47 outlines how problems that might occur in the solution workflow can be traced and identified.
- "Use cases" on page 47 details the actors and courses of the use cases associated with the product.

Definitions of connector and adapter

The term ″connector″ used throughout this document refers to the runtime portion of an adapter. References to specific connectors are related to specific adapters, as follows:

- ″iSoftConnector″ refers specifically to the runtime component of an Adapter for iSoft.
- ″JTextConnector″ refers specifically to the runtime component of an Adapter for JText.
- ″JTextISoftConnector″ refers specifically to the runtime component of an adapter based on the Adapter for JText.
- ″JTextRWLConnector″ refers specifically to the runtime component of an adapter based on the Adapter for JText.
- ″SAPConnector″ refers specifically to the runtime component of an I Adapter for mySAP.com (SAP R/3 V. 4.X).

If you are exchanging messages with UCCnet through an AS2/EDIINT interface protocol, use an iSoftConnector. If you are exchanging messages through the UCCnet Command Line Utility (CLU) or testing your installation, use a JTextISoftConnector. ″AS2 channel connector″ used throughout can refer to an iSoftConnector, JTextISoftConnector, depending on the protocol used to exchange messages.

Processing a business object: example workflows (DTD support)

The information in the following sections outlines at a high level how the WebSphere Business Integration Express for Item Synchronization product handles the following workflows, which support the DTD-based implementations:

- ItemAdd — This workflow adds a new item to UCCnet. It is described in the section "ItemAdd workflow: adding a new item to UCCnet (DTD support)."
- ItemPublicationAdd — This workflow makes a new item available to trading partners and processes their responses to it. It is described in the section "ItemPublicationAdd workflow: making a new item available to trading partners and processing their responses to it" on page 7. (ItemAdd and ItemPublicationAdd workflows normally occur sequentially.)
- ItemChange — This workflow updates item information in UCCnet. It is described in the section "ItemChange workflow: updating item information in UCCnet (DTD support)" on page 11.
- ItemPublicationChange — This workflow makes an updated item available to trading partners and processes their responses to it. It is described in the section "ItemPublicationChange workflow: making updated item information available to trading partners and processing their responses to it" on page 14. (ItemChange and ItemPublicationChange workflows normally occur sequentially.)
- ItemDelist — This workflow makes an item permanently unavailable to trading partners. It is described in the section "ItemDelist workflow: making an item permanently unavailable to trading partners (DTD support)" on page 19.
- ItemWithdrawal — This workflow makes an item temporarily unavailable to all or selected trading partners. It is described in the section "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (DTD support)" on page 20.

Refer to the Installation Guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

UCCnet_ItemSync, UCCnet_requestWorklist, UCCnet_processWorklist, and Notify_by_eMail collaboration objects log error messages if they encounter error situations during any stage of processing. See the section "Logging" on page 47 for detailed information. Tracing can also be enabled for all collaboration objects to record logical flaws and data processed. See the section "Tracing" on page 47 for detailed information.

Note: "AS2 channel server" used throughout this document, refers to the iSoft Peer-to-Peer Agent.

ItemAdd workflow: adding a new item to UCCnet (DTD support)

The description below shows how high-level components of the Item Synchronization perform the ItemAdd workflow. In the ItemAdd workflow, a new item is added to UCCnet. The source of the flow is the creation of a new item in the source ERP application. This workflow does not result in notifications being sent to subscribed demand-side trading partners. Another workflow – ItemPublicationAdd detailed in the section "ItemPublicationAdd workflow: making a new item available to trading partners and processing their responses to it" on page 7 — accomplishes sending these notifications.

Notes:

1. The mappings used in processing ItemAdd messages use value translation tables. The InterChange Server Express implements these tables as cross-references.

2. If you are using schema support, refer to the documentation found in "ItemAdd workflow: adding a new item to UCCnet (schema support)" on page 22
   1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
   2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with a *Create* verb by passing it through the Sa4CwItemBasic input map.
   3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
   4. The UCCnet_ItemSync collaboration object adds an entry for the new item to the PROCESSED_GTIN table, setting the value for the **withdrawn** field for this entry to `N`. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
   5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemAdd transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
   6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into an application-specific business object, a UCCnetDTD_envelope business object. The ItemBasic business object is converted by passing through the RouterMap_CwItemBasic_to_UCCnetDTD_envelope router map and CwItemBasic_to_UCCnetDTD_envelope_documentCommand_item translation map.
   7. .
   8. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemAdd XML message in UCCnet format.
   9. The AS2 channel connector passes this message to the AS2 channel server.
   10. The AS2 channel server creates the digest, encrypts, and transmits the ItemAdd message to UCCnet.
   11. UCCnet generates and returns to the AS2 channel server a Message Disposition Notification (MDN) to indicate successful receipt of the ItemAdd message.
   12. The AS2 channel server delivers the MDN to the AS2 channel connector.
   13. The AS2 channel connector sends the MDN to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.
   14. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
   15. The UCCnet_processWorklist collaboration object receives the business object on its *From* port, identifies it as an MDN, and dispatches it to its

SIMPLE_RESPONSE subdiagram. See the section "SIMPLE_RESPONSE subdiagram" on page 43 for more information on this subdiagram.

16. The SIMPLE_RESPONSE subdiagram sends email to the recipients configured in the UCCnet_processWorklist_SIMPLE_RESPONSEObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
17. UCCnet creates a worklist containing the notification response for a successful ItemAdd.
18. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
19. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
20. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope map.
21. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.
22. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
23. The AS2 channel connector passes the message to the AS2 channel server.
24. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
25. UCCnet delivers the worklist containing the notification response for a successful ItemAdd to the AS2 channel server.
26. The AS2 channel server delivers the notification to the AS2 channel connector.
27. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
28. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.

29. The UCCnet_processWorklist collaboration object receives the business object on its *From* port, identifies it as an ItemAdd notification, and dispatches it to its ITEM_ADD_CHANGE subdiagram.

As a result of the ItemAdd workflow, UCCnet has been updated with the new item information. Now, the supplier's demand-side trading partners must be made aware that the item is available. This on-going workflow, referred to as the ItemPublicationAdd workflow, is continued in the section "ItemPublicationAdd workflow: making a new item available to trading partners and processing their responses to it."

ItemPublicationAdd workflow: making a new item available to trading partners and processing their responses to it

The information in this section describes how the high-level components of the Item Synchronization Collaboration perform the ItemPublicationAdd workflow. In the ItemPublicationAdd workflow, a new item that was passed to UCCnet through the ItemAdd workflow (detailed in the section "ItemAdd workflow: adding a new item to UCCnet (DTD support)" on page 4) is made available to the supplier's demand-side trading partners. The demand-side trading partners' responses to the new item are processed as well. As a result, the ItemPublicationAdd workflow is described as two subflows:

- The subflow that makes the new item available to the supplier's demand-side trading partners, described in the section "ItemPublicationAdd subflow 1: making a new item available to trading partners."
- The subflow that processes the demand-side trading partners' responses to the new item, described in the section "ItemPublicationAdd subflow 2: processing trading partners' responses to a new item" on page 9.

ItemPublicationAdd subflow 1: making a new item available to trading partners

The description below describes how the ItemPublicationAdd workflow makes a new item available to a supplier's demand-side trading partners. At this point in processing, the ItemAdd workflow has completed and a UCCnetGBO_envelope business object has arrived in the ITEM_ADD_CHANGE subdiagram of the UCCnet_processWorklist collaboration object.

1. The UCCnet_processWorklist collaboration object's ITEM_ADD_CHANGE subdiagram does the following:
   a. Maps the UCCnetGBO_envelope business object into a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through either the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope or UCCnetGBO_envelope_notification_to_UCCnetDTD_envelope_ publishCommand router and input maps. The business object contains the corresponding ItemPublicationAdd request. This message includes a request for UCCnet to publish the item to the trading partners listed in the message.
   b. Sends the ItemPublicationAdd request over the *ITEM_ADD_CHANGE* port to the AS2 channel connector.
   c. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.

   See the section "ITEM_ADD_CHANGE subdiagram" on page 40 for more information on this subdiagram.

2. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemPublicationAdd XML message in UCCnet format.
3. The AS2 channel connector passes the message to the AS2 channel server.
4. The AS2 channel server creates the digest, encrypts, and transmits the ItemPublicationAdd message to UCCnet.
5. UCCnet generates and sends to the AS2 channel server an MDN indicating successful receipt of the ItemPublicationAdd message.
6. The AS2 channel server delivers the MDN to the AS2 channel connector.
7. The AS2 channel connector sends the MDN to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.
8. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
9. The UCCnet_processWorklist collaboration object receives the business object on its *From* port, identifies it as an MDN, and dispatches it to its SIMPLE_RESPONSE subdiagram. See the section "SIMPLE_RESPONSE subdiagram" on page 43 for more information on this subdiagram.
10. The SIMPLE_RESPONSE subdiagram sends email to the recipients configured in the UCCnet_processWorklist_SIMPLE_RESPONSEObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
11. UCCnet performs a compliance check on the data and, if all data exists in the appropriate formats, creates a worklist containing a PUB_RELEASE_NEW_ITEM notification to indicate a successful ItemPublicationAdd.
12. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
13. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
14. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
15. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the

RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.

16. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
17. The AS2 channel connector passes the message to the AS2 channel server.
18. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
19. UCCnet delivers the worklist containing the PUB_RELEASE_NEW_ITEM notification indicating a successful ItemPublicationAdd to the AS2 channel server.
20. The AS2 channel server delivers the notification to the AS2 channel connector.
21. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
22. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
23. The UCCnet_processWorklist collaboration object receives the object on its *From* port, identifies it as a PUB_RELEASE_NEW_ITEM, and dispatches it to its NEW_ITEM_PUBLICATION_REQUEST subdiagram. See the section "NEW_ITEM_PUBLICATION_REQUEST subdiagram" on page 39 for more information on this subdiagram.

As a result of this subflow of the ItemPublicationAdd workflow, a new item is made available to a supplier's demand-side trading partners. The on-going subflow that alerts them of the item's existence and processes their responses to it, referred to as the ItemPublicationAdd subflow 2, is continued in the section "ItemPublicationAdd subflow 2: processing trading partners' responses to a new item."

ItemPublicationAdd subflow 2: processing trading partners' responses to a new item

The description below describes how the ItemPublicationAdd workflow alerts demand-side trading partners that a new item is available and processes their responses to it.

At this point in processing, the first subflow of the ItemPublicationAdd workflow has completed and a UCCnetGBO_envelope business object has arrived in the NEW_ITEM_PUBLICATION_REQUEST subdiagram of the UCCnet_processWorklist collaboration object.

1. The UCCnet_processWorkflow collaboration object's NEW_ITEM_PUBLICATION_REQUEST subdiagram logic does the following:
   a. Verifies that the GTIN value associated with the item is in the PROCESSED_GTIN table and that the item is not withdrawn. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
   b. Checks that the new item to be published is supplied by a trading partner listed in the trading_partner table and verifies that the demand-side trading partners to whom notification will be sent are also in this table. See the section "Using the trading_partner table" on page 36 for more information on the trading_partner table.

c. Sends the ItemPublicationAdd for the GTIN to the demand-side trading partners identified in the business object. It accomplishes this by mapping the business object into a UCCnetDTD_envelope business object. The business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_notification_to_UCCnetDTD_envelope_ publishCommand maps. The message is sent over the *NEW_ITEM_PUBLICATION_REQUEST* port to the AS2 channel connector, from the AS2 channel connector to the AS2 channel server, and then to UCCnet.

d. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.

2. The trading partners can respond with any of the following responses:
   - AUTHORIZE — The product information has been integrated into the demand-side user's legacy environment and the demand-side user is ready to begin trading.
   - PEND_PUBLICATION — The demand-side user is unsure about the proper action to take on the product. The product is being studied, but no action is possible at this time.
   - REJECT_PUBLICATION — The demand-side user has no interest in the product.
   - PRE_AUTHORIZATION — The demand-side user wants to begin the process of integrating the product into its legacy environment. This response might indicate that the supplier contact the demand-side user to begin the process of deciding on order quantities, pricing specifics, etc.
   - DE_AUTHORIZATION — The demand-side user has removed the product from the assortment and wants no further updates sent for the product.

   For this example, assume a demand-side trading partner responded with an AUTHORIZE response. UCCnet creates a worklist containing this notification response and delivers it to the AS2 channel server.
3. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

   Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

   The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
4. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
5. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
6. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The

UCCnetGBO_envelope business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.

7. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
8. The AS2 channel connector passes the message to the AS2 channel server.
9. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
10. UCCnet performs a compliance check on the data and, if all data exists in the appropriate formats, generates an AUTHORIZE notification to indicate a successful receipt. This notification is returned to the AS2 channel server.
11. The AS2 channel server delivers the notification to the AS2 channel connector.
12. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
13. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
14. The UCCnet_processWorklist collaboration object receives the object on its *From* port, identifies it as an AUTHORIZE response, and dispatches it to its AUTHORIZATION_RESPONSES subdiagram. See the section "AUTHORIZATION_RESPONSES subdiagram" on page 41 for more information on this subdiagram.
15. The AUTHORIZATION_RESPONSES subdiagram does the following:
    a. Sends email to the recipients configured in the UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
    b. Logs the event in the audit_log table. See "Using the audit_log table" on page 35 for more information on the audit_log table.

ItemChange workflow: updating item information in UCCnet (DTD support)

The description below shows how high-level components of the Item Synchronization perform the ItemChange workflow. The ItemChange workflow sends updated information about an existing item to UCCnet. The source of the flow is a change to the data of an existing item in the ERP source application. Issuing a change does not result in notifications being sent to subscribed demand-side trading partners. Another workflow – ItemPublicationChange detailed in the section "ItemPublicationChange workflow: making updated item information available to trading partners and processing their responses to it" on page 14 — accomplishes sending these notifications.

Notes:

1. The mappings used in processing ItemChange messages use value translation tables. The InterChange Server Express implements these tables as cross-references.

2. If you are using schema support, refer to the documentation found in "ItemChange workflow: updating item information in UCCnet (schema support)" on page 26

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with a *Update* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object checks if the item exists in the PROCESSED_GTIN table and processes it, as follows:
   - If the item exists in the table and the value for its **withdrawn** field is set to `N`, the collaboration object continues processing it.
   - If the item exists in the table and the value for its **withdrawn** field is set to `Y`, the collaboration object does the following:
     - Changes the value of the entry's **withdrawn** field to `N`.
     - Changes the value of the entry's **delete** field to `U`.
     - Changes the business object verb to *UNWITHDRAWN*.
     - Continues processing it.
   - If the item does not exist in the table, the collaboration object changes the verb to *Create* and adds it to the PROCESSED_GTIN table, setting the entry's **withdrawn** field to `N`.

   Assume for this example that the item already exists in the table and is not withdrawn. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemChange transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into a UCCnetDTD_envelope. The ItemBasic business object is converted by passing through the RouterMap_CwItemBasic_to_UCCnetDTD_envelope and CwItemBasic_to_UCCnetDTD_envelope_documentCommand_item router and translation maps.
7. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemChange XML message in UCCnet format.
8. The AS2 channel connector passes this message to the AS2 channel server.
9. The AS2 channel server creates the digest, encrypts, and transmits the ItemChange message to UCCnet.

10. UCCnet generates and returns to the AS2 channel server an MDN to indicate successful receipt of the ItemChange message.
11. The AS2 channel server delivers the MDN to the AS2 channel connector.
12. The AS2 channel connector sends the MDN to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.
13. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope.
14. The UCCnet_processWorklist collaboration object receives the business object on its *From* port, identifies it as an MDN, and dispatches it to its SIMPLE_RESPONSE subdiagram. See the section "SIMPLE_RESPONSE subdiagram" on page 43 for more information on this subdiagram.
15. The SIMPLE_RESPONSE subdiagram sends email to the recipients configured in the UCCnet_processWorklist_SIMPLE_RESPONSEObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
16. UCCnet creates a worklist containing the notification response ItemChange and delivers it to the AS2 channel server.
17. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
18. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
19. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
20. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.
21. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
22. The AS2 channel connector passes the message to the AS2 channel server.
23. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.

24. An ItemChange notification is generated by UCCnet to indicate a successful ItemChange. This notification is returned to the AS2 channel server.
25. The AS2 channel server delivers the notification to the AS2 channel connector.
26. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
27. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
28. The UCCnet_processWorklist collaboration receives the business object on its *From* port, identifies it as an ItemChange notification, and dispatches it to its ITEM_ADD_CHANGE subdiagram.

As a result of the ItemChange workflow, UCCnet has updated the item information. Now, the supplier's demand-side trading partners must be notified that updated information about the item exists. This ongoing workflow, referred to as the ItemPublicationChange workflow, is continued in the section "ItemPublicationChange workflow: making updated item information available to trading partners and processing their responses to it."

ItemPublicationChange workflow: making updated item information available to trading partners and processing their responses to it

The information in this section describes how the high-level components of the Item Synchronization perform the ItemPublicationChange workflow. In the ItemPublicationChange workflow, updated item information that was passed to UCCnet through the ItemChange workflow (detailed in the section "ItemChange workflow: updating item information in UCCnet (DTD support)" on page 11) is made available to the supplier's demand-side trading partners. The demand-side trading partners' responses to this item information must then be processed. As a result, the ItemPublicationChange workflow is described as two subflows:

- The subflow that makes updated item information available to the supplier's demand-side trading partners, described in the section "ItemPublicationChange subflow 1: making updated item information available to trading partners."
- The subflow that processes the demand-side trading partners' responses to the updated information, described in the section "ItemPublicationChange subflow 2: processing trading partners' responses to updated item information" on page 16.

ItemPublicationChange subflow 1: making updated item information available to trading partners

The description below describes how the ItemPublicationChange workflow makes updated item information available to a supplier's demand-side trading partners. At this point in processing, the ItemChange workflow has completed and a UCCnetGBO_envelope business object has arrived in the ITEM_ADD_CHANGE subdiagram of the UCCnet_processWorklist collaboration object.

1. The UCCnet_processWorklist collaboration object's ITEM_ADD_CHANGE subdiagram does the following:
   a. Maps the UCCnetGBO_envelope business object into a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the

RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_notification_to_UCCnetDTD_envelope_publishCommand router and transformation maps. The business object contains the corresponding ItemPublicationAdd request. This message includes a request for UCCnet to publish the item to the trading partners listed in the message.

b. Sends the ItemPublicationChange request over the *ITEM_ADD_CHANGE* port to the AS2 channel connector.

c. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.

See the section "ITEM_ADD_CHANGE subdiagram" on page 40 for more information on this subdiagram.

2. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemPublicationChange XML message in UCCnet format.
3. The AS2 channel connector passes the message to the AS2 channel server.
4. The AS2 channel server creates the digest, encrypts, and transmits the ItemPublicationChange message to UCCnet.
5. UCCnet generates and sends to the AS2 channel server an MDN indicating successful receipt of the ItemPublicationChange message.
6. The AS2 channel server delivers the MDN to the AS2 channel connector.
7. The AS2 channel connector sends the MDN to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object.
8. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
9. The UCCnet_processWorklist collaboration object receives the business object on its *From* port, identifies it as an MDN, and dispatches it to its SIMPLE_RESPONSE subdiagram. See the section "SIMPLE_RESPONSE subdiagram" on page 43 for more information on this subdiagram.
10. The SIMPLE_RESPONSE subdiagram sends email to the recipients configured in the UCCnet_processWorklist_SIMPLE_RESPONSEObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
11. UCCnet creates a worklist containing the notification response PUB_RELEASE_DATA_CHANGE and delivers it to the AS2 channel server.
12. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications from the following directory (dependent on platform) that is created during installation of the product
13. to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.

14. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
15. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
16. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.
17. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
18. The AS2 channel connector passes the message to the AS2 channel server.
19. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
20. UCCnet performs a compliance check on the data and, if all data exists in the appropriate formats, generates a PUB_RELEASE_DATA_CHANGE notification to indicate a successful ItemPublicationChange. This notification is returned to the AS2 channel server.
21. The AS2 channel server delivers the notification to the AS2 channel connector.
22. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
23. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
24. The UCCnet_processWorklist collaboration object receives the object on its *From* port, identifies it as a PUB_RELEASE_DATA_CHANGE, and dispatches it to its NEW_ITEM_PUBLICATION_REQUEST subdiagram. See the section "NEW_ITEM_PUBLICATION_REQUEST subdiagram" on page 39 for more information on this subdiagram.

As a result of this subflow of the ItemPublicationChange workflow, updated item information is made available to a supplier's demand-side trading partners. The on-going subflow that alerts them of the available updated information and processes their responses to it, referred to as the ItemPublicationChange subflow 2, is continued in the section "ItemPublicationChange subflow 2: processing trading partners' responses to updated item information."

ItemPublicationChange subflow 2: processing trading partners' responses to updated item information

The description below describes how the ItemPublicationChange workflow alerts demand-side trading partners that new item information is available and processes their responses to it.

At this point in processing, the first subflow of the ItemPublicationChange workflow has completed and a UCCnetGBO_envelope business object has arrived in the NEW_ITEM_PUBLICATION_REQUEST subdiagram of the UCCnet_processWorklist collaboration object.

1. The UCCnet_processWorkflow collaboration object's NEW_ITEM_PUBLICATION_REQUEST subdiagram logic does the following:
   a. Verifies that the GTIN value associated with the item is in the PROCESSED_GTIN table and that the item is not withdrawn. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
   b. Checks that the new item information to be published is supplied by a trading partner listed in the trading_partner table and verifies that the demand-side trading partners to whom notification will be sent are also in this table. See the section "Using the trading_partner table" on page 36 for more information on the trading_partner table.
   c. Sends the ItemPublicationChange for the GTIN to the demand-side trading partners identified in the business object. It accomplishes this by mapping the business object into a UCCnetDTD_envelope business object. The business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_notification_to_UCCnetDTD_envelope_ publishCommand router and transformation maps. The message is sent over the *NEW_ITEM_PUBLICATION_REQUEST* port to the AS2 channel connector, from the AS2 channel connector to the AS2 channel server, and then to UCCnet.
   d. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.
2. The trading partners can respond with any of the following responses:
   - AUTHORIZE — The product information has been integrated into the demand-side user's legacy environment and the demand-side user is ready to begin trading.
   - PEND_PUBLICATION — The demand-side user is unsure about the proper action to take on the product. The product is being studied, but no action is possible at this time.
   - REJECT_PUBLICATION — The demand-side user has no interest in the product.
   - PRE_AUTHORIZATION — The demand-side user wants to begin the process of integrating the product into its legacy environment. This response might indicate that the supplier contact the demand-side user to begin the process of deciding on order quantities, pricing specifics, etc.
   - DE_AUTHORIZATION — The demand-side user has removed the product from the assortment and wants no further updates sent for the product.

   For this example, assume a demand-side trading partner responded with an AUTHORIZE response. UCCnet creates a worklist containing this notification response and delivers it to the AS2 channel server.
3. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

   Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.

4. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
5. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
6. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The UCCnetGBO_envelope business object is converted by passing through the RouterMap_UCCnetGBO_envelope_to_UCCnetDTD_envelope and UCCnetGBO_envelope_to_UCCnetDTD_envelope router and translation maps.
7. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
8. The AS2 channel connector passes the message to the AS2 channel server.
9. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
10. UCCnet performs a compliance check on the data and, if all data exists in the appropriate formats, generates an AUTHORIZE notification to indicate a successful receipt. This notification is returned to the AS2 channel server.
11. The AS2 channel server delivers the notification to the AS2 channel connector.
12. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetDTD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
13. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetDTD_envelope_to_UCCnetGBO_envelope input map.
14. The UCCnet_processWorklist collaboration object receives the object on its *From* port, identifies it as an AUTHORIZE response, and dispatches it to its AUTHORIZATION_RESPONSES subdiagram. See the section "AUTHORIZATION_RESPONSES subdiagram" on page 41 for more information on this subdiagram.
15. The AUTHORIZATION_RESPONSES subdiagram does the following:
    a. Sends email to the recipients configured in the UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject collaboration object (an instance of the Notify_by_eMail collaboration template). See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
    b. Logs the event in the audit_log table. See "Using the audit_log table" on page 35 for more information on the audit_log table.

ItemDelist workflow: making an item permanently unavailable to trading partners (DTD support)

The description below shows how high-level components of the Item Synchronization perform the ItemDelist workflow. The ItemDelist workflow requests that UCCnet make an item in the repository permanently unavailable. After an item has been delisted, it cannot be returned to active trading. (To remove an item from active trading only temporarily, issue an ItemWithdrawal, as discussed in the section "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (DTD support)" on page 20.) The source of the flow is the delist of an existing item in the ERP source application. This workflow does not result in notifications being sent to demand-side trading partners.

Note: If you are using schema support, refer to the documentation found in "ItemDelist workflow: making an item permanently unavailable to trading partners (schema support)" on page 30

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with an *Delist* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object removes the item from the PROCESSED_GTIN table. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemDelist transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted to a UCCnetDTD_envelope business object. The ItemBasic business object is converted by passing through the RouterMap_CwItemBasic_to_UCCnetDTD_envelope and CwItemBasic_to_UCCnetDTD_envelope_publishCommand_documentIdentifier router and translation maps.
7. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemDelist XML message in UCCnet format.
8. The AS2 channel connector passes this message to the AS2 channel server.
9. The AS2 channel server creates the digest, encrypts, and transmits the ItemDelist message to UCCnet.
10. UCCnet generates an MDN to indicate successful receipt of the ItemDelist message.

As a result of the ItemDelist workflow, the item has been permanently delisted in UCCnet and removed from the PROCESSED_GTIN table.

ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (DTD support)

The description below shows how high-level components of the Item Synchronization perform the ItemWithdrawal workflow. The ItemWithdrawal workflow requests that UCCnet make an item temporarily unavailable to all or selected trading partners. An item might be temporarily removed, for instance, if it is out of season or not in production. It might also be made available only to a specific set of demand-side trading partners as a special order item. (To remove an item from active trading permanently, issue an ItemDelist, as discussed in the section "ItemDelist workflow: making an item permanently unavailable to trading partners (DTD support)" on page 19.) The source of the flow is the withdrawal of an existing item in the ERP source application. This workflow does not result in notifications being sent to demand-side trading partners.

Note: If you are using schema support, refer to the documentation found in "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (schema support)" on page 32

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with an *Withdraw* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object locates the item in the PROCESSED_GTIN table and sets the value for the **withdrawn** field to `Y`. This action prevents the publication of the item in response to an incoming INITIAL_ITEM_LOAD_REQUEST. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemWithdrawal transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into a UCCnetDTD_envelope business object. The ItemBasic business object is converted by passing through the RouterMap_CwItemBasic_to_UCCnetDTD_envelope and CwItemBasic_to_UCCnetDTD_envelope_publishCommand_ documentIdentifier router and translation maps.

7. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the ItemWithdrawal XML message in UCCnet format.
8. The AS2 channel connector passes this message to the AS2 channel server.
9. The AS2 channel server creates the digest, encrypts, and transmits the ItemWithdrawal message to UCCnet.
10. UCCnet generates and returns to the AS2 channel server an MDN to indicate successful receipt of the ItemWithdrawal message.

As a result of the ItemWithdrawal workflow, the item has been temporarily withdrawn from UCCnet and has been indicated as withdrawn in the PROCESSED_GTIN table.

Processing a business object: example workflows (Schema support)

The information in the following sections outlines at a high level how the IBM WebSphere Business Integration Express for Item Synchronization product handles the following workflows, which support the schema-based implementations:

- ItemAdd — This workflow adds a new item to UCCnet. It is described in the section "ItemAdd workflow: adding a new item to UCCnet (schema support)" on page 22.
- CatalogueItemNotification_Add — This workflow makes a new item available to trading partners and processes their responses to it. It is described in the section "CatalogueItemNotification_Add workflow" on page 24. (ItemAdd and CatalogueItemNotification_Add workflows normally occur sequentially.)
- ItemChange — This workflow updates item information in UCCnet. It is described in the section "ItemChange workflow: updating item information in UCCnet (schema support)" on page 26.
- CatalogueItemNotification_Change — This workflow makes an updated item available to trading partners and processes their responses to it. It is described in the section "CatalogueItemNotification_Change workflow: making updated item information available to trading partners and processing their responses to it" on page 28. (ItemChange and CatalogueItemNotification_Change workflows normally occur sequentially.)
- ItemDelist — This workflow makes an item permanently unavailable to trading partners. It is described in the section "ItemDelist workflow: making an item permanently unavailable to trading partners (schema support)" on page 30.
- ItemWithdrawal — This workflow makes an item temporarily unavailable to all or selected trading partners. It is described in the section "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (schema support)" on page 32.

Refer to the Installation Guide for detailed information on creating port connections between collaboration objects and between collaboration objects and connectors.

UCCnet_ItemSync, UCCnet_requestWorklist, UCCnet_processWorklist, and Notify_by_eMail collaboration objects log error messages if they encounter error situations during any stage of processing. See the section "Logging" on page 47 for detailed information. Tracing can also be enabled for all collaboration objects to record logical flaws and data processed. See the section "Tracing" on page 47 for detailed information.

Note: "AS2 channel server" used throughout this document refers to the iSoft Peer-to-Peer Agent.

ItemAdd workflow: adding a new item to UCCnet (schema support)

The description below shows how high-level components of the Item Synchronization perform the ItemAdd workflow. In the ItemAdd workflow, a new item is added to UCCnet. The source of the flow is the creation of a new item in the source ERP application. This workflow does not result in notifications being sent to subscribed demand-side trading partners. Another workflow – CatalogueItemNotification_Add, detailed in the section "CatalogueItemNotification_Add workflow" on page 24 — accomplishes sending these notifications.

Notes:

1. The mappings used in processing ItemAdd messages use value translation tables. The Interchance Server Express implements these tables as cross-references.
2. If you are not using schema support, refer to the documentation found in "ItemAdd workflow: adding a new item to UCCnet (DTD support)" on page 4.

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with a *Create* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object adds an entry for the new item to the PROCESSED_GTIN table, setting the value for the **withdrawn** field for this entry to `N`. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemAdd transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the ItemCommandRouter collaboration object.
7. The ItemCommandRouter collaboration object received the business object and determines that it represents an item to be added to the UCCnet, and passes the business object to the TO_RCIR port.
8. The ItemBasic business object is converted into an application-specific business object by the CwItemBasic_to_UCCnetXSD_envelope_registerCommand_itemAddChange map.
9. Once the conversion is complete, the business object is passed to UCCnet through the AS2 channel connector.
10. The connector send the business object to the Data Handler for XML, which produces the ItemAdd XML message in UCCnet format.

11. The connector then passes the message to the AS2 channel server.
12. The server created the digest, encrypts, and then transmits the ItemAdd message to UCCnet.
13. UCCnet creates a worklist containing the notification response RCIR_ADD_Response and delivers it to the AS2 channel server.
14. A time-triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications and send them to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
15. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetXSD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
16. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope by passing it the UCCnetXSD_envelope_to_UCCnetGBO_envelope.
17. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its *From* port and passes it to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into a UCCnetXSD_envelope. The UCCnetGBO_envelope business object is converted by passing through the UCCnetGBO_envelope_to_UCCnetXSD_envelope translation map.
18. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the SML message in UCCnet format.
19. The AS2 channel connector passes the message to the AS2 channel server.
20. The AS2 channel server creates the digest, and encrypts and transmits the message to UCCnet.
21. An RCIR_ADD_Response notification is generated by UCCnet to indicate a successful ItemAdd. This notification is returned to the AS2 channel server.
22. The AS2 channel server delivered the notification to the AS2 channel connector.
23. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetXSD_envelope business object. The business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
24. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
25. The UCCnet_processWorklist collaboration object receives the business object on its From port, identifies it as an RCIR_ADD_RESPONSE notification, and dispatches it to its RCIR_ADD_RESPONSE subdiagram.

As a result of the ItemAdd workflow, UCCnet has been updated with the new item information. Now, the supplier's demand-side trading partners must be made aware that the item is available. This on-going workflow, referred to as the CatalogueItemNotification_Add workflow, is continued in the section "CatalogueItemNotification_Add workflow."

CatalogueItemNotification_Add workflow

The information in this section describes how the high-level components of the Item Synchronization perform the CatalogueItemNotification_Add workflow. In the CatalogueItemNotification_Add workflow, a new item that was passed to UCCnet through the ItemAdd workflow (detailed in the section "ItemAdd workflow: adding a new item to UCCnet (schema support)" on page 22) is made available to the supplier's demand-side trading partners. The demand-side trading partners' responses to the new item are processed as well.

1. The UCCnet_processWorklist collaboration object receives a UCCnetGBO_envelope business object on its From port, identifies it as an RCIR_ADD_RESPONSE notification, and dispatches it to its RCIR_ADD_RESPONSE subdiagram.
2. The RCIR_ADD_RESPONSE subdiagram does the following:
   a. Retrieves the ItemBasic business object for the received message.
   b. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.
   c. Sends the ItemBasic business object out through its CIN_DISPATCHER port to the UCCnetXSD_CIN_Dispatcher collaboration object.

   The UCCnetXSD_CIN_Dispatcher collaboration object receives the ItemBasic business object on its From port
3. It maps the ItemBasic business object to a UCCnetXSD_envelope business object using the map defined in its collaboration parameters.
4. The collaboration object uses the category code from the new UCCnetXSD_envelope to retrieve the GLNs of any trading partners that subscribe to the category found in the CIN_DISPATCHER_GLN_FILE property.
5. The collaboration object sends a CatalogueItemNotification_ADD (CIN_ADD) business object out to the AS2 connector for each GLN found in the GLN subscription file.
6. At this point, the trading partners can respond with any of the following Catalogue Item Confirmation responses:

   REVIEW
   This state is used to tell parties that an item is being reviewed by the retailer.

   REJECTED
   This state is used to tell parties that an item is rejected and that no additional information is requested at this time.

   ACCEPTED
   This state is used to tell initiators that an items has been accepted by the retailed, but has not yet been synchronized. This is similar to the UCCnet Pre-Authorization.

   SYNCHRONISED
   This state is used to tell initiators that an item has been accepted by the retailer and will be synchronized. This is similar to the UCCnet Authorization.

The rest of this example assumes that a demand-side trading partner has responded with a SYNCHRONISED response.

7. UCCnet creates a worklist containing the notification response and delivers it to the AS2 channel server.
8. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

   Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

   The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
9. The JTextRWLConnector received the worklist query command from its `event` directory and sends it to the Data Handler for XML.
10. The Data Handler for XML converts the worklist query command into a UCCnetXSD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the command related to it.
11. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, the business object is first converted to a UCCnetGBO_envelope business object by passing it through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
12. The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its From port and passes it to the AS2 channel connector over its To port. Before the business object arrives at the connector, it is converted to a UCCnetXSD_envelope business object.
13. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
14. The AS2 channel connector passes the message tot the AS2 channel server.
15. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
16. UCCnet performs a compliance check on the data and, if all the data exists in the appropriate format, generates a SYNCHRONISED notification to indicate a successful receipt.
17. The notification is returned to the AS2 channel server.
18. The AS2 channel server delivers the notification to the AS2 channel connector.
19. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetXSD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
20. The AS2 channel connector delivers the business object to the UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing it through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
21. The UCCnet_processWorklist collaboration object receives the object on its From port, identifies it as a CATALOGUE_ITEM_CONFIRMATION response, and dispatches it to its CATALOGUE_ITEM_CONFIRMATION subdiagram.
22. The CATALOGUE_ITEM_CONFIRMATION subdiagram carries out the following tasks:

- It sends an email to the recipients defined by the UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject business object, which is an instance of the Notify_by_eMail collaboration template. See "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information on how to configure properties controlling email.
- It logs the even in the audit_log table. See "Using the audit_log table" on page 35 for more information on the audit log table.

ItemChange workflow: updating item information in UCCnet (schema support)

The description below shows how high-level components of the Item Synchronization perform the ItemChange workflow. The ItemChange workflow sends updated information about an existing item to UCCnet. The source of the flow is a change to the data of an existing item in the ERP source application. Issuing a change does not result in notifications being sent to subscribed demand-side trading partners. Another workflow – CatalogueItemNotification_Change, detailed in the section "CatalogueItemNotification_Change workflow: making updated item information available to trading partners and processing their responses to it" on page 28 — accomplishes sending these notifications.

Notes:

1. The mappings used in processing ItemChange messages use value translation tables. The InterChange Server Express implements these tables as cross-references.
2. If you are not using schema support, refer to the documentation found in "ItemChange workflow: updating item information in UCCnet (DTD support)" on page 11.

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with a *Create* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object checks if the item exists in the PROCESSED_GTIN table and processes it, as follows:
   - If the item exists in the table and the value for its **withdrawn** field is set to `N`, the collaboration object continues processing it.
   - If the item exists in the table and the value for its **withdrawn** field is set to `Y`, the collaboration object does the following:
     – Changes the value of the entry's **withdrawn** field to `N`.
     – Changes the value of the entry's **delete** field to `U`.

- Changes the business object verb to *UNWITHDRAWN*.
- Continues processing it.

• If the item does not exist in the table, the collaboration object changes the verb to *Create* and adds it to the PROCESSED_GTIN table, setting the entry's **withdrawn** field to `N`.

Assume for this example that the item already exists in the table and is not withdrawn. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.

5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemChange transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivered the ItemBasic business object to the ItemCommandRouter collaboration object.
7. The ItemCommandRouter collaboration object determines that the object represents an Item Change and send it to the TO_RCIR port.
8. The ItemBasic business object is converted into an application-specific business object by using the CwItemBasic_to_UCCnetXSD_envelope_registerCommand_itemAddChange map.It is then passed to UCCnet through the AS2 channel.
9. The AS2 channel connector send the business object to the Data Handler for XML, which produces the ItemChange XML message in UCCnet format.
10. The AS2 channel connector passes this message to the AS2 channel server.
11. The AS2 channel server creates the digest, encrypts, and transmits the ItemChange message to UCCnet.
12. UCCnet creates a worklist containing the notification response RCIR_CHANGE_Response and delivers it to the AS2 channel server.
13. A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

    Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.

    The JTextRWLConnector polls its `event` directory for any worklist query commands that might have been delivered. See the section "Polling UCCnet for worklists" on page 37 for more information on this process.
14. The JTextRWLConnector retrieves the worklist query command from its `event` directory and sends it to the Data Handler for XML, which converts it into a UCCnetXSD_envelope. This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.
15. The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
16. The UCCnet_requestWorklist collaboration object received the UCCnetGBO_envelope business object on its From port and passes it to the AS2 channel connector over its To port. Before the business object arrives at the connector it is converted to a UCCnetXSD_envelope by passing it through the UCCnetGBO_envelope_to_UCCnetXSD_envelope map.

17. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
18. The AS2 channel connector passes the message to the AS2 channel server.
19. The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
20. An RCIR_CHANGE_Response notification is generated by UCCnet to indicate a successful ItemChange. This notification is returned to the AS2 channel server.
21. The AS2 channel server delivers the notification to the AS2 channel connector.
22. The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetXSD_envelope application-specific business object.
23. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
24. The UCCnet_processWorklist collaboration object receives the business object on its From port, identifies it as an RCIR_CHANGE_RESPONSE notification, and dispatches it to its RCIR_ADD_RESPONSE subdiagram.

As a result of the ItemChange workflow, UCCnet has been updated with the new item information. Now, the supplier's demand-side trading partners must be notified that the item is available. This ongoing workflow, referred to as the CatalogueItemNotification_Change workflow, is continued in the section "CatalogueItemNotification_Change workflow: making updated item information available to trading partners and processing their responses to it."

CatalogueItemNotification_Change workflow: making updated item information available to trading partners and processing their responses to it

The information in this section describes how the high-level components of the Item Synchronization perform the CatalogueItemNotification_Change workflow. In the CatalogueItemNotification_Change workflow, updated item information that was passed to UCCnet through the ItemChange workflow (detailed in the section "ItemChange workflow: updating item information in UCCnet (schema support)" on page 26) is made available to the supplier's demand-side trading partners. The demand-side trading partners' responses to this item information must then be processed.

- The UCCnet_processWorklist collaboration object receives a UCCnetGBO_envelope business object on its From port, identifies it as an RCIR_ADD_RESPONSE notification, and dispatches it to its RCIR_ADD_RESPONSE subdiagram.
- The UCCnet_processWorklist collaboration object receives a UCCnetGBO_envelope business object on its From port, identifies it as an RCIR_ADD_RESPONSE notification, and dispatches it to its RCIR_ADD_RESPONSE subdiagram.
- The RCIR_ADD_RESPONSE subdiagram does the following:
  1. Retrieves the ItemBasic business object for the received message.
  2. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.
  3. Sends the ItemBasic business object out through its CIN_DISPATCHER port to the UCCnetXSD_CIN_Dispatcher collaboration object.

- The UCCnetXSD_CIN_Dispatcher collaboration object receives the ItemBasic business object on its From port
- It maps the ItemBasic business object to a UCCnetXSD_envelope business object using the map defined in its collaboration parameters.
- The collaboration object uses the category code from the new UCCnetXSD_envelope to retrieve the GLNs of any trading partners that subscribe to the category found in the CIN_DISPATCHER_GLN_FILE property.
- The collaboration object sends a CatalogueItemNotification_CHANGE (CIN_CHANGE) business object out to the AS2 connector for each GLN found in the GLN subscription file.
- At this point, the trading partners can respond with any of the following Catalogue Item Confirmation responses:

  REVIEW
  This state is used to tell parties that an item is being reviewed by the retailer.

  REJECTED
  This state is used to tell parties that an item is rejected and that no additional information is requested at this time.

  ACCEPTED
  This state is used to tell initiators that an items has been accepted by the retailer, but has not yet been synchronized. This is similar to the UCCnet Pre-Authorization.

  SYNCHRONISED
  This state is used to tell initiators that an item has been accepted by the retailer and will be synchronized. This is similar to the UCCnet Authorization.

  The rest of this example assumes that a demand-side trading partner has responded with a SYNCHRONISED response.
- UCCnet creates a worklist containing the notification response and delivers it to the AS2 channel server.
- A chronologically triggered process must be configured to move query command messages tailored to retrieve specific UCCnet notifications to the `event` directory of the JTextRWLConnector:

  Note: This process is not part of the product and must be customized by the user. The installation path is dependent on the path set when the product is installed.
- The JTextRWLConnector polls its `event` directory for any worklist query commands that have been delivered. See the section “Polling UCCnet for worklists” on page 37 for more information on this process.
- The JTextRWLConnector received the worklist query command from its `event` directory and sends it to the Data Handler for XML.
- The Data Handler for XML converts the worklist query command into a UCCnetXSD_envelope business object. This business object contains the entire UCCnet message, including each individual data instance and the command related to it.
- The JTextRWLConnector passes the business object to a UCCnet_requestWorklist collaboration object. However, the business object is first converted to a UCCnetGBO_envelope business object by passing it through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.

- The UCCnet_requestWorklist collaboration object receives the UCCnetGBO_envelope business object on its From port and passes it to the AS2 channel connector over its To port. Before the business object arrives at the connector, it is converted to a UCCnetXSD_envelope business object.
- The AS2 channel connector sends the business object to the Data Handler for XML, which produces the XML message in UCCnet format.
- The AS2 channel connector passes the message to the AS2 channel server.
- The AS2 channel server creates the digest, encrypts, and transmits the message to UCCnet.
- UCCnet performs a compliance check on the data and, if all the data exists in the appropriate format, generates a SYNCHRONISED notification to indicate a successful receipt.
- The notification is returned to the AS2 channel server.
- The AS2 channel server delivers the notification to the AS2 channel connector.
- The AS2 channel connector sends the notification to the Data Handler for XML, which converts it into a UCCnetXSD_envelope business object. This business object contains the entire UCCnet notification, including each individual data instance and the commands related to it.
- The AS2 channel connector delivers the business object to the UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing it through the UCCnetXSD_envelope_to_UCCnetGBO_envelope input map.
- The UCCnet_processWorklist collaboration object receives the object on its From port, identifies it as a CATALOGUE_ITEM_CONFIRMATION response, and dispatches it to its CATALOGUE_ITEM_CONFIRMATION subdiagram
- The CATALOGUE_ITEM_CONFIRMATION subdiagram carries out the following tasks:
  - It sends an email to the recipients defined by the UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject business object, which is an instance of the Notify_by_eMail collaboration template. See “Sending email through UCCnet_processWorklist collaboration object subdiagrams” on page 44 for more information on how to configure properties controlling email.
  - It logs the even in the audit_log table. See “Using the audit_log table” on page 35 for more information on the audit log table.

ItemDelist workflow: making an item permanently unavailable to trading partners (schema support)

The description below shows how high-level components of the Item Synchronization perform the ItemDelist workflow. The ItemDelist workflow requests that UCCnet make an item in the repository permanently unavailable. After an item has been delisted, it cannot be returned to active trading. (To remove an item from active trading only temporarily, issue an ItemWithdrawal, as discussed in the section “ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (schema support)” on page 32.) The source of the flow is the delist of an existing item in the ERP source application. This workflow does not result in notifications being sent to demand-side trading partners.

Note: If you are not using schema support, refer to the documentation found in “ItemDelist workflow: making an item permanently unavailable to trading partners (DTD support)” on page 19

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with an *Delist* verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object removes the item from the PROCESSED_GTIN table. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemDelist transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object sends the ItemBasic business object through its To port to the ItemCommandRouter collaboration object.
7. The ItemCommandRouter receives the business object on its From port, and determines that the business object is being delisted.
8. The ItemCommandRouter sends the ItemBasic business object through its TO_CIN port to the UCCnetXSD_CIN_Dispatcher collaboration object.
9. The UCCnetXSD_CIN_Dispatcher collaboration object receives the ItemBasic business object on its From port.
10. The collaboration object maps the ItemBasic business object to a UCCnetXSD_envelope business object using the map defined by its collaboration parameters.
11. The collaboration object the uses the category code from the new UCCnetXSD_envelope to retrieve the GLNs of any trading partners that have subscribed to the categories found in the CIN_DISPATCHER_GLN_FILE property of the business object.
12. The collaboration object sends a CatalogueItemNotification_ADD (CIN_ADD) business object to the AS2 connector for each GLN found in the GLN subscription file. Before the business objects arrive at the connector, they are converted into UCCnetXSD_envelope business objects.
13. The AS2 channel connector sends each business object to the Data Handler for XML, which produces the CatalogueItemNotification_delist XML message in UCCnet format.
14. The AS2 channel connector passes this message to the AS2 channel server.
15. The AS2 channel server creates the digest, encrypts, and transmits the CatalogueItemNotification_delist message to UCCnet.
16. UCCnet generates an MDN to indicate successful receipt of the CatalogueItemNotification_Delist message.

As a result of the ItemDelist workflow, the item has been permanently delisted in UCCnet and removed from the PROCESSED_GTIN table.

ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (schema support)

The description below shows how high-level components of the Item Synchronization perform the ItemWithdrawal workflow. The ItemWithdrawal workflow requests that UCCnet make an item temporarily unavailable to all or selected trading partners. An item might be temporarily removed, for instance, if it is out of season or not in production. It might also be made available only to a specific set of demand-side trading partners as a special order item. (To remove an item from active trading permanently, issue an ItemDelist, as discussed in the section "ItemDelist workflow: making an item permanently unavailable to trading partners (schema support)" on page 30.) The source of the flow is the withdrawal of an existing item in the ERP source application. This workflow does not result in notifications being sent to demand-side trading partners.

Note: If you are not using schema support, refer to the documentation found in "ItemWithdrawal workflow: making an item temporarily unavailable to all or selected trading partners (DTD support)" on page 20

1. A trigger from the ERP source provides the item (for example, an IDOC from SAP) to the connector portion of an adapter specific to that ERP. In this example, we use the SAPConnector. The SAPConnector converts the input from the ERP into a SAP application-specific business object.
2. The SAPConnector passes the SAP application-specific business object to a UCCnet_ItemSync collaboration object, first transforming it into a generic ItemBasic business object with an Update verb by passing it through the Sa4CwItemBasic input map.
3. The UCCnet_ItemSync collaboration object accepts the object on its *From* port and checks that required fields contain information, as detailed in the section "Checking that item data exists for fields required by UCCnet" on page 33. If all required fields are complete, the collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email to a configured address, as detailed in the section "Alerting email recipients of processing errors" on page 43. For this example, assume all fields are complete.
4. The UCCnet_ItemSync collaboration object changes the **Withdrawn** column in the PROCESSED_GTIN table to a value of `Y`. See the section "Using the PROCESSED_GTIN table" on page 34for more information on the PROCESSED_GTIN table.
5. The UCCnet_ItemSync collaboration object adds an entry to the audit_log table to identify the ItemDelist transaction processed. See the section "Using the audit_log table" on page 35 for more information about the audit_log table.
6. The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the *From* port of the ItemCommandRouter collaboration object by sending it out from its *To* port.
7. The ItemCommandRouter collaboration object uses a combination of the verb and the DeleteFlag field of the ItemBasic business object to determine that the business object is being withdrawn.

8. The ItemCommandRouter collaboration object sends the ItemBasic business object out from its *TO_CIN* port to the *From* port of the UCCnetXSD_CIN_Dispatcher collaboration object.
9. The UCCnetXSD_CIN_Dispatcher collaboration object's logic does the following:
   a. Maps from the incoming ItemBasic business object to a UCCnetXSD_envelope business object using the map configured in the collaboration parameters.
   b. Uses the category code from the new UCCnetXSD_envelope business object to retrieve the GLN's of any trading partners subscribed to the category found in the CIN_DISPATCHER_GLN_FILE, also a configurable collaboration property.
   c. Sends out a CatalogueItemNotification_ADD (CIN_ADD) business object out to the AS2 connector for each GLN found in the GLN subscription file.
10. The UCCnetXSD_CIN_Dispatcher collaboration object sends the CatalogueItemNotification business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into a UCCnetXSD_envelope business obect. The ItemBasic business object is converted by passing through a CwItemBasic_to_UCCnetXSD_envelope_notifyCommand_ catalogueItem map.
11. The AS2 channel connector sends the business object to the Data Handler for XML, which produces the CatalogueItemNotification_Delist XML message in UCCnet format.
12. he AS2 channel connector passes this message to the AS2 channel server.
13. The AS2 channel server creates the digest, encrypts, and transmits the CatalogueItemNotification_Withdraw message to UCCnet.
14. UCCnet generates an MDN to indicate successful receipt of the CatalogueItemNotification_Withdraw message.

As a result of the ItemWithdrawal workflow, the item has been temporarily withdrawn from UCCnet and has been indicated as withdrawn in the PROCESSED_GTIN table.

Checking that item data exists for fields required by UCCnet

UCCnet requires its community of trading partners to provide standardized item data in particular formats to its registry. As a result, UCCnet requires requests for ItemAdd, ItemChange, ItemDelist, and ItemWithdrawal publications to have data provided for certain fields. If the data for the required fields is not present, UCCnet does not process the publications. Data might be missing if the ERP does not require information for these same fields and the ERP user is not aware of the UCCnet requirements.

To help ensure that ItemAdd, ItemChange, ItemDelist, and ItemWithdrawal publications are accepted by UCCnet, when a UCCnet_ItemSync collaboration object accepts an ItemBasic business object, it checks that the following fields that are required by UCCnet to have data contain information (i.e., are not NULL):

- For ItemAdd and ItemChange publications:
  - gtin
  - dimension
  - height
  - volume

  - productHierarchy
  - barCodeId
  - unitOfWgt (if either the grossWeight or netWeight fields contain values)
- For ItemDelist and ItemWithdrawal publications:
  - gtin

Note: The UCCnet_ItemSync collaboration object checks only that the required fields are not NULL. It does not verify that the information within them is in the correct format for UCCnet.

If all required fields are complete, the UCCnet_ItemSync collaboration object continues processing it. If all required fields are not complete, the collaboration object aborts processing and sends an email requesting the missing information to an email address provided in the UCCnet_ItemSync collaboration object's SEND_MAIL_TO configuration property. See the section "Alerting email recipients of processing errors" on page 43 for more information on how email is handled within the product.

Using the PROCESSED_GTIN table

The PROCESSED_GTIN table is a relational table provided with the IBM WebSphere Business Integration Express for Item Synchronization product. It maintains the complete list of the supplier's items that exist in the UCCnet repository by using each item's tracking ID, or GTIN, as the primary key. This table permits a UCCnet_processWorklist collaboration object to process incoming INITIAL_ITEM_LOAD_REQUEST commands without the need to communicate with the back-end ERP system. See the section "INITIAL_ITEM_LOAD_REQUEST subdiagram" on page 40 for more information on this subdiagram.

The UCCnet_ItemSync and UCCnet_processWorklist collaboration objects both interact with this table. A UCCnet_ItemSync collaboration object updates the table each time it processes an ItemBasic business object. The collaboration object performs this processing only if all of the fields for which UCCnet requires data are complete (see the section "Checking that item data exists for fields required by UCCnet" on page 33 for more information). The type of processing the collaboration object performs depends on the verb attached to the ItemBasic business object, as follows:

- If the ItemBasic business object has a *Create* verb, the UCCnet_ItemSync collaboration object checks if the item exists in the PROCESSED_GTIN table and processes it, as follows:
  - If the item does not already exist in the PROCESSED_GTIN table, the collaboration object adds an entry for it to the table, and sets the value for the **withdrawn** field for this entry to `N`.
  - If the item already exists in the table, the collaboration object changes its verb to *Update*.
- If the ItemBasic business object has an *Update* verb, the UCCnet_ItemSync collaboration object checks if the item exists in the PROCESSED_GTIN table and processes it, as follows:
  - If the item exists in the table and the value for its **withdrawn** field is set to `N`, the collaboration object continues processing it.
  - If the item exists in the table and the value for its **withdrawn** field is set to `Y`, the collaboration object does the following:
    - Changes the value of the entry's **withdrawn** field to `N`.

- Changes the value of the entry's **delete** field to `U`.
- Changes the business object verb to *UNWITHDRAWN*.
- Continues processing it.

– If the item does not exist in the table, the collaboration object changes the business object's verb to *Create* and adds it to the PROCESSED_GTIN table, setting the entry's **withdrawn** field to `N`.

- If the ItemBasic business object has a *Delist* verb, the UCCnet_ItemSync collaboration object removes the item from the PROCESSED_GTIN table.
- If the ItemBasic business object has a *Withdraw* verb, the UCCnet_ItemSync collaboration object locates the item in the PROCESSED_GTIN table and sets the value for the entry's **withdrawn** field to `Y`. This action prevents the publication of the item in response to an incoming INITIAL_ITEM_LOAD_REQUEST.

The UCCnet_processWorklist collaboration object reads this table during processing, as follows:

- Its NEW_ITEM_PUBLICATION_REQUEST subdiagram verifies that specific items are in the table.
- Its INITIAL_ITEM_LOAD_REQUEST subdiagram generates publication messages for all items in the table with a withdrawn value of `N`.

The UCCnet_ItemSync collaboration object connects to the database through its GtinDB_USER, GtinDB_PASSWORD, JDBC_DRIVER, and JDBC_URL configuration properties; the UCCnet_processWorklist collaboration object, through its DB_USER, DB_PASSWORD, JDBC_DRIVER, and JDBC_URL configuration properties. See Item Synchronization Collaboration for detailed information on these properties.

Connection information for the PROCESSED_GTIN table is configured as part of the UCCnet_ItemSync collaboration. The GTIN and WITHDRAWN columns are added to the table via running the supplied InitializeRelationshipTables.sql file for the database type (i.e., DB2®, or Microsoft® SQL Server). See the Installation Guide for installation instructions.

Using the audit_log table

The audit_log table is provided with the IBM WebSphere Business Integration Express for Item Synchronization product. It is used to track the events associated with UCCnet activities to support complete end-to-end auditing. This audit support provides irrefutable documentation that transmissions have occurred between trading partners. It also provides a profile of which trading partners are participating in the trading community and with which products that participation is associated.

The audit service is composed of three components:

- An audit_log table that receives log entries from each participant component in the solution. A sample entry includes the following fields. Sample values are included for each:
  – LOG_SEQ_NO — 1
  – LOG_SOURCE_NAME — UCCnet2
  – GLN_CODE — NA
  – SOURCE_SYSTEM
  – PRODUCT_ID — 2050000000454
  – VERB_NAME — Create

  - TRANS_ID — SAPConnector_1015606877187_1
  - TRANS_TYPE — UCCnet_processWorklist
  - TRANS_STATUS — ITEM_ADD_CHANGE
  - MSG_FILEPATH_TEXT — C:\IBM\WebSphereICS\UCCnet-1051628537493.bo
  - LOG_DTTM - May 5, 2003 1:44:56 PM
- A logging framework that is utilized by Collaborations and Adapters to log critical events to the audit_log table.
- A report generation facility to support data analysis and visualization.

UCCnet_ItemSync and UCCnet_processWorklist collaboration objects impact the audit_log table. Any time an item is added to, updated or delisted within, or withdrawn from the ERP, and an ItemBasic business object is subsequently passed to a UCCnet_ItemSync collaboration object, the collaboration object records an entry in the audit_log table detailing the event. A UCCnet_processWorklist collaboration object's NEW_ITEM_PUBLICATION_REQUEST, INITIAL_ITEM_LOAD_REQUEST, ITEM_ADD_CHANGE, and AUTHORIZATION_RESPONSES subdiagrams also record entries in the audit_log table during processing. See the sections "NEW_ITEM_PUBLICATION_REQUEST subdiagram" on page 39, "INITIAL_ITEM_LOAD_REQUEST subdiagram" on page 40, "ITEM_ADD_CHANGE subdiagram" on page 40, and "AUTHORIZATION_RESPONSES subdiagram" on page 41 for more information about how these subdiagrams operate. Each audit entry is associated with the value listed for the collaboration object's SUPPLIER_NAME attribute.

Connection from the UCCnet_ItemSync and UCCnet_processWorklist collaboration objects to the audit_log table is provided by the IBM JDBC Driver for DB2, or Microsoft SQL Server. Table creation is performed via running the supplied audit_log.sql file for the database type (i.e., DB2, or Microsoft SQL Server). See the Installation guide for installation instructions.

Using the trading_partner table

The trading_partner table, or GLN table, is a relational table provided with theIBM WebSphere Business Integration Express for Item Synchronization product. It maintains the complete list of trading partners by using the Global Location Number (GLN) of each as the key.

The NEW_ITEM_PUBLICATION_REQUEST subdiagram of a UCCnet_processWorklist collaboration object checks that a new item or updated item information to be published is supplied by a trading partner from this table. It also verifies that the demand-side trading partners to whom notification will be sent are in the table. The INITIAL_ITEM_LOAD_REQUEST subdiagram of a UCCnet_processWorklist collaboration object checks the trading_partner table to verify that the demand-side trading partner requesting the INITIAL_ITEM_LOAD_REQUEST exists. The ITEM_ADD_CHANGE subdiagram of a UCCnet_processWorklist collaboration object utilizes maps that read from the trading_partner table.

Connection information for the trading_partner table is configured as part of the UCCnet_processWorklist collaboration object. The following columns are added to the table via running the supplied InitializeRelationshipTables.sql file for the database type (i.e., DB2, or Microsoft SQL Server):

- gln_code
- trading_partner_name

- trading_partner_contact
- trading_partner_group
- trading_partner_type
- initial_load_flag

See the Installation guide for installation instructions.

Note: After these columns are in place, the table must be populated manually through methods provided by the individual database.

Polling UCCnet for worklists

UCCnet never spontaneously sends notification messages; it must be polled for them via query command messages tailored to retrieve the specific UCCnet notifications. Specific worklist query commands are supplied with the IBM WebSphere Business Integration Express for Item Synchronization product. The commands include messages to query all notifications, all items, or the first five items.

A chronologically triggered process must be configured to move the query command messages from this directory to the `event` directory of the JTextRWLConnector. This process is not part of the product and must be customized by the user. The JTextRWLConnector polls its event directory for any worklist query commands that might have been delivered at a polling interval, which is set by the user. When the JTextRWLConnector finds a worklist query command, it sends it to the Data Handler for XML, which converts it into a UCCnet*XXX*_envelope business object.

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

This business object contains the entire UCCnet message, including each individual data instance and the commands related to it.

The JTextRWLConnector then passes this business object to a UCCnet_requestWorklist collaboration object, which passes it to the AS2 channel server for transmission to UCCnet. UCCnet responds with the appropriate worklist, which initiates ongoing workflows in the solution.

The DTD_URL and SET_UNIQUE_IDS properties of the UCCnet_requestWorklist collaboration object affect the outgoing XML message in systems using the DTD XML definition type. The DocType line in the XML is set according to the value of the DTD_URL property. If outgoing messages are required to have unique message IDs, the SET_UNIQUE_IDS property must be set to ALL

Both the worklist request XML and the polling interval can be changed. For example, the worklist query command XML message tailored for Authorization Notifications (query type="NOTIFICATION" with name="AUTHORIZATION_INFORMATION" and status="UNREAD") can be used to request the worklist authorization notification contents. A similar request can be constructed to read any dead letter notifications. As an alternative, all notifications can be requested. The polling interval is set in the **PollFrequency** attribute of the JTextRWLConnector and is in milliseconds.

The UCCnet_requestWorklist collaboration supports the notification type="PUBLICATION_INFORMATION" for the topics PEND_PUBLICATION, PRE_AUTHORIZATION, AUTHORIZATION, REJECT_PUBLICATION,

DE_AUTHORIZATION, and in the notifications for NEW_ITEM_PUBLICATION_REQUEST and ITEM_INFORMATION (ITEM_ADD, ITEM_CHANGE).

Using subdiagrams

All messages initiated from UCCnet are in UCCnet XML format. Since the UCCnet XML format's top-level tag (<envelope>) is the same for all messages, a component is needed to distinguish among the various notification and response XML messages and return different business objects for them. An object based on the UCCnet_processWorklist collaboration template performs this task.

A UCCnet_processWorklist collaboration object is instantiated when the AS2 channel connector forwards to it a UCCnetGBO_envelope business object. This business object is created from the following process:

1. The AS2 channel connector receives the UCCnet worklist document (envelope) from the AS2 channel server.
2. The AS2 channel connector sends the document to the Data Handler for XML, which converts it into a UCCnet*XXX*_envelope business object. In these business object names, *XXX* is either `DTD` or `XSD`, depending on whether you are using a DTD XML definition or a schema-based XML definition.
3. The AS2 channel connector delivers the business object to a UCCnet_processWorklist collaboration object. However, first the business object is converted to a UCCnetGBO_envelope business object by passing through an input map of the form UCCnet*XXX*_envelope_to_UCCnetGBO_envelope.

The UCCnet_processWorklist collaboration object parses the UCCnetGBO_envelope business object, creating a separate UCCnetGBO_envelope business object for each notification or response. The collaboration object routes each business object representing a single notification to the appropriate subdiagram. Each subdiagram handles a particular set of notification or response messages, as follows:

- NEW_ITEM_PUBLICATION_REQUEST subdiagram — Handles notifications associated with the single notification topic NEW_ITEM_PUBLICATION_REQUEST. See the section "NEW_ITEM_PUBLICATION_REQUEST subdiagram" on page 39 for more information on this subdiagram.
- INITIAL_ITEM_LOAD_REQUEST subdiagram — Handles notifications associated with the single notification topic INITIAL_ITEM_LOAD_REQUEST. See the section "INITIAL_ITEM_LOAD_REQUEST subdiagram" on page 40 for more information on this subdiagram.
- ITEM_ADD_CHANGE subdiagram — Handles notifications for the ITEM_ADD and ITEM_CHANGE topics. See the section "ITEM_ADD_CHANGE subdiagram" on page 40 for more information on this subdiagram.
- CATEGORY_ADD_CHANGE subdiagram — Handles notifications for the CATEGORY_ADD and CATEGORY_CHANGE topics. See the section "CATEGORY_ADD_CHANGE subdiagram" on page 41 for more information on this subdiagram.
- AUTHORIZATION_RESPONSES subdiagram — Handles notifications for the topics AUTHORIZE, DE_AUTHORIZATION, PEND_PUBLICATION, PRE_AUTHORIZATION, and REJECT_PUBLICATION. See the section "AUTHORIZATION_RESPONSES subdiagram" on page 41 for more information on this subdiagram.

- DEAD_LETTER_PUB_RECEIPT subdiagram — Handles notifications associated with the single notification topic DEAD_LETTER_PUB_RECEIPT. See the section "DEAD_LETTER_PUB_RECEIPT subdiagram" on page 41 for more information on this subdiagram.
- CATALOGUE_ITEM_CONFIRMATION — Handles the process of the CATALOGUE_ITEM_CONFIRMATION responses, received by the UCCnet_processWorklist collaboration object. See "CATALOGUE_ITEM_CONFIRMATION" on page 42 for more information.
- RCIR_ADD_RESPONSE — Handles notifications for the ADD and CHANGE topics. See the section "RCIR_ADD_RESPONSE" on page 42 for more information.
- SIMPLE_RESPONSE subdiagram — Handles immediate responses to commands such as MDNs from UCCnet. See the section "SIMPLE_RESPONSE subdiagram" on page 43 for more information on this subdiagram.
- UNKNOWN_MESSAGES subdiagram — Handles incoming messages not recognized as supported. See the section "UNKNOWN_MESSAGES subdiagram" on page 43 for more information on this subdiagram.
- UNKNOWN_RESPONSE subdiagram — Handles incoming messages that are recognized as notification messages, but are not supported. See the section "UNKNOWN_RESPONSE subdiagram" on page 43 for more information on this subdiagram.

NEW_ITEM_PUBLICATION_REQUEST subdiagram

This subdiagram handles notifications associated with the single notification topic NEW_ITEM_PUBLICATION_REQUEST. This notification is generated as a result of the following:

- A new item being added to the source ERP.
- Item data being updated in the source ERP.
- A demand-side trading partner requesting through UCCnet that a supply-side trading partner publish a specific item (GTIN) or items to it so it can synchronize them. The demand-side partner's request produces a notification in the worklist of the supply-side trading partner. This notification has the type="NEW_ITEM_PUBLICATION_REQUEST".

The logic in this subdiagram does the following:

1. Verifies that the GTIN value associated with the item is in the PROCESSED_GTIN table and that the item is not withdrawn. See the section "Using the PROCESSED_GTIN table" on page 34 for more information on the PROCESSED_GTIN table.
2. Checks that the new item or new item information to be published is supplied by a trading partner listed in the trading_partner table and verifies that the demand-side trading partners to whom notification will be sent are also in this table. See the section "Using the trading_partner table" on page 36 for more information on the trading_partner table.
3. Sends the ItemPublicationAdd or ItemPublicationChange for the GTIN to the demand-side trading partners identified in the business object. It accomplishes this by mapping the business object into an application-specific business object of the form UCCnet*XXX*_envelope business object.

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

   The business object is converted by passing through router and transformation maps of the forms

RouterMap_UCCnetGBO_envelope_to_UCCnet*XXX*_envelope and UCCnetGBO_envelope_notification_to_UCCnet*XXX*_envelope_publishCommand. The message is sent over the *NEW_ITEM_PUBLICATION_REQUEST* port to the AS2 channel connector, from the AS2 channel connector to the AS2 channel server, and then to UCCnet.

4. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.

The follow-up workflow is that of the second subflow of either the ItemPublicationAdd or ItemPublicationChange workflow, as detailed in the sections "ItemPublicationAdd subflow 2: processing trading partners' responses to a new item" on page 9 and "ItemPublicationChange subflow 2: processing trading partners' responses to updated item information" on page 16.

INITIAL_ITEM_LOAD_REQUEST subdiagram

This subdiagram handles notifications associated with the single notification topic INITIAL_ITEM_LOAD_REQUEST. This notification is generated as a result of a demand-side trading partner requesting through UCCnet to initiate synchronizing all the items currently traded with a given supply-side trading partner. The demand-side partner's request produces a notification in the worklist of the supply-side trading partner.

The logic in this subdiagram does the following:

1. Writes a record to the audit_log table indicating receipt of the INITIAL_ITEM_LOAD_REQUEST. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.
2. Checks the trading_partner table to verify that the GLN requesting the INITIAL_ITEM_LOAD_REQUEST exists. See the section "Using the trading_partner table" on page 36 for more information on the trading_partner table.
3. Sends the business object to UCCnet over the *INITIAL_ITEM_LOAD_REQUEST* port via the AS2 channel connector.

The follow-up workflow is that of an ItemPublicationAdd subflow 1 targeted to the trading partner who initiated the flow, as detailed in the section "ItemPublicationAdd subflow 1: making a new item available to trading partners" on page 7. The PROCESSED_GTIN table provides the list of GTINs for the ItemPublicationAdd, and the incoming message provides the trading partner's GLN. There are no external business process steps for this flow.

ITEM_ADD_CHANGE subdiagram

This subdiagram handles notifications for the ITEM_ADD and ITEM_CHANGE topics. These notifications are sent by UCCnet to indicate completion of a particular item synchronization request, such as one initiated by an ItemAdd or ItemChange workflow. The subdiagram logic does the following:

1. Maps the UCCnetGBO_envelope business object into an application-specific business object of the form UCCnet*XXX*_envelope business object.

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

   The UCCnetGBO_envelope business object is converted by passing through router and transformation maps of the forms RouterMap_UCCnetGBO_envelope_to_UCCnet*XXX*_envelope and UCCnetGBO_envelope_notification_to_UCCnet*XXX*_envelope_

publishCommand. The business object contains the corresponding ItemPublicationAdd or ItemPublicationChange request. This message includes a request for UCCnet to publish the item to the trading partners listed in the message.

2. Sends the ItemPublicationAdd or ItemPublicationChange request over the *ITEM_ADD_CHANGE* port to the AS2 channel connector.
3. Logs the notification in the audit_log table. See the section "Using the audit_log table" on page 35 for more information on the audit_log table.

The follow-up workflow is that of the first subflow of either the ItemPublicationAdd or ItemPublicationChange workflow, as detailed in the sections "ItemPublicationAdd subflow 1: making a new item available to trading partners" on page 7 and "ItemPublicationChange subflow 1: making updated item information available to trading partners" on page 14.

CATEGORY_ADD_CHANGE subdiagram

This subdiagram handles notifications for the CATEGORY_ADD and CATEGORY_CHANGE topics. These notifications are sent by UCCnet when a request is made by the supply-side trading partner to add or change a category to better classify or organize the items in its available inventory. The subdiagram logic sends an email containing the category maintenance information to selected recipients by instantiating a collaboration object based on the Notify_by_eMail collaboration template called UCCnet_processWorklist_CATEGORY_ADD_CHANGEObject. The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information. There is no follow-on flow.

AUTHORIZATION_RESPONSES subdiagram

This subdiagram handles notifications for the following topics: AUTHORIZE, DE_AUTHORIZATION, PEND_PUBLICATION, PRE_AUTHORIZATION, and REJECT_PUBLICATION. UCCnet generates these notifications as a result of authorization actions taken by demand-side trading partners, which are forwarded to UCCnet. Typically they are issued in response to an ItemPublicationAdd issued by the supply-side trading partner, but they can be issued at anytime. The subdiagram logic does the following:

1. Instantiates a collaboration object based on the Notify_by_eMail collaboration template called UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject, which sends an email to selected recipient(s). The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.
2. Logs the event in the audit_log table. See "Using the audit_log table" on page 35 for more information on the audit_log table.

DEAD_LETTER_PUB_RECEIPT subdiagram

This subdiagram handles notifications for the DEAD_LETTER_PUB_RECEIPT topic. These notifications result from a supplier request for which a target demand-side trading partner has not subscribed. The subdiagram logic instantiates a collaboration object based on the Notify_by_eMail collaboration template called

UCCnet_processWorklist_DEAD_LETTER_PUB_RECEIPTObject, which sends an email to selected recipient(s). The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.

CATALOGUE_ITEM_CONFIRMATION

This subdiagram handles the process of the CATALOGUE_ITEM_CONFIRMATION responses, received by the UCCnet_processWorklist collaboration object. UCCnet generates these responses as a result of the authorization actions taken by demand-side trading partners. The responses can have one of the following states:

- SYNCHRONISED
- ACCEPTED
- REVIEW
- REJECTED

The responses are normally generated in answer to a request generated by the supply-side trading partner as part of the CatalogueItemNotification_Change workflow. However, the responses can be issued at any time. This subdiagram carries out the following logic:

- It instantiates a collaboration object called UCCnet_processWorklist_CATALOGUE_ITEM_CONFIRMATIONObject, based on the Notify_by_eMail collaboration template. This collaboration object sends an email to a set of defined recipients. The message, subject, and recipient list are defined by the collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectfully. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.
- It logs the event in the audit_log table. See the section "Using the audit_log table" on page 35 for more information.

RCIR_ADD_RESPONSE

This subdiagram handles notifications for the ADD and CHANGE topics. These notifications are sent by UCCnet when a request in made by the supply-side trading partner to add or change an item.

The subdiagram logic first records the occurrence of the RCIR_RESPONSE message in the audit_log. Then, it builds a skeleton ItemBasic business object, defining only the UPCEANCODE and ITEM_DOMAIN attributes. It next sends this ItemBasic business object out through the DestinationAppRetrieve port so that the user can respond with a completed ItemBasic Business Object. In a production environment, an additional process is required to retrieve the fully defined ItemBasic business object using the UPCEANCODE and ITEM_DOMAIN fields, and to return it to the DestinationAppRetrieve port.

Once the fully defined ItemBasic business object has been returned, the RCIR_ADD_RESPONSE subdiagram sends it out through the CIN_DISPATCHER port to the UCCnetXSD_CIN_Dispatcher collaboration object. This action triggers the UCCnetXSD_CIN_Dispatcher collaboration object to generate a CATALOGUE_ITEM_NOTIFICATION message and send it to any interested trading partners.

SIMPLE_RESPONSE subdiagram

This subdiagram handles immediate responses to commands such as MDNs from UCCnet. The subdiagram logic instantiates a collaboration object based on the Notify_by_eMail collaboration template called UCCnet_processWorklist_SIMPLE_RESPONSEObject, which sends an email to selected recipient(s). The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.

UNKNOWN_MESSAGES subdiagram

This subdiagram handles incoming messages not recognized as supported. The subdiagram logic instantiates a collaboration object based on the Notify_by_eMail collaboration template called UCCnet_processWorklist_UNKNOWN_MESSAGESObject, which sends an email to selected recipient(s). The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.

UNKNOWN_RESPONSE subdiagram

This subdiagram handles incoming messages that are recognized as notification messages, but are not supported. The subdiagram logic instantiates a collaboration object based on the Notify_by_eMail collaboration template called UCCnet_processWorklist_UNKNOWN_RESPONSEObject, which sends an email to selected recipient(s). The email message, subject, and recipients are configured in this collaboration object's EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. See the section "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44 for more information.

Sending email

UCCnet_ItemSync and UCCnet_processWorklist collaboration objects can be configured to send email to alert when processing errors occur. A UCCnet_processWorklist collaboration object can also instantiate collaboration objects based on the Notify_by_eMail collaboration template to respond by email to configured recipients when specific processing circumstances occur.

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

- "Alerting email recipients of processing errors"
- "Sending email through UCCnet_processWorklist collaboration object subdiagrams" on page 44

Alerting email recipients of processing errors

UCCnet_ItemSync and UCCnet_processWorklist collaboration objects can be configured to send email to alert recipients when processing errors occur.

UCCnet_ItemSync collaboration object

This collaboration object uses two configuration properties to control whether email is sent and to identify the mail recipients.

- SEND_EMAIL — This property controls whether email is sent to the email address specified in the SEND_EMAIL_TO configuration property. Set the property value to `all` to send email or to `none` to not send email. If the value is left empty, no email is sent even if recipients exist in the SEND_EMAIL_TO property.
- SEND_EMAIL_TO — This property defines the email addresses to which error messages are sent. Multiple addresses can be provided in a comma-delimited list. This property must be configured by the user.

UCCnet_processWorklist collaboration object

This collaboration object uses one configuration property to control whether email is sent and to identify the mail recipients. The SEND_EMAIL_TO property defines the email addresses to which error messages are sent. Multiple addresses can be provided in a comma-delimited list. If a value exists for this property, the collaboration object sends email. If the property is left blank, the object does not send email. This property must be configured by the user.

Sending email through UCCnet_processWorklist collaboration object subdiagrams

A UCCnet_processWorklist collaboration object contains several subdiagrams that respond to specific types of workflow processing. Several of these subdiagrams include functionality that sends an email to a set of configured addresses by instantiating a collaboration object based on the Notify_by_eMail collaboration template, as follows:

- CATEGORY_ADD_CHANGE subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_CATEGORY_ADD_CHANGEObject.
- AUTHORIZATION_RESPONSES subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_AUTHORIZATION_RESPONSESObject.
- DEAD_LETTER_PUB_RECEIPT subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_DEAD_LETTER_PUB_RECEIPTObject.
- CATALOGUE_ITEM_CONFIRMATION subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_CATALOGUE_ITEM_CONFIRMATION
- SIMPLE_RESPONSE subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_SIMPLE_RESPONSEObject.
- UNKNOWN_MESSAGES subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_UNKNOWN_MESSAGESObject.
- UNKNOWN_RESPONSE subdiagram — Instantiates a collaboration object called UCCnet_processWorklist_UNKNOWN_RESPONSEObject.

Each of these objects based on the Notify_by_eMail collaboration template can be configured to contain the email message, subject, and recipients specific to its processing situation through its EMAIL_MESSAGE, EMAIL_SUBJECT, and EMAIL_NOTIFICATION_RCPTS configuration properties, respectively. These properties can also contain the names of files, which 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 Notify_by_eMail collaboration object substitutes data from the business object into these variables dynamically. See the following sections for more information on these features:

- "Specifying message text, subjects, and recipients in external files" on page 45

- "Specifying changing individual or multiple message recipients"
- "Using substitution variables in message and subject text" on page 46

The value of the AUTO_RESPOND property of the UCCnet_processWorklist collaboration object determines whether email is sent. The value of this collaboration object's DTD_URL property sets the DTD line in the XML in any outgoing XML message.

Specifying message text, subjects, and recipients in external files

A Notify_by_eMail collaboration object allows the contents of its properties 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.

A Notify_by_eMail collaboration object uses the following configuration properties to identify the email message text, subject text, and recipients:

EMAIL_MESSAGE
Identifies the message text.

EMAIL_SUBJECT
Identifies the subject text.

EMAIL_NOTIFICATION_RCPTS
Identifies the recipient or list of recipients.

The collaboration object distinguishes whether the content of a property is an actual value or filename based on whether the value is prefixed by the character `@`. If the value of the property is prefixed with the character `@`, the Notify_by_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`, set the value of the EMAIL_NOTIFICATION_RCPTS property, as follows:

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

If the value of a property does not start with the character `@`, the Notify_by_eMail collaboration object obtains the email value directly from the attribute.

Specifying changing individual or multiple message recipients

A Notify_by_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 collaboration object based on the Notify_by_eMail template substitutes data from the business object into these variables dynamically. Variables to be substituted must be enclosed in the prefix characters ${ and the suffix character }. 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:

getRoot
Substitutes the entire triggering business object.

getDate
Substitutes the current date and time.

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. For instance, in the following sample message:

```
UCCnet_processWorklist_AUTHORIZATION_RESPONSES.mail:            \
Date:  ${getDate}
BusinessObject:  ${getName}.${getVerb}
Topic:
${ROOT.body[0].response.acknowledge.acknowledgement.            \
subdocumentValid[0].subdocumentValid[0]resultList[0].           \
notification.topic}
GLN:
${ROOT.body[0].response.acknowledge.acknowledgement.            \
subdocumentValid[0].subdocumentValid[0].resultList[0].          \
notification.notificationDetail.transactionInformation.         \
entityIdentification.globalLocationNumber.gln}
GTIN:
${TLO.body.body_Wrapper1[0].response.acknowledge.               \
acknowledgement.subdocumentValid[0].subdocumentValid[0].        \
resultList.resultList_Wrapper1[0].notification.                 \
notificationDetail.authorizationNotification.publication.       \
item.itemInformation.globalTradeItemNumber.gtin}

${getRoot}
```

the following variables are filled in automatically during the generation of the message, as follows:

- ${getDate}, with the current date and time.
- ${getName}, with the name of the triggering business object.
- ${getVerb}, with the verb of the triggering business object.

- All variables beginning with `${ROOT.body[0]. . .}`, with the values for those attributes.
- `${getRoot}` with the entire triggering business object.

Logging

If UCCnet_ItemSync, UCCnet_requestWorklist, UCCnet_processWorklist, and Notify_by_eMail collaboration objects encounter error situations during any stage of processing, they do the following:

- Log the error in the configured log destination.
- Return the object to the calling collaboration object through the *From* port.

Note: For error logging to occur, tracing must be enabled. Also, use separate files for tracing and logging. 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 IBM WebSphere Business Integration Express for Item Synchronization product. 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.

Tracing

All collaboration objects based on collaboration templates included in the IBM WebSphere Business Integration Express for Item Synchronization product 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 IBM WebSphere Business Integration Express for Item Synchronization product. 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.

Use cases

The following sections outline the actors and courses for the use cases supplied with the IBM WebSphere Business Integration Express for Item Synchronization product.

Actors

Each use case in the IBM WebSphere Business Integration Express for Item Synchronization product uses multiple actors detailed in the table below.

Table 1. Use case actors

Actor	Description
ERP Legacy System	An Enterprise Resource Planning back-end system storing a supply-side trading partner's product information.
Supplier	The establishment participating as the supply-side trading partner in the exchange of product information.
Retailer	The establishment participating as the demand-side trading partner in the exchange of product information.
UCCnet	A subsidiary of the Uniform Code Council, Inc., that implements multi-industry standards for product identification and related electronic communications. It accomplishes these tasks by requiring its community of trading partners to provide standardized item data in particular formats to its registry. The system also includes integral compliance checks to assure the integrity of the data in all cases.

Courses

ItemAdd

The following table describes the main course that can be followed in the ItemAdd use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 2. ItemAdd use case description (main course) — DTD support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides the item (e.g., an IDOC from SAP) to the legacy application connector (in this solution, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into an application-specific business object, a UCCnetDTD_envelope by passing through the appropriate maps. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemAdd document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.

Table 2. ItemAdd use case description (main course) — DTD support (continued)

#	Activity by the actor	System activity
M3	UCCnet sends confirmation of receipt.	The Message Disposition Notification (MDN) is generated by UCCnet and returned to the AS2 channel server.
M4	Supplier receives receipt acknowledgement.	None
M5	UCCnet checks that item data meets compliance specifications.	None
M6	If item data meets compliance specifications, UCCnet adds the item to its registry and sends the supplier confirmation of the ItemAdd.	None
M7	Supplier receives confirmation of ItemAdd.	None

Table 3. ItemAdd use case description (main course) — schema support

#	Activity by the actor	System activity
M1	Supplier assembles item data	A trigger from the ERP legacy system provides the item to the legacy application connector. For example, an IDOC from SAP is provided to the SAPConnector. A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its To port. Before the business object arrives at the connector, it is passed through a map to convert it into a UCCnetXSD_envelope business object. The connector builds the UCCnet XML document form the incoming business object and delivers this ItemAdd document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	ItemCommandRouter is called.	The UCCnet_ItemSync collaboration routes the incoming ItemBasic business object to the ItemCommandRouter. The item is sent out to the TO_RCIR port for delivery to UCCnet, and gets routed through the map for delivery to UCCnet.
M4	Supplier receives receipt acknowledgement	None.
M5	UCCnet checks that the item data meets compliance specifications.	None

Table 3. ItemAdd use case description (main course) — schema support (continued)

#	Activity by the actor	System activity
M6	If item data meets compliance specifications, UCCnet adds the item to its registry and sends the supplier confirmation of the ItemAdd	None.
M7	System receives RCIR_ADD_RESPOINSE confirmation of ItemAdd.	CATALOGUE_ITEM_NOTIFICATION_ADD use case begins.

ItemPublicationAdd

The following table describes the main course that can be followed in the ItemPublicationAdd use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 4. ItemPublicationAdd use case description (main course)

#	Activity by the actor	System activity
M1	UCCnet responds to a polling request from the IBM WebSphere Business Integration Express for Item Synchronization product.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.
M2	UCCnet responds by sending any notification messages out to the AS2 channel server.	Subsequently, a worklist is delivered to the AS2 channel server from UCCnet which contains the notification (response) PUB_RELEASE_NEW_ITEM. The response document is delivered to the AS2 channel connector. The business object is delivered to the UCCnet_processWorklist collaboration object. This collaboration object identifies the business object as representing a PUB_RELEASE_NEW_ITEM. It dispatches it to its NEW_ITEM_PUBLICATION_REQUEST subdiagram, which checks the message for validity, performs audit logging, and sends the business object back to UCCnet with instructions to publish the new item to the target Retailer trading partners.
M3	The Retailer receives publication of the new item.	None
M4	The Retailer returns one of the authorization responses.	In response, these Retailers can respond with any of the following: AUTHORIZE, PEND_PUBLICATION, REJECT_PUBLICATION, PRE_AUTHORIZATION, or DE_AUTHORIZATION.
M5	UCCnet receives a polling request from the product for new notification messages.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.
M6	The response document is delivered to the AS2 channel server.	None

Table 4. ItemPublicationAdd use case description (main course) (continued)

#	Activity by the actor	System activity
M7	The product receives an Authorization response and processes it.	The business object is delivered to the UCCnet_processWorklist collaboration object, which performs audit logging and emails the configured recipients based on which type of response is received.

CatalogueItemNotification_Add

The following table describes the main course that can be followed in the CatalogueItemNotification_Add use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 5. CatalogueItemNotification_Add use case description (main course)

#	Activity by the actor	System activity
M1	UCCnet responds to a polling request from the IBM WebSphere Business Integration Express for Item Synchronization.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.
M2	UCCnet responds by sending any notification messages out to the AS2 channel server.	In response, a worklist containing the PUB_RELEASE_NEW_ITEM notification is delivered to the AS2 channel server from UCCnet. The response document is delivered to the AS2 channel connector. The business object is delivered to the UCCnet_processWorklist collaboration object. This collaboration object identifies the business object as representing a PUB_RELEASE_NEW_ITEM. The collaboration object then dispatches it to the NEW_ITEM_PUBLICATION_REQUEST subdiagram, which checks the message for validity, performs audit logging, and sends the business object back to UCCnet with instructions to publish the new item to the target Retailer trading partners.
M3	ItemCommandRouter is called.	The ItemSync collaboration object routes the incoming ItemBasic business object to the ItemCommandRouter. The item is then routed through the map and is then sent out to the TO_RCIR port for delivery to UCCnet.
M4	Supplier receives acknowledgement.	None
M5	UCCnet checks that the item data meets compliance specifications.	None
M6	If the item data is compliant, UCCnet adds the item to its registry and sends the supplier confirmation of the ItemAdd	None
M7	System receives RCIR_ADD_RESPONSE confirmation of ItemAdd.	CIN_ADD use case begins.

ItemChange

The following table describes the main course that can be followed in the ItemChange use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 6. ItemChange use case description (main course) — DTD support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides a change to item information to the legacy application connector (in this solution, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into an application-specific business object, a UCCnetDTD_envelope business object, by passing through the appropriate maps. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemChange document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	UCCnet sends confirmation of receipt.	The MDN is generated by UCCnet and returned to the AS2 channel server.
M4	Supplier receives receipt acknowledgement.	None
M5	UCCnet checks that item data meets compliance specifications.	None
M6	If item data meets compliance specifications, UCCnet adds the item information to its registry and sends the supplier confirmation of the ItemChange.	None
M7	Supplier receives confirmation of ItemChange.	None

Table 7. ItemChange use case description (main course) — Schema support

#	Activity by the actor	System activity
M1	Supplier assembles item data	A trigger from the ERP legacy system provides the item to the legacy application connector. For example, an IDOC from SAP is provided to the SAPConnector. A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.

Table 7. ItemChange use case description (main course) — Schema support (continued)

#	Activity by the actor	System activity
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object sends the ItemBasic business object through its TO port to the AS2 channel connector. Before the business object arrives at the connector, it is converted into a UCCnetXSD_envelope. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemAdd document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	ItemCommandRouter is called.	The UCCnet_ItemSync collaboration routes the incoming ItemBasic business object to the ItemCommandRouter. The item is sent out to the TO_RCIR port for delivery to UCCnet, and gets routed through the map for delivery to UCCnet.
M4	Supplier receives receipt acknowledgement	None.
M5	UCCnet checks that the item data meets compliance specifications.	None
M6	If item data meets compliance specifications, UCCnet adds the item to its registry and sends the supplier confirmation of the ItemAdd	None.
M7	System receives RCIR_ADD_RESPONSE confirmation of ItemChange.	CIN_CHANGE use case begins.

ItemPublicationChange

The following table describes the main course that can be followed in the ItemPublicationChange use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 8. ItemPublicationChange use case description (main course)

#	Activity by the actor	System activity
M1	UCCnet responds to a polling request from the IBM WebSphere Business Integration Express for Item Synchronization product.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.

Table 8. ItemPublicationChange use case description (main course) (continued)

#	Activity by the actor	System activity
M2	UCCnet responds by sending any notification messages out to the AS2 channel server.	Subsequently, a worklist is delivered to the AS2 channel server from UCCnet which contains the notification (response) PUB_RELEASE_DATA_CHANGE. The response document is delivered to the AS2 channel connector. The business object is delivered to the UCCnet_processWorklist collaboration object. This collaboration object identifies the business object as representing a PUB_RELEASE_DATA_CHANGE. It dispatches it to its NEW_ITEM_PUBLICATION_REQUEST subdiagram, which checks the message for validity, performs audit logging, and sends the business object back to UCCnet with instructions to publish the new item information to the target Retailer trading partners.
M3	A Retailer receives publication of the new item.	None
M4	The Retailer returns one of the authorization responses.	In response, these Retailers can respond with any of the following: AUTHORIZE, PEND_PUBLICATION, REJECT_PUBLICATION, PRE_AUTHORIZATION, or DE_AUTHORIZATION.
M5	UCCnet receives a polling request from the product for new notification messages.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.
M6	The response document is delivered to the AS2 channel connector.	None
M7	The product receives an Authorization response and processes it.	The business object is delivered to the UCCnet_processWorklist collaboration object, which performs audit logging and emails the configured recipients based on which type of response is received.

CatalogueItemNotification_Change

The following table describes the main course that can be followed in the CatalogueItemNotification_Change use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 9. CatalogueItemNotification_Change use case description (main course)

#	Activity by the actor	System activity
M1	UCCnet responds to a polling request from the IBM WebSphere Business Integration Express for Item Synchronization product.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.

Table 9. CatalogueItemNotification_Change use case description (main course) (continued)

#	Activity by the actor	System activity
M2	UCCnet responds by sending any notification messages out to the AS2 channel server.	Subsequently, a worklist is delivered to the AS2 channel server from UCCnet which contains the notification (response) RCIR_CHANGE_RESPONSE. The response document is delivered to the AS2 channel connector. The business object is delivered to the UCCnetXSD_processWorklist collaboration object. This collaboration object identifies the business object as representing a RCIR_CHANGE_REPONSE. It dispatches it to its RCIR_CHANGE_RESPONSE subdiagram, which checks the message for validity, performs audit logging. Then it sends the ItemBasic to the UCCnet_CIN_Dispatcher collaboration object, which in turn sends a CATALOGUE_ITEM_NOTIFICATION to UCCnet for each GLN interested in the modified item.
M3	The retailer receives publication of the new item.	None
M4	The Retailer returns one of the authorization responses.	In response, these Retailers can respond with any of the following: REVIEW, REJECTED, ACCEPTED, or SYNCHRONISED.
M5	UCCnet receives a polling request from the product for new notification messages.	The UCCnet_requestWorklist collaboration object polls UCCnet for the pending worklist.
M6	The response document is delivered to the AS2 channel connector.	None
M7	The product receives a response and processes it.	The business object is delivered to the UCCnet_processWorklist collaboration object, which performs audit logging and emails the configured recipients.

ItemDelist

The following table describes the main course that can be followed in the ItemDelist use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 10. ItemDelist use case description (main course) — DTD support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides an item delist to the legacy application connector (in this solutiont, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.

Table 10. ItemDelist use case description (main course) — DTD support (continued)

#	Activity by the actor	System activity
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into a UCCnetDTD_envelope business object by passing through the appropriate maps. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemDelist document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	UCCnet sends confirmation of receipt.	The MDN is generated by UCCnet and returned to the AS2 channel server.

Table 11. ItemDelist use case description (main course) — schema support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides an item withdraw to the legacy application connector (in this solution, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the ItemCommandRouter. The ItemCommandRouter delivers the ItemBasic business object to the UCCnetXSD_CIN_Dispatcher. The UCCnetXSD_CIN_Dispatcher builds an appropriate CIN_DELIST message for each subscribed GLN and delivers each to the AS2 channel connector over its To port. Before the business object arrives at the connector, it is passed through a map to convert it into a UCCnetXSD_envelope. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemDelist document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	UCCnet sends confirmation of receipt.	The MDN is generated by UCCnet and returned to the AS2 channel server.

ItemWithdrawal

The following table describes the main course that can be followed in the ItemWithdrawal use case of the IBM WebSphere Business Integration Express for Item Synchronization product.

Table 12. ItemWithdrawal use case description (main course) — DTD support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides an item withdrawal to the legacy application connector (in this solution, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.

Table 12. ItemWithdrawal use case description (main course) — DTD support (continued)

#	Activity by the actor	System activity
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the AS2 channel connector over its *To* port. Before the business object arrives at the connector, it is converted into an application-specific business object, a UCCnetDTD_envelope, by passing through the appropriate maps. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemWithdrawal document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	UCCnet sends confirmation of receipt.	The MDN is generated by UCCnet and returned to the AS2 channel server.

Table 13. ItemWithdrawal use case description (main course) — schema support

#	Activity by the actor	System activity
M1	Supplier assembles item data.	A trigger from the ERP legacy system provides an item withdrawal to the legacy application connector (in this solution, the SAPConnector). A map then generates the ItemBasic business object, which is passed to a UCCnet_ItemSync collaboration object.
M2	Supplier sends item data to UCCnet.	The UCCnet_ItemSync collaboration object delivers the ItemBasic business object to the ItemCommandRouter. The ItemCommandRouter delivers the ItemBasic business object to the UCCnetXSD_CIN_Dispatcher. The UCCnetXSD_CIN_Dispatcher builds an appropriate CIN_WITHDRAW message for each subscribed GLN and delivers each to the AS2 channel connector over its To port. Before the business object arrives at the connector, it is converted into an application-specific business object, a UCCnetXSD_envelope business object, by passing through the appropriate maps. The connector builds the UCCnet XML document from the incoming business object and delivers this ItemDelist document to the AS2 channel server. The AS2 channel server creates the digest, encrypts, and transmits the document to UCCnet.
M3	UCCnet sends confirmation of receipt.	The MDN is generated by UCCnet and returned to the AS2 channel server.

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
CrossWorlds
DB2
DB2 Universal Database
Lotus
Lotus Notes
MQIntegrator
MQSeries
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.

WebSphere Business Integration Express for Item Synchronization V 4.3