BEA WebLogic Collaborate Enabler for RosettaNet

User Guide

BEA WebLogic Collaborate Enabler for RosettaNet 1.0
Document Edition 1.0
February 2001

Copyright

Restricted Rights Legend

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA WebLogic Collaborate Enabler for RosettaNet User Guide

Document Edition	Date	SoftwareVersion
1.0	February 2001	1.0

Contents

About This Document

User Guide

Index

About This Document

This document describes the BEA WebLogic Collaborate Enabler for RosettaNet and explains how to work with it. This document covers the following topics:

- Overview of the WebLogic Collaborate Enabler for RosettaNet
- Architecture and Product Overview
- Configuring the WebLogic Collaborate Enabler for RosettaNet Software
- WebLogic Process Integrator Features for RosettaNet PIP Workflows
- Using the Workflow Examples
- Walkthrough of the Workflow Examples
- Descriptions of Business Operations
- Message Validation Processes

What You Need to Know

This document is written for application developers who are creating c-enablers for BEA WebLogic Collaborate and the RosettaNet business protocol. This document focuses on using the WebLogic Process Integrator, which is included in WebLogic Collaborate, to create workflows for the c-enabler. Therefore, you need to be familiar with WebLogic Collaborate, WebLogic Process Integrator, and RosettaNet.

e-docs Web Site

BEA product documentation is available at http://e-docs.bea.com, which is the BEA product documentation Web site.

How to Print this Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

The online documentation set also includes a PDF version of this document. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDF, open the WebLogic Collaborate Enabler for RosettaNet documentation Home page, click the PDF Files link and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com.

Related Information

For information about WebLogic Collaborate, see the WebLogic Collaborate online documentation set at http://e-docs.bea.com. For information about RosettaNet, see http://www.rosettanet.org.

Contact Us!

Your feedback on the WebLogic Collaborate Enabler for RosettaNet documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Collaborate Enabler for RosettaNet documentation.

In your e-mail message, please indicate that you are using the documentation for the WebLogic Collaborate Enabler for RosettaNet 1.0 release.

If you have any questions about this version of WebLogic Collaborate Enabler for RosettaNet, or if you have problems installing and running WebLogic Collaborate Enabler for RosettaNet, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: buildobjclient [-v] [-o name] [-f *file-list*]... [-l *file-list*]...
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: buildobjclient [-v] [-o name] [-f *file-list*]... [-l *file-list*]...
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

User Guide

The following sections describe the c-enabler for RosettaNet and provide procedures for configuring and using the c-enabler for RosettaNet:

- Architecture and Product Overview
- Configuring the WebLogic Collaborate Enabler for RosettaNet Software
- WebLogic Process Integrator Features for RosettaNet PIP Workflows
- Using the Workflow Examples
- Walkthrough of the Workflow Examples
- Descriptions of Business Operations
- Message Validation Processes

Overview of the WebLogic Collaborate Enabler for RosettaNet

In addition to routing XOCP messages, the WebLogic Collaborate c-hub can route messages that use the RosettaNet business protocol. The basic WebLogic Collaborate c-enabler available with WebLogic Collaborate 1.0 allows applications to engage in business-to-business conversations using the XOCP protocol.

The BEA WebLogic Collaborate Enabler for RosettaNet software consists of a c-enabler for RosettaNet that enhances the XOCP c-enabler by providing the ability to send and receive RosettaNet messages according to the protocol described in the *RosettaNet Implementation Framework version (RNIF) 1.1*. In addition, the c-enabler for RosettaNet extends the basic WebLogic Process Integrator functionality with features that allow WebLogic Process Integrator workflows to participate in RosettaNet Partner Interface Processes (PIPs).

You can configure a c-enabler for RosettaNet to communicate directly with another c-enabler for RosettaNet or any RNIF 1.1 compliant system. You can also configure the c-enabler for RosettaNet to communicate to another c-enabler for RosettaNet or RNIF 1.1 compliant system via the WebLogic Collaborate c-hub.

About RosettaNet

RosettaNet is an independent non-profit consortium of technology companies whose purpose is to define and lead the implementation of open and common electronic business processes for information technology electronic distribution channels. These processes are designed to standardize the electronic business interfaces used by participating supply-chain partners. The *RosettaNet Implementation Framework Specification* (http://www.rosettanet.org) is an implementation guideline for applications that implement RosettaNet Partner Interface Processes (PIPs). RosettaNet describes its PIPs as follows:

“RosettaNet Partner Interface Processes (PIPs) define business processes between supply-chain companies, providing the models and documents for the implementation of standards. PIPs fit into six Clusters, or groups of core business processes, that represent the backbone of the supply chain. Each Cluster is broken down into Segments, cross-enterprise processes involving more than one type of supply chain company. Within each Segment are individual PIPs. PIPs are specialized system-to-system XML-based dialogs that define business processes between supply chain companies. Each PIP includes a technical specification based on the RosettaNet Implementation Framework (RNIF), a Message Guideline document with a PIP-specific version of the Business Dictionary and an XML Message Guideline document (an XML document type definition or DTD).”

The Business Dictionary defines the Business Properties, Business Data Entities, and Fundamental Business Data Entities in PIP Message Guidelines.

Architecture and Product Overview

The c-enabler for RosettaNet consists of the following RosettaNet-specific components:

- RosettaNet Protocol Layer
- WebLogic Collaborate—WebLogic Process Integrator integration layer to support modeling and execution of RosettaNet PIPs
- WebLogic Process Integrator Templates for PIPs

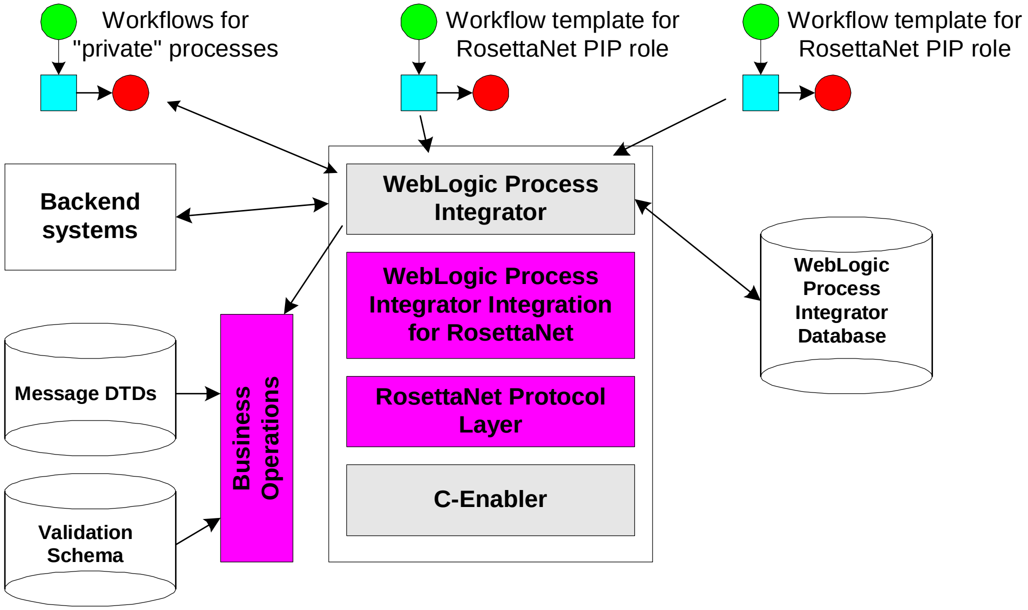
The following c-enabler for RosettaNet components are leveraged from WebLogic Collaborate:

- C-enabler
- WebLogic Process Integrator

For a description of these components, see the *BEA WebLogic Collaborate Developer Guide*.

The following diagram shows the c-enabler for RosettaNet architecture.

Figure 1 C-Enabler for RosettaNet Architecture


Workflows for "private" processes
Workflow template for RosettaNet PIP role
Workflow template for RosettaNet PIP role
Backend systems
WebLogic Process Integrator
WebLogic Process Integrator Integration for RosettaNet
RosettaNet Protocol Layer
C-Enabler
WebLogic Process Integrator Database
Message DTDs
Validation Schema
Business Operations

RosettaNet Protocol Layer

The RosettaNet protocol layer provides the ability to send and receive messages via the Internet according to the RNIF specification for transport, message packaging, and security. The c-enabler for RosettaNet is a factory for c-enabler for RosettaNet sessions. A c-enabler for RosettaNet session manages the URL for a location at which the c-enabler for RosettaNet can receive RosettaNet messages.

Digital Signatures

The c-enabler for RosettaNet does not provide an out-of-the-box implementation for digital signatures, but it offers a plug-in mechanism for security solutions that implement this functionality. For more information, see “Configuring a Plug-In for Digital Signatures” on page 12.

WebLogic Process Integrator Integration for RosettaNet

WebLogic Process Integrator software models and executes workflows that implement RosettaNet PIPs. The RosettaNet protocol layer and WebLogic Process Integrator work together to provide workflows with the ability to do the following:

- Have the WebLogic Process Integrator send RosettaNet messages. (See "Sending a RosettaNet Message" on page 23.)
- Have the WebLogic Process Integrator wait for RosettaNet messages. (See "Waiting for a RosettaNet Message" on page 21.)
- Indicate, through workflow template properties set by you, which PIP and role are implemented by the workflow. (See "Linking WebLogic Process Integrator Templates with PIPs" on page 15.)
- Pass incoming RosettaNet messages to the correct workflow instance and initiate the correct type of workflow upon receipt of a RosettaNet message. (See "Configuring Workflow Sessions" on page 17 and "Starting a Workflow upon Receipt of a RosettaNet Message" on page 19.)

WebLogic Process Integrator Templates for PIPs

RosettaNet PIPs define the public processes in which trading partners participate while performing e-business transactions. For example, PIP3A2 defines the process that a Customer trading partner follows with a Product Supplier trading partner to get the price and availability of goods that the customer wants to buy and the product supplier wants to sell. Trading partners participating in PIPs need to implement the public process defined by their roles in the PIP, and they need to connect their internal systems, as well as their private processes and workflows, to the public process.

The WebLogic Collaborate Enabler for RosettaNet software defines example workflow templates for the following PIPs:

- PIP3A2 Query Price and Availability v. 1.3
- PIP0A1 Notification of Failure v. 1.0

Message Validation

The RosettaNet PIP definitions contain detailed validation rules for messages exchanged in the PIP. These rules go beyond the constraints that can be expressed with an XML Document Type Definition (DTD). Both the service content and the service header of a RosettaNet message need to be validated.

The required validation rules are expressed in XML schema documents. Each PIP workflow template requires PIP-specific XML schema documents. At run time, validation of incoming and outgoing messages is performed by WebLogic Process Integrator business operations that call the XML parser. The XML parser validates the service header and service content by using the corresponding XML schema documents.

For a more detailed description of how messages are validated in the workflow examples, see "Message Validation Processes" on page 85.

Configuring the WebLogic Collaborate Enabler for RosettaNet Software

This section provides configuration information for the WebLogic Collaborate Enabler for RosettaNet software.

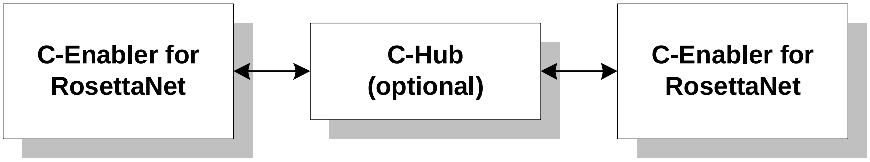
C-Enabler for RosettaNet and C-Hub Configurations

The following sections show three configurations in which the c-enabler for RosettaNet can interact with the c-hub and other systems.

Configuration 1: C-Enabler for RosettaNet to C-Enabler for RosettaNet

Two c-enablers for RosettaNet can interact with each other via a c-hub. Alternatively, a c-enabler for RosettaNet can interact directly with another c-enabler for RosettaNet.

Figure 2 One C-Enabler for RosettaNet Interacting with Another

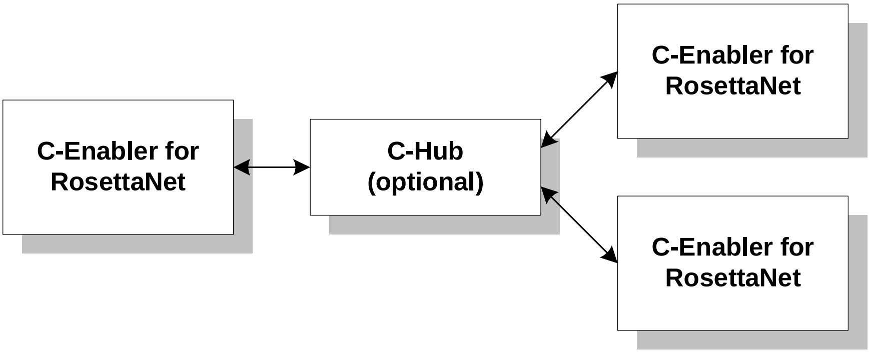


Each c-enabler for RosettaNet must have a separate session and the appropriate workflow template definition for its RosettaNet PIP role.

Configuration 2: C-Enabler for RosettaNet to Multiple C-Enablers for RosettaNet

A c-enabler for RosettaNet can interact with multiple c-enablers for RosettaNet via a c-hub. Alternatively, a c-enabler for RosettaNet can interact directly with multiple c-enablers for RosettaNet if required by the PIP technical specification.

Figure 3 One C-Enabler for RosettaNet Interacting with Multiple C-Enablers for RosettaNet

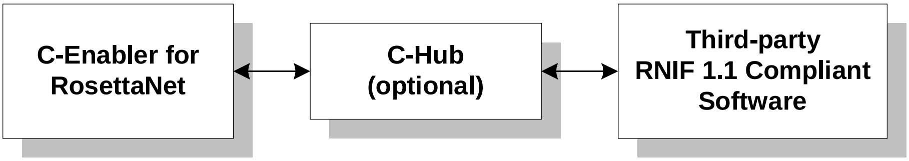


This arrangement should *not* be considered a configuration where one c-enabler for RosettaNet broadcasts messages to other c-enablers for RosettaNet. The RNIF 1.1 specifies only point-to-point communication. Each of the c-enablers for RosettaNet must have a separate session and the appropriate workflow template definition for its RosettaNet PIP role.

Configuration 3: C-Enabler for RosettaNet to RNIF 1.1-Compliant System

A c-enabler for RosettaNet can interact with another RNIF 1.1 compliant system via a c-hub. Alternatively, a c-enabler for RosettaNet can interact directly with another RNIF 1.1 compliant system.

Figure 4 C-Enabler for RosettaNet Interacting with a RNIF 1.1-Compliant System



Each of the RosettaNet PIP trading partners must have a separate session and the appropriate workflow template definition for its RosettaNet PIP role. The workflow template definition is not applicable for a system that does not use the c-enabler for RosettaNet.

Configuring the C-Hub

To configure the WebLogic Collaborate c-hub to route messages to and from the c-enabler for RosettaNet, the c-hub needs to be configured with the following:

- RosettaNet business protocol
- Conversations defined for the RosettaNet PIPs
- Trading partners with the information required by the RosettaNet business protocol
- C-space that is enabled with the RosettaNet business protocol and contains the appropriate role subscriptions

For detailed instructions for configuring the c-hub with this information, see Configuring Business Protocols in the *BEA WebLogic Collaborate C-Hub Administration Guide*.

Configuring the C-Enabler for RosettaNet

The c-enabler for RosettaNet configuration file contains the configuration information for the RosettaNet Protocol layer. The c-enabler for RosettaNet reads this information at start-up time. For general information about configuring c-enablers, see the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

For the c-enabler for RosettaNet, the following information must be included in the c-enabler for RosettaNet configuration file:

- A `business-protocol` element that defines the RosettaNet business protocol on the c-enabler for RosettaNet. Valid business protocols currently include XOCP (supported by the WebLogic Collaborate 1.0 c-enabler) and RosettaNet (supported by the c-enabler for RosettaNet).
- One or more c-enabler for RosettaNet sessions that refer to the RosettaNet business protocol.

The following table summarizes the elements and attributes that are required by the c-enabler for RosettaNet in addition to those described in the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

Table 1 Additional Elements and Attributes for C-Enabler for RosettaNet

Element	Attribute	Description
`business-protocol`		Defines a business protocol. (Subelement of the enabler element.)
	`name`	Specifies the logical name for the protocol.
	`session-factory-class`	Must be set, for RosettaNet, to: `com.bea.b2b.enabler.rosettanet.RNEnablerSessionFactory`

Table 1 Additional Elements and Attributes for C-Enabler for RosettaNet

Element	Attribute	Description
`session`		Defines the c-enabler for RosettaNet session. Multiple sessions may be defined if you need to communicate with more than one trading partner.
	`business-protocol`	Must match the `name` attribute of the `business-protocol` element.
`hub-url`		Defines the URL of the c-hub to which the c-enabler for RosettaNet sends messages. **Note:** If a c-hub is not used, this element defines the URL of the trading partner, which could be another c-enabler for RosettaNet or another RNIF 1.1 compliant system, to which the c-enabler for RosettaNet sends messages.
`trading-partners`	`name`	Specifies the business ID (the RosettaNet specified DUNS ID) of the trading partner hosting the c-enabler for RosettaNet. If the c-enabler for RosettaNet is configured to send messages to a c-hub, this business ID must match the business ID defined for the trading partner in the c-hub repository.

The following example of a c-enabler for RosettaNet configuration file contains settings required to run the c-enabler for RosettaNet.

Listing 1 Example of a C-Enabler for RosettaNet Configuration File

```
<?xml version="1.0"?>
<!DOCTYPE enabler SYSTEM "EnablerConfig.dtd">
<enabler name="Product Supplier">
  <business-protocol
     session-factory-class=
     "com.bea.b2b.enabler.rosettanet.RNEnablerSessionFactory"
     name="RN" />
  <session name="supplier-session" c-space-name="PriceSpace"
         business-protocol="RN">
    <hub-url ref="http://127.0.0.1:7001/TransportServlet/RN"/>
    <enabler-url ref="http://127.0.0.1:7501/rn2"/>
    <security-info>
      <trading-partner name="987654321"/>
    </security-info>
  </session>
</enabler>
```

The example c-enabler for RosettaNet configuration files may also be referenced. They are located in `<WLC_HOME>/rosettanet/enabler`. A configuration file is provided for each workflow that requires a session. The configuration files are `customerEnabler.xml`, `supplierEnabler.xml`, and `failureadminEnabler.xml`. The PIP0A1_Notifier workflow does not require a configuration file because it is a subworkflow of PIP3A2_Customer and uses the latter's configuration settings.

Configuring SSL Security

When you configure the c-enabler for RosettaNet, you need to specify a value for the `security-info` element in the c-enabler for RosettaNet configuration file. If you are using SSL security, you need to specify values for the following subelements:

- Certificate
- Private key
- Trading partner

If you are not using SSL with your c-enabler for RosettaNet, you only need to specify the trading partner subelement. For more information about configuring security information for the c-enabler for RosettaNet, see Configuring C-Enablers in the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

The following example shows a c-enabler for RosettaNet configuration file that contains the settings required to enable SSL.

Listing 2 Example of SSL Configuration

```
<?xml version="1.0"?>
<!DOCTYPE enabler SYSTEM "EnablerConfig.dtd">
<enabler name="Product Supplier">
 <business-protocol
  session-factory-class=
  "com.bea.b2b.enabler.rosettanet.RNEnablerSessionFactory"
  name="RN" />
 <session name="supplier-session" c-space-name="PriceSpace"
  business-protocol="RN">
  <hub-url ref="https://127.0.0.1:7002/TransportServlet/RN"
    certificate-field-name="fingerprint"
    certificate-field-value="1be4aad0678554175170d30c9b6bef0c"
    server-certificate-field-name="fingerprint"
    server-certificate-field-value="1be4aad0678554175170d30c9b6bef0c"
   hub-user="hub"/>
  <enabler-url ref="https://127.0.0.1:7602/rn2"/>
  <security-info>
   <trading-partner name="987654321"/>
   <certificatelocation="D:\bea\wlcsp1\wlcollaborate1.0
    \rosettanet\enabler\myserver\987654321_cert.pem"/>
   <private-key location="D:\bea\wlcsp1\wlcollaborate1.0
    \rosettanet\enabler\myserver\987654321-key.der"/>
  </security-info>
 </session>
</enabler>
```

Configuring a Plug-In for Digital Signatures

Note: The plug-in for digital signatures is an optional feature of the c-enabler for RosettaNet. If you do not use digital signatures skip this section.

The WebLogic Collaborate Enabler for RosettaNet software provides a plug-in architecture for creating and validating digital signatures. The digital signature plug-in is a Java class that implements the com.bea.b2b.protocol.security.WLCDigitalSignatureValidator interface. The RosettaNet protocol layer calls the plug-in before sending a message and after receiving a message.

To configure a digital signature plug-in, add the signature-validator-class attribute to the business-protocol element that defines the RosettaNet business protocol in the c-enabler for RosettaNet configuration file. The value of the signature-validator-class attribute must be the class name of the plug-in. For information about updating the c-enabler for RosettaNet configuration file, see "Configuring the C-Enabler for RosettaNet" on page 9.

You can optionally add the signature-validator-class-init attribute to the same business-protocol element. The string data specified on this attribute is passed to the WLCDigitalSignatureValidator.init(String initString) method. This method is invoked immediately after the object instance for the digital signature plug-in Java class is created.

Sample Digital Signature Plug-In Configuration

The following example shows a c-enabler for RosettaNet configuration file in which a digital signature plug-in is configured.

Listing 3 Example Configuration for a Digital Signature Plug-In

```
<?xml version="1.0"?>
<!DOCTYPE enabler SYSTEM "EnablerConfig.dtd">
<enabler name="Customer">
  <business-protocol
    session-factory-class=
    "com.bea.b2b.enabler.rosettanet.RNEnablerSessionFactory"
    signature-validator-class=
    "mypkg.DigitalSignatureValidator"
    signature-validator-class-init=
    "d:/CertJ10/certj10/sample/db/flatfile"
    name="RN" />
  <session name="customer-session" c-space-name="PriceSpace"
business-protocol="RN">
    <hub-url ref="http://127.0.0.1:7001/TransportServlet/RN"/>
    <enabler-url ref="http://127.0.0.1:7501/rn1"/>
    <security-info>
      <trading-partner name="123456789"/>
    </security-info>
  </session>
</enabler>
```

How the Digital Signature Plug-In Works

After reading in the c-enabler for RosettaNet configuration file, the protocol layer instantiates the digital signature plug-in class. After instantiation, the `init(String initString)` method is invoked. The `initString` passed here is the data available from the `signature-validator-class-init` element. As the implementer of the plug-in class, you should use this `init` string to configure relevant information; for example, the location of the certificates.

The protocol layer calls the `createSignature()` method on the plug-in to create a digital signature for an outgoing message. The message content to be signed is passed in, as well as a `com.bea.protocol.messaging.Message` object representing the entire RosettaNet message. This object should be cast to `com.bea.protocol.rosettanet.messaging.RNMessage` to get more information. (For details about the `RNMessage` class, see the Javadoc provided with WebLogic Collaborate.) If `createSignature()` throws an exception, the message is not sent. The return value of `createSignature()` is the digital signature.

For incoming messages, the protocol layer first calls the `verifySignature()` method to verify the digital signature of the message. The message content and the signature are passed in. If `verifySignature()` throws an exception, the message is rejected and is not processed by the higher layers of the c-enabler for RosettaNet. Subsequently, the `verifySender()` method is called to verify that the trading partner information in the service header and content matches the information in the digital signature. The plug-in can retrieve the service header and content through the `Message` object parameter passed in to the `verifySender()` method. (For details, see the preceding discussion about casting the `Message` object to RNMessage.) If `verifySender()` throws an exception, the message is rejected and is not processed by the higher layers of the c-enabler for RosettaNet.

For information about the `com.bea.b2b.protocol.security.WLCDigitalSignatureValidator` interface, see the Javadoc provided with the WebLogic Collaborate Enabler for RosettaNet software.

Linking WebLogic Process Integrator Templates with PIPs

RosettaNet PIPs define the public processes in which trading partners participate while performing e-business transactions. Trading partners participating in PIPs need to implement the public processes defined by their roles in the PIP, and they need to connect their internal systems, as well as their private processes and workflows, to the public process. The WebLogic Collaborate Enabler for RosettaNet software provides a set of example WebLogic Process Integrator workflow templates that trading partners can use to implement their own PIPs.

When configuring workflow templates that model PIPs, you must include the information described in the following table.

Table 2 Conversation Properties for PIP Workflow Templates

Property	Value	Description
`Name`	*`PIP_name`*	Business process identifier of the PIP, shown in a format identical to that defined for the `GlobalProcessCode` element of the Service Header in the *RosettaNet Service Header Part Message Guideline*. The *RosettaNet Service Header Part Message Guideline* is available at `www.rosettanet.org` under the RNIF 1.1 area, Business Signals, Service Header & Preamble v1.1. Example: The PIP name for PIP3A2 Request Price and Availability should be set to 3A2.
`Version`	*`PIP_version`*	Version of the PIP, for example, 1.3.

Table 2 Conversation Properties for PIP Workflow Templates (Continued)

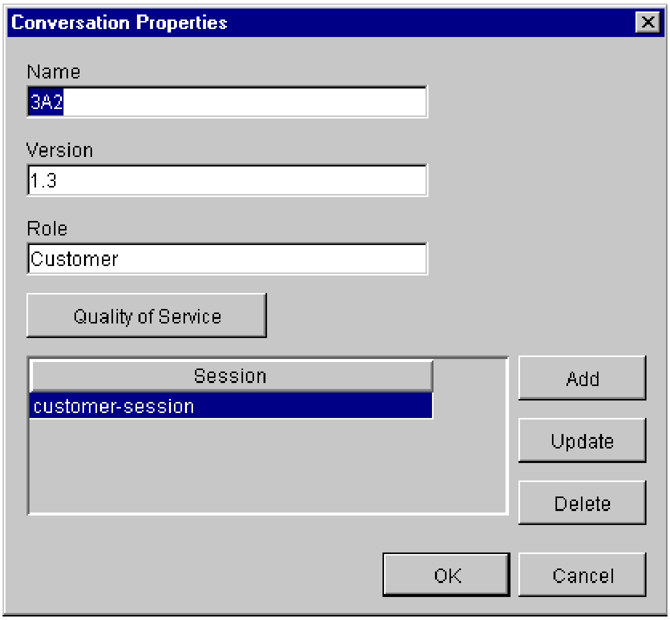
Property	Value	Description
`Role`	*`PIP_roles`*	PIP role for which the template is designed. The format is identical to that defined for the `GlobalRoleClassificationCode` element of the Service Header in the *RosettaNet Service Header Part Message Guideline*. Example: For PIP3A2, a Customer role and a Product Supplier are specified.
`Session`	*`RosettaNet_c-enabler_session`*	C-enabler for RosettaNet session name, as defined in the c-enabler for RosettaNet configuration file. Example: `customer-session` as defined in `<WLC_HOME>/rosettanet/enabler/customerConfig.xml`.

The c-enabler for RosettaNet uses this information at run time to determine the following:

- Which workflow template definition to instantiate upon receipt of a message for a new PIP instance
- Which c-enabler for RosettaNet session to use when a message is sent from a workflow

You configure WebLogic Process Integrator templates in the Conversation Properties dialog box, which you access from the Properties dialog box of the template. For a detailed description of how to open workflow template definitions and define conversation properties, see the topic Defining Conversation Properties in Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

The following illustration shows the Conversation Properties dialog box with the settings required when Name is defined as PIP3A2; Version, as 1.3; Role, as Customer; and Session as customer-session. No information is needed for Quality of Service. It is not applicable.



Conversation Properties

Name: 3A2

Version: 1.3

Role: Customer

Session: customer-session

Configuring Workflow Sessions

The c-enabler for RosettaNet session and all related active WebLogic Process Integrator templates are referred to, collectively, as a workflow session. Workflow sessions for c-enabler for RosettaNet sessions are created when you boot the WebLogic Server that hosts the c-enabler for RosettaNet.

For each workflow session, configure a com.bea.wlpi.Start startup class in the <WLC_HOME>/rosettanet/enabler/weblogic.properties file with the arguments described in the following table, all of which are required.

Table 3 Configuration Arguments for com.bea.wlpi.Start

Argument	Description
ConfigFile	Name of the c-enabler for RosettaNet configuration file.
SessionName	Name of the c-enabler for RosettaNet session.
User	WebLogic Process Integrator user that the c-enabler for RosettaNet should impersonate when creating a new workflow instance (upon receipt of an incoming message).
Password	Password of the WebLogic Process Integrator user.
OrgName	Organization to which the WebLogic Process Integrator user belongs.

The following listing shows a fragment of a sample weblogic.properties file that specifies the startup class and defines a workflow session named customer-session.

Listing 4 Defining a Workflow Session

```
weblogic.system.startupClass.PIP3A2Customer=com.bea.b2b.wlpi.Start

weblogic.system.startupArgs.PIP3A2Customer=ConfigFile=customerEnabler.xml,Sessi
onName=customer-session,

User=bea,Password=12345678,OrgName=BEA
```

The startup class creates the c-enabler for RosettaNet session. It then looks up and registers all the active WebLogic Process Integrator templates that are configured for the c-enabler for RosettaNet session.

WebLogic Process Integrator Features for RosettaNet PIP Workflows

The c-enabler for RosettaNet extends the basic WebLogic Process Integrator functionality with features that allow WebLogic Process Integrator workflows to participate in RosettaNet PIPs. WebLogic Process Integrator workflows can:

- Be started by an incoming RosettaNet message
- Wait for a RosettaNet message
- Send a RosettaNet message

The RosettaNet extensions to WebLogic Process Integrator are similar to those in WebLogic Process Integrator for the WebLogic Collaborate XOCP c-enabler, which are described in Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

Note: To access workflow diagrams, you must invoke the WebLogic Process Integrator Studio. For instructions for starting the WebLogic Process Integrator Studio see “Starting WebLogic Process Integrator Studio” on page 31.

Starting a Workflow upon Receipt of a RosettaNet Message

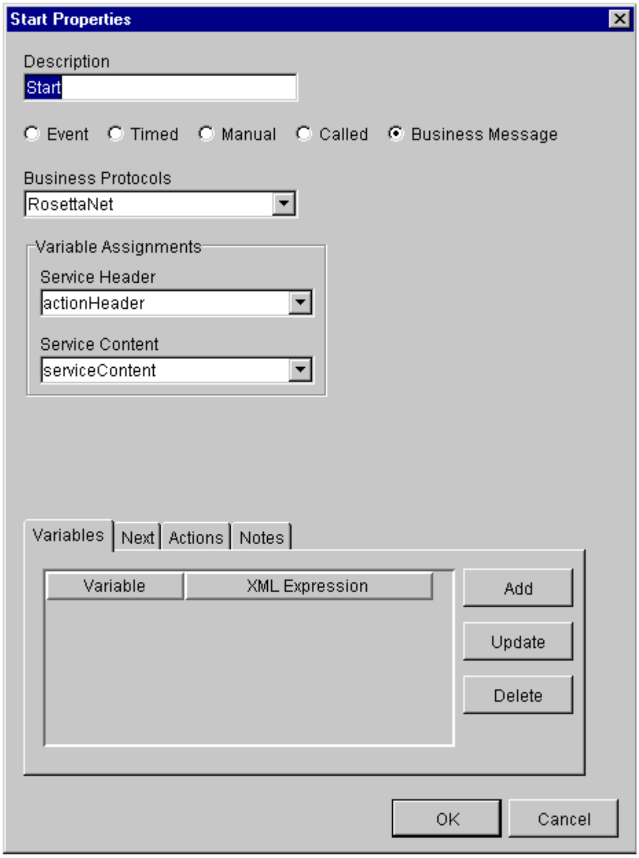
You can configure a workflow to be started automatically by the c-enabler for RosettaNet upon receipt of the first message for a PIP instance. The example PIP3A2_Supplier workflow instance provides an example of a workflow instance that is started when the first message of this PIP for the Supplier role arrives.

The PIP3A2_Supplier workflow instance serves as an example of how to configure a workflow template definition to be started by an incoming message:

1. In the PIP3A2_Supplier workflow diagram, double-click the Start node to display the Start Properties dialog box.

2. Select the Business Message radio button.

3. Set the Business Protocols field to RosettaNet.



Start Properties

Description: Start

○ Event ○ Timed ○ Manual ○ Called ● Business Message

Business Protocols: RosettaNet

Variable Assignments

Service Header: actionHeader

Service Content: serviceContent

4. Set the Variable Assignments fields. The Service Header field is populated with the header of the incoming XML message and the Service Content field is populated with the content of the incoming XML message. Only String variable names are valid in these fields.

Workflow and PIP Instances

Each PIP instance is identified by a unique combination of a process ID (*`ProcessIdentity/instanceIdentifier`*) and the DUNS (Data Universal Numbering System) ID of the initiating trading partner. All RosettaNet messages contain these values in the Service Header. The c-enabler for RosettaNet maintains a list of all workflow instances and the PIP instances associated with them.

After a workflow is started in response to an incoming RosettaNet message, the workflow is associated with the PIP instance to which the message belongs. When a workflow is started by an event, timer, a manual request or a call from another workflow, the c-enabler for RosettaNet creates a new PIP instance for it. In a single c-enabler for RosettaNet session only one workflow instance can be associated with a PIP instance.

Waiting for a RosettaNet Message

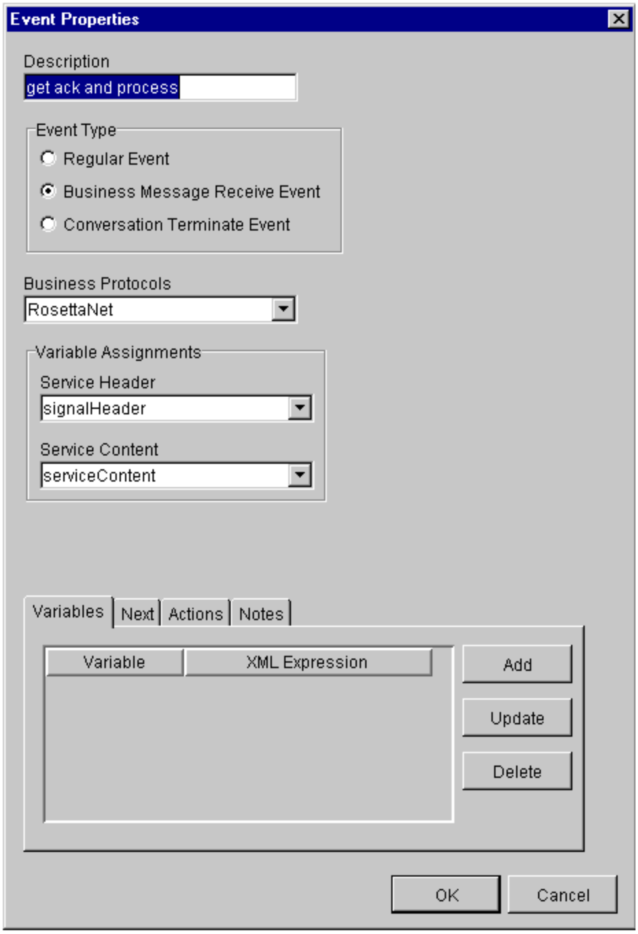
Workflows can contain events that are triggered when a message is received for the PIP instance associated with the workflow. The following example workflows contain events that are triggered when a RosettaNet message is received:

- PIP3A2_Customer
  - Get Ack and Process
  - Get Reply
- PIP3A2_Supplier: Get Ack and Process
- PIP0A1_Notifier: Get Ack and Process

The Get Ack and Process event in the PIP3A2_Supplier workflow instance serves as an example of how to configure an event node to be triggered upon receipt of a RosettaNet message:

1. In the PIP3A2_Supplier workflow diagram, double-click the Get Ack and Process event node to display the Event Properties dialog box.
2. Click the Business Message Receive Event radio button under the Event Type field.

3. Set the Business Protocols field to RosettaNet.



Event Properties

Description: get ack and process

Event Type

○ Regular Event

● Business Message Receive Event

○ Conversation Terminate Event

Business Protocols: RosettaNet

Variable Assignments

Service Header: signalHeader

Service Content: serviceContent

4. Set the Variable Assignments fields. The Service Header field is populated with the header of the incoming XML message, and the Service Content field, with the content of the incoming XML message. Only String variable names are valid in these fields.

Sending a RosettaNet Message

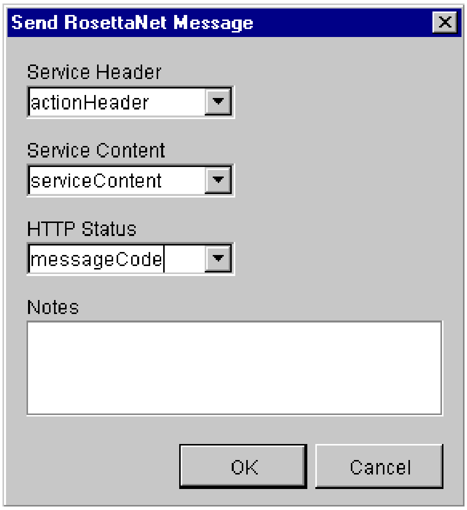
A workflow can send a RosettaNet message by invoking an integration action called Send RosettaNet Message. The following example workflows contain nodes that define Send RosettaNet Message actions:

- PIP3A2_Customer
  - Task: Send Query Message
  - Task: Send Acknowledgement Exception
  - Task: Send Acknowledgement Message
- PIP3A2_Supplier
  - Task: Send Acknowledgement Exception
  - Task: Send Acknowledgement Message
  - Task: Send General Exception
  - Task: Send Response Message
- PIP0A1_Notifier—Task: Send Failure Notification Message
- PIP0A1_Admin
  - Task: Send Acknowledgement Exception
  - Task: Send Acknowledgement Message

For an example of a task that defines a Send RosettaNet Message action, consider the Send Response Message in the PIP3A2_Supplier workflow instance. To walk through this example, complete the following procedure:

1. In the PIP3A2_Supplier workflow diagram, double-click the Send Response Message task node to display the Task Properties dialog box.
2. Select the Activated tab under the Actions field.
3. Choose the Send RosettaNet Message action.

4. Click the Update button to display the Send RosettaNet Message dialog.



Send RosettaNet Message

Service Header: actionHeader

Service Content: serviceContent

HTTP Status: messageCode

Notes

5. Specify parameters for the Send RosettaNet Message action, as described in the following table.

Table 4 Parameters for Send RosettaNet Message Action

In this field . . .	Specify the workflow variable that contains . . .
`Service Header`	String variable that contains information for the RosettaNet message header
`Service Content`	String variable that contains information for the RosettaNet message content
`HTTP Status`	Integer variable that contains the HTTP status resulting from sending the RosettaNet message

The procedure for adding a Send RosettaNet Message action to a Task node is:

1. In the workflow diagram, double-click the Task node to display the Task Properties dialog box.

2. Select the Activated tab under the Actions field.

3. Click the Add button to display the Add Action dialog.

4. In the Integrations Actions folder, choose Send RosettaNet Message.
5. Click OK to display the Send RosettaNet Message dialog.
6. Specify parameters for the RosettaNet Message action, as described in the Parameters for Send RosettaNet Action table.

Using the Workflow Examples

The workflow examples provide templates for specific PIPs and concise scenarios of how to use the WebLogic Collaborate Enabler for RosettaNet technology to develop compliant PIP solutions. This section briefly describes the four workflow examples. For a step-by-step description of each workflow, see “Walkthrough of the Workflow Examples” on page 37.

- *PIP3A2_Customer*—Implements the Customer role of the PIP3A2 Query Price and Availability specification. The customer builds and sends the Query message, waits for an acknowledgement, and then waits for a reply to the query from the supplier.
- *PIP3A2_Supplier*—Implements the Product Supplier role of the PIP3A2 Query Price and Availability specification. This workflow starts upon receipt of a query message. After validating the message, the workflow sends an acknowledgement, builds a response, and sends it. The supplier then waits for an acknowledgement message from the customer.
- *PIP0A1_Notifier*—Implements the Failure Notifier role of the PIP0A1 Notification of Failure specification. This subworkflow starts when the customer workflow exceeds its maximum number of retries while trying to send the Query message. The Failure Notifier builds and sends the notification of failure message and waits for an acknowledgement from the Failure Report Administrator.
- *PIP0A1_Admin*—Implements the Failure Report Administrator role of the PIP0A1 Notification of Failure specification. The Failure Report Administrator starts upon receipt of a failure message. After validating the message, the workflow sends an acknowledgement message to the Failure Notifier.

Configuring the C-Hub for the Workflow Examples

The `<WLC_HOME>/hub/weblogic.properties` file for the c-hub included with the WebLogic Collaborate installation can be used for the examples. For more information about the c-hub `weblogic.properties` file, see the *BEA WebLogic Collaborate C-Hub Administration Guide*.

The c-hub repository data is included for the RosettaNet example. The installation process stores the `RepData.xml` file in the `<WLC_HOME>/rosettanet/hub` directory. The RosettaNet example assumes the c-hub database has been created. For more information about configuring the c-hub repository, see the topic Configuring the Repository in Setting Up the C-Hub in the *BEA WebLogic Collaborate C-Hub Administration Guide*.

To use the RosettaNet business protocol, you must load the c-hub repository with the appropriate data. Therefore, before you can run the example, you must import `RepData.xml` into the c-hub repository, by completing the following procedure:

1. Start the c-hub.
2. In a browser, bring up the C-Hub Administration Console at the following URL:

   ```
   http://<Collaborate machine>:<hubPort>/WLCHubAdmin
   ```

3. In the Console window, select the Configuration option and the Hub tab. The Hub tab is displayed.
4. Click the Import button at the bottom of the Hub tab. The Hub import window is displayed.
5. Click the Browse button to navigate to the `<WLC_HOME>/rosettanet/hub` directory and select the `RepData.xml` file.
6. Select Yes or No for the Initialize Database option. If you select Yes, all other c-hub data are removed. If you select No, the data is loaded with the other repository data.
7. Click the Import button. A status message reports whether the data was imported successfully.

Configuring the C-Enabler for RosettaNet for the Workflow Examples

To configure the c-enabler for RosettaNet for the workflow examples, complete the following procedure:

1. On the command line, navigate to the c-enabler for RosettaNet subdirectory of your WebLogic Collaborate Enabler for RosettaNet installation, `<WLC_HOME>/rosettanet`.
2. Run the `setenv` command. WebLogic Collaborate Enabler for RosettaNet provides `setenv.cmd` for Windows and `setenv.sh` for UNIX.
3. Create the database for the WebLogic Process Integrator. Run appropriate command from the `<WLC_HOME>/rosettanet/wlpi` directory.
   - Oracle 8.1.5—use `createOracle.cmd` for Windows or `createOracle.sh` for UNIX
   - Microsoft SQL Server 7.0—use `createMSSQL.cmd` (Windows only)
   - Cloudscape 3.5.0—use `createCloud.cmd` for Windows or `createCloud.sh` for UNIX
4. Start the c-enabler for RosettaNet.

Configuring WebLogic Process Integrator for the Workflow Examples

This section provides instructions for configuring WebLogic Process Integrator to run the workflow examples. The WebLogic Process Integrator may be configured by using a command script to perform all the configuration steps, alternatively you may follow the manual configuration process.

Using the RNConfig Command Script

The RNConfig command script performs the following steps:

- Imports the business operations
- Creates the workflow templates
- Imports the workflow templates

The RNConfig command scripts assumes the c-enabler for RosettaNet is running with the configuration parameters defined in the following table.

Table 5 Configuration Parameters Required to Run RNConfig

Parameter Name	Parameter Value	Associated Configuration File
`enabler-url`	`http://localhost:7501/`	This parameter is located in the c-enabler for RosettaNet configuration file. For instructions on updating this file, see "Configuring the C-Enabler for RosettaNet" on page 9.
`User`	`bea`	This parameter is located in the c-enabler for RosettaNet weblogic.properties file. For instructions on updating this file, see "Configuring Workflow Sessions" on page 17.
`Password`	`12345678`	This parameter is located in the c-enabler for RosettaNet weblogic.properties file. For instructions on updating this file, see "Configuring Workflow Sessions" on page 17.
`OrgName`	`BEA`	This parameter is located in the c-enabler for RosettaNet weblogic.properties file. For instructions on updating this file, see "Configuring Workflow Sessions" on page 17.

To run the RNConfig command script and start the example workflows, complete the following procedure:

1. On the command line, navigate to the c-enabler for RosettaNet subdirectory of your WebLogic Collaborate Enabler for RosettaNet installation, `<WLC_HOME>/rosettanet/enabler`.

2. Run the `RNConfig` command. WebLogic Collaborate Enabler for RosettaNet provides `RNConfig.cmd` for Windows and `rnconfig.sh` for UNIX.
3. Start WebLogic Process Integrator Studio. On the command line, navigate to `<WLC_HOME>/rosettanet/wlpi`.
4. Run the `Studio` command. WebLogic Collaborate Enabler for RosettaNet provides `Studio.cmd` for Windows and `studio.sh` for UNIX.
5. In the Studio login dialog box, enter the following information:
   - user: `bea`
   - password: `12345678`
   - server: `t3://localhost:7501`
   - organization: `BEA`
6. Make the PIP3A2_Customer template active by, open the PIP3A2_Customer template in the folder tree, right-click it, and select Properties.
7. Select the Active check box and save the workflow.
8. Start the workflow examples from the Worklist client that is provided with WebLogic Process Integrator (in `<WLC_HOME>/rosettanet/wlpi`).
9. Make sure both the c-hub and the c-enabler for RosettaNet are running.
10. Run the `Worklist` command. WebLogic Collaborate Enabler for RosettaNet provides `Worklist.cmd` for Windows and `worklist.sh` for UNIX
11. In the Worklist client: select Workflow, and then Start Workflow.
12. Select the PIP3A2_Customer workflow and click OK.

The c-enabler for RosettaNet `wlc.log` (in `<WLC_HOME>/rosettanet/enabler`) indicates when each workflow starts and completes. You can also use WebLogic Process Integrator Studio to monitor the workflow status.

Manual Configuration Process

This section provides instructions for manually configuring WebLogic Process Integrator to support the workflow examples. The first subsection, “High-Level Procedure for Configuring and Starting Workflow Examples” on page 30 is actually a summary of several low-level procedures you should follow to prepare and run the workflow examples. The low-level procedures are provided in the remaining subsections:

- Starting WebLogic Process Integrator Studio
- Defining Business Operations
- Creating Workflow Templates
- Importing Workflow Templates
- Running the Workflow Examples

High-Level Procedure for Configuring and Starting Workflow Examples

To define and run the PIP3A2 and PIP0A1 workflow examples, complete the following procedure:

1. Start WebLogic Process Integrator Studio. (See "Starting WebLogic Process Integrator Studio" on page 31.)

2. Define business operations. (See "Defining Business Operations" on page 31.) Business operations stored in the WebLogic Process Integrator database are not included in the XML definition of the sample workflows. All the sample workflows use business operations to call methods on Java classes. If the business operations are not defined in your database, you receive a warning message when importing the workflows. This warning indicates that the system cannot find the relevant business operation defined in the workflow template definition.

3. Create four new workflow templates: PIP3A2_Customer, PIP3A2_Supplier, PIP0A1_Notifier, and PIP0A1_Admin. (See "Creating Workflow Templates" on page 35.) The order in which you import the four workflows is important because PIP0A1_Notifier is implemented as a subworkflow. You must import and activate the PIP0A1_Notifier workflow template definition before importing the PIP3A2_Customer workflow template definition.

4. Import the workflow template definitions for PIP0A1_Notifier, PIP0A1_Admin, PIP3A2_Customer, and PIP3A2_Supplier. (See "Importing Workflow Templates" on page 36.)

5. Execute the workflows. (See"Running the Workflow Examples" on page 36.)

Starting WebLogic Process Integrator Studio

To start the WebLogic Process Integrator Studio:

1. On the command line, navigate to `<WLC_HOME>/rosettanet/wlpi`.
2. Run the `Studio` command. WebLogic Collaborate Enabler for RosettaNet provides `Studio.cmd` for Windows and `studio.sh` for UNIX.
3. In the Studio login dialog box, enter the following information:
   - user: `bea`
   - password: `12345678`
   - server: `t3://localhost:7501`
   - organization: `BEA`

For more information about defining users, roles, and organizations in WebLogic Process Integrator, see the *BEA WebLogic Process Integrator Studio Guide*.

Defining Business Operations

The RosettaNet workflows use several business operations to perform customized actions from within the workflows. A business operation represents a method call on an Entity/Session EJB or Java class instance. All the example business operations call methods on Java class instances.

For detailed instructions for adding business operations, see the topic Defining Business Operations in Defining the Order Processing Workflow in the *BEA WebLogic Process Integrator Tutorial*.

The business operations provided in the example include both general and PIP3A2-specific operations. The general business operations may be used, without modification, from within other PIP workflows to perform actions common across all PIP workflows. The PIP3A2-specific business operations may be leveraged for other PIP workflows but Java code modifications are required.

To add the example business operations, start the WebLogic Process Integrator Studio (see "Starting WebLogic Process Integrator Studio" on page 31), choose first Configuration and then Business Operations from the menu, and, finally, add the business operations listed in the following table. You must use the exact name of each business operation as specified in the table. (You are not required to update the parameter names but you can improve readability by doing so.)

Table 6 Names and Details for General Business Operations

General Business Operation	Details
`Log`	*Full Java class name:* `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `void log (String p0)` *Parameter name*: `message` *Description*: Logs the provided message.
`Add Document Type`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `String addDocType (String p0, String p1)` *Parameter names*: `xmlStringDoc`, `documentType` *Description*: Adds the `DOCTYPE` to the given XML document string.
`Validate Header`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean validateHeader (String p0, String p1)` *Parameter names*: `xmlStringDoc`, `schemaFileName` *Description*: Validates the Service Header according to the given XML schema file. The XML schema has been created, using the Service Header message guideline, and it is used to validate the XML document representation of the Service Header.

Table 6 Names and Details for General Business Operations (Continued)

General Business Operation	Details
`Validate Content`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean validateContent String p0, String p1)` *Parameter names*: `xmlStringDoc`, `schemaFileName` *Description*: Validates the Service Content according to the given XML schema file. The XML schema has been created, using the PIP specific message guideline, and it is used to validate the XML document representation of the PIP-specific Service Content.
`Prepare Receipt Exception`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean prepareReceiptException (String p0)` *Parameter name*: `workflowInstanceId` *Description*: Prepares the receipt acknowledgement exception message (including both header and content). Because the message is built correctly in the workflow it is not necessary to validate it.
`Is this a GeneralException Message?`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean generalExceptionMessage (String p0)` *Parameter name*: `headerXmlString` *Description*: Returns indication of whether the provided service header is associated with a general exception message.

Table 6 Names and Details for General Business Operations (Continued)

General Business Operation	Details
`Process Receipt Acknowledgement`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean processReceiptAcknowledgement (String p0)` *Parameter name*: `workflowInstanceId` *Description*: Processes the receipt acknowledgement message. Processing includes validation of the Service Header and determination of the message type.
`Prepare Receipt Acknowledgement`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `boolean prepareReceiptAcknowledgement (String p0)` *Parameter name*: `workflowInstanceId` *Description*: Prepares the receipt acknowledgement message (including both header and content). Because the message is built correctly in the workflow it is not necessary to validate it.
`Load XML Doc String`	*Full Java class name*: `com.bea.b2b.rosettanet.RNHelper` *Method to call*: `String getServiceContentString (String p0)` *Parameter name*: *`filename`* *Description*: Retrieves a parsed version of the service content for the given PIP.

The following table describes the Names and Details for PIP3A2-Specific Business Operations.

Table 7 Names and Details for PIP3A2-Specific Business Operations

PIP3A2-Specific Business Operation	Details
`Prepare Query`	*Full Java class name*: `com.bea.b2b.rosettanet.pip3a2.PrepareQuery` *Method to call*: `boolean manipulate (String p0)` *Parameters*: `workflowInstanceId` *Description*: Saves and sets the necessary template variables in the given workflow instance.
`Validate Query`	*Full Java class name*: `com.bea.b2b.rosettanet.pip3a2.ValidateQuery` *Method to call*: `boolean manipulate (String p0)` *Parameters*: `workflowInstanceId` *Description*: Saves variables from the message received in the workflow variables.
`Prepare Response`	*Full Java class name*: `com.bea.b2b.rosettanet.pip3a2.PrepareResponse` *Method to call*: `boolean manipulate (String p0)` *Parameters*: `workflowInstanceId` *Description*: Sets the necessary values in the response message from template variables in the given workflow instance.

For more information, see “Descriptions of Business Operations” on page 79.

Creating Workflow Templates

For detailed instructions for creating new workflow template definitions, see the topic Creating a New Workflow Template Definition in Defining the Order Processing Workflow in the *BEA WebLogic Process Integrator Tutorial*.

To create template definitions for the example, go in the folder tree on the left side of the main Studio screen, right-click the Templates folder, and choose Create Template. Give the templates the same names as the files in the `<WLC_HOME>/rosettanet/templates` directory: PIP0A1_Notifier, PIP0A1_Admin, PIP3A2_Customer, and PIP3A2_Supplier.

Importing Workflow Templates

For detailed instructions for importing workflow template definitions, see the topic Importing the Start Order Processing Workflow Template Definition in Defining the Start Order Processing Workflow in the *BEA WebLogic Process Integrator Tutorial*.

To import the PIP0A1_Notifier template from the `<WLC_HOME>/rosettanet/templates` directory and make it active:

1. Right-click the PIP0A1_Notifier template you just created and select Import Template.
2. Traverse to the `<WLC_HOME>/rosettanet/templates` directory.
3. Click Save.
4. WebLogic Process Integrator displays a message to indicate that it is resolving the Business Operation names. Click OK.
5. Open the template in the folder tree, right-click it, and select Properties.
6. Select the Active check box and save the workflow. This template must be active before importing the PIP3A2_Customer template because it is a subworkflow of the PIP3A2_Customer workflow.

Import the other workflow templates from the `<WLC_HOME>/rosettanet/templates` directory, using the procedure described above. Make sure you set all the templates to Active and that you save the workflows. The templates must be active in order for the workflows to run.

Running the Workflow Examples

To start the workflow examples from the Worklist client that is provided with WebLogic Process Integrator (in `<WLC_HOME>/rosettanet/wlpi`), complete the following procedure:

1. Make sure both the c-hub and the c-enabler for RosettaNet are running.
2. In the Worklist client: select Workflow, and then Start Workflow.
3. Select the PIP3A2_Customer workflow and click OK.

The c-enabler for RosettaNet `wlc.log` (in `<WLC_HOME>/rosettanet/enabler`) indicates when each workflow starts and completes. You can also use WebLogic Process Integrator Studio to monitor the workflow status.

For detailed instructions for the tasks described here, see the following documents:

- For executing workflows using the Worklist client, see Executing the Workflow Example in the *BEA WebLogic Process Integrator Tutorial*.
- For checking the workflow status, see Executing the Workflow Example in the *BEA WebLogic Process Integrator Tutorial*.

Walkthrough of the Workflow Examples

The example workflows interact through XML messaging. This section describes how to define these workflows and how you can implement your own PIP using the c-enabler for RosettaNet. These descriptions are based on the following workflow examples:

- PIP3A2_Customer
- PIP3A2_Supplier
- PIP0A1_Notifier
- PIP0A1_Admin

Getting Started

This section provides information designed to help you learn the fundamentals of PIP workflows before walking through an individual example.

Recommended Reading

The following RosettaNet documents are recommended reading if you want to fully understand the example workflow, and required reading if you want to implement your own PIP using the c-enabler for RosettaNet. All documents are available in the "Standards" section at the following URL: `http://www.rosettanet.org`.

- RosettaNet Implementation Framework (RNIF) v1.1—The RNIF is an open, common networked-application framework designed to allow RosettaNet Supply Chain and Solution Partners to collaborate in executing RosettaNet PIPs.
- RNIF Technical Advisories—RNIF Technical advisories are updates and additional information for RNIF v1.1.
- RNIF Technical Recommendations—Technical Recommendations describe features or enhancements not yet available in a published version of the RNIF v1.1. Implementation of Technical Recommendations is optional.
- RNIF Business Signals, Service Header & Preamble—The RNIF Business Signals, Service Header & Preamble contains message guidelines and XML document type definitions (DTDs) for the RNIF Business Signals, Service Header and Preamble.
- Understanding a PIP Blueprint—Reference for PIP blueprint components and evaluation. Is available under Supporting Documents in the Standards Section.
- PIPs of interest—PIPs are specialized system-to-system XML-based dialogs that define business processes between supply chain companies. Each PIP includes a technical specification based on the RosettaNet Implementation Framework (RNIF), a Message Guideline document with a PIP-specific version of the Business Dictionary, and XML document type definitions (DTDs) for the PIP-specific messages.

Creating New PIP Workflow Templates

Over ninety percent of the current PIPs have the same PIP Business Process Flow as either PIP3A2 or PIP0A1. The PIP Business Process Flow is defined in the PIP technical specification. There are two approaches you can take when implementing your own PIP.

The first approach is to create the PIP workflows from scratch using the PIP3A2 or PIP0A1 workflows as examples. The *BEA WebLogic Process Integrator Tutorial* provides a detailed example of *how* to define workflows using the features of WebLogic Process Integrator Studio. The PIP3A2 and PIP0A1 workflows provide examples of *what* to define in the workflows. Specifically:

- What nodes to create
- What nodes to connect together
- What actions to define in the nodes

The second approach is to import the example workflows and change the existing nodes and workflow variables to suit the requirements of the PIP your are implementing. The first approach is recommended if you are *not* experienced in implementing PIP workflows with WebLogic Process Integrator Studio.

Defining Workflow Variables

A workflow variable is used to store application-specific information required by a workflow at run time. This information is often used to control the logic within a workflow and it may be used by business operations. If you are implementing a new PIP you must create and use workflow variables.

Workflow variables are used extensively throughout the example workflows. Each variable name is proceeded with a $ when used in the workflow. The variables used in the example workflows are discussed in the remainder of this section. The variables used by the example business operations are discussed in “Descriptions of Business Operations” on page 79.

For more information about defining workflow variables, see the topic Defining WebLogic Process Integrator Variables for Workflows in Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

PIP3A2_Customer

To understand the details of the PIP3A2_Customer workflow, see *PIP Specification, PIP3A2: Query Price and Availability*. For information about exception handling in PIPs, see *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

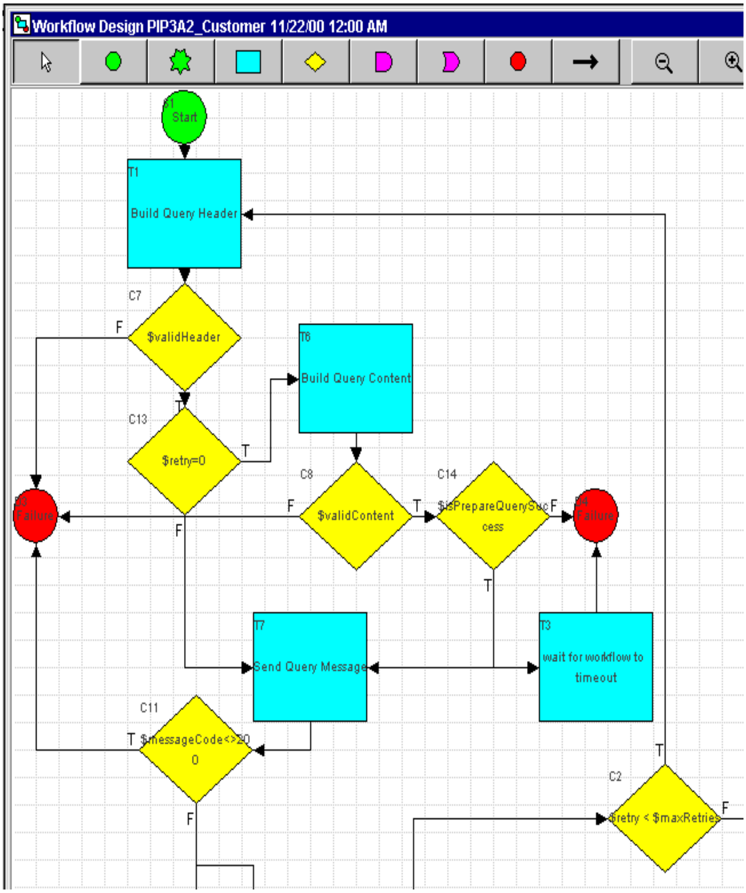
Before starting this workflow, the content of the query message should be available. In the example template the content of the query message is read from the `<WLC_HOME>/rosettanet/enabler/3A2PriceAndAvailabilityQueryMessage.xml` file. In a production environment, the query message would be generated by a private workflow or backend system.

In PIP3A2 the Customer role always initiates the PIP. Typically the workflow for the Customer portion of PIP3A2 is triggered by a backend system programmatically or by another, private workflow.

The following sections present the workflow diagram for each example, as well as a description of each node in the workflow diagram.

Note: Where the workflow diagram is too large to be accommodated in a single figure, it is divided into parts.

The following figure shows the first part of the PIP3A2_Customer workflow.



Note: The variables associated with each workflow are displayed in a subfolder under the workflow template folder. To view the properties for a particular variable, right-click the variable name in the folder.

Start

The start node logs the start of the workflow and initializes several workflow variables that are used throughout the workflow. The Log business operation may be used by other PIP workflows. The variables `fromDUNS`, `toDUNS`, `fromPartnerClassCode`, and `toPartnerClassCode` should be set to the appropriate values to reflect the information of the actual trading partners as defined in the c-enabler for RosettaNet configuration files. See “Configuring the C-Enabler for RosettaNet” on page 9. The variables that are shown in upper case are constants that refer to schemas and XML files. They should not be modified for the example, but they must be defined when a new PIP workflow is being implemented. Many of the other variables are specific to PIP3A2 and do not require modification. The workflow template provides descriptions of all variables. The following table describes all the variables initialized in the Start node and shows the values with which they are initialized.

Table 8 Variables Initialized in the Start Node

Variable	Initialization Value	Description
`fromDUNS`	`123456789`	The unique business identifier of the trading partner sending the message. The number is specified by RosettaNet. This number must match the trading partner value in the c-enabler for RosettaNet configuration file. The variable is used when building the service header.
`toDUNS`	`987654321`	The unique business identifier of the trading partner sending the message. The number is specified by RosettaNet. This number must match the trading partner value in the c-enabler for RosettaNet configuration file. The variable is used when building the service header.
`initiatingDUNS`	`$fromDUNS`	The unique business identifier of the trading partner that starts the PIP conversation. It is initialized with the value of the *`fromDUNS`* variable.The variable is used when building the service header.
`fromPartnerClassCode`	`Shopper`	Code that identifies a partner’s function in the supply chain. The partner must determine the code, it is not specified in the PIP technical specification. The variable is used when building the service header.
`toPartnerClassCode`	`Retailer`	Code that identifies a partner’s function in the supply chain. The partner must determine the code, it is not specified in the PIP technical specification. The variable is used when building the service header.

Table 8 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`processCode`	`Query Price and Availability`	This code is the name of a PIP specification document. The variable is used when building the service header.
`processIndicatorCode`	`3A2`	This is the classification of the PIP specification by cluster identifier and segment identifier. The variable is used when building the service header.
`processVersionId`	`1.3`	Version of the PIP. The variable is used when building the service header.
`retry`	`0`	The number of times the message has tried to be sent. The variable is used by a decision node in the workflow.
`timeout`	`false`	Timeout status. The variable is used by a decision node in the workflow.
`reason`	`SUCCESS`	Reason for failure. Updated by various nodes in the workflow if a failure occurs.
`processInstanceId`	`Date()`	A unique alpha-numeric identifier for the process instance. The variable is initialized by using the `Date()` function in Studio. The variable is used when building the service header.
`transactionInstanceId`	`Date()`	A unique alpha-numeric identifier for the transaction instance. The variable is initialized by using the `Date()` function in Studio. The variable is used when building the service header.
`actionInstanceId`	`Date()`	A unique alpha-numeric identifier for the action instance. The variable is initialized by using the `Date()` function in Studio. The variable is used when building the service header.
`businessActionCode`	`Price and Availability Query`	A phrase that specifies a business action. It is specified in the PIP technical specification. The variable is used when building the service header.

Table 8 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`EXCEPTION_RECEIPT_SCHEMA`	`ReceiptAcknowledgementExceptionMessageGuideline.xsd`	The name of the Receipt Acknowledgement Exception Message Guideline schema file. The variable is not used by this example workflow.
`HEADER_SCHEMA`	`ServiceHeaderPartMessageGuideline.xsd`	The name of the Service Header Part Message Guideline schema file. The variable is used by the Process Receipt Acknowledgement business operation and by the Build Query Header and Validate Response Header tasks.
`PRICE_QUERY_SCHEMA`	`3A2PriceAndAvailabilityQueryMessageGuideline_v1_3.xsd`	The name of the Query Message schema file. The variable is used by the Build Query Content task.
`PRICE_RESPONSE_3A2`	`3A2PriceAndAvailabilityResponseMessage.xml`	The name of the ResponseMessage file. The variable is not used by this example workflow.
`PRICE_RESPONSE_SCHEMA`	`3A2PriceAndAvailabilityResponseMessageGuideline_v1_3.xsd`	The name of the Response Message schema file. The variable is used by the Validate Response Content task.
`QUERY_PRICE_3A2`	`3A2PriceAndAvailabilityQueryMessage.xml`	The name of the Query Message file. The variable is used by the Build Query Content task.
`RECEIPT_ACKNOWLEDGE`	`ReceiptAcknowledgementMessage.xml`	The name of the Receipt Acknowledgement Message file. The variable is not used by this example workflow.
`RECEIPT_EXCEPTION`	`ReceiptAcknowledgementExceptionMessage.xml`	The name of the Receipt Acknowledgement Exception Message file. The variable is not used by this example workflow.
`RECEIPT_SCHEMA`	`ReceiptAcknowledgementMessageGuideline.xsd`	The name of the Receipt Acknowledgement Exception Message Guideline schema file. The variable is not used by this example workflow.

Table 8 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`DOCUMENT_TYPE`	`<!DOCTYPE ServiceHeader SYSTEM \”ServiceHeaderPartMessageGuideline.dtd\”>`	The `DOCTYPE` line that needs to be inserted in the service header XML string. The service header is a String variable that contains the service header XML structure. There is no way to put the `DOCTYPE` in the structure via WebLogic Process Integrator so it is inserted with the Add Document Type business operation after the structure is built. The variable is used by the Build Query Header task.
`EXCEPTION_DTD`	`<!DOCTYPE ServiceHeader SYSTEM \”GeneralExceptionMessageGuideline.dtd\”>`	The `DOCTYPE` line for the General Exception Message. The variable is not used by this example workflow.
`msgProblem`	`false`	Result of the Process Receipt Acknowledgement business operation. The variable is used by a decision node in the workflow.
`fromRoleClassCode`	`Customer`	Code identifying the party’s role in the supply chain. This value is specified in the PIP technical specification. The variable is used when building the service header.
`toRoleClassCode`	`Product Supplier`	Code identifying the party’s role in the supply chain. This value is specified in the PIP technical specification. The variable is used when building the service header.
`transactionCode`	`Query Price and Availability`	The code is the name of the business activity and the transaction dialog in the PIP technical specification. The variable is used when building the service header.
`fromBusinessServiceCode`	`Customer Service`	The unique identity of a business service network component. The code is defined in the PIP technical specification. The variable is used when building the service header.

Table 8 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`toBusinessServiceCode`	`Product Supplier Service`	The unique identity of a business service network component. The code is defined in the PIP technical specification. The variable is used when building the service header.
`documentFunctionCode`	`Request`	Code identifying the function of a document as either a request or a response. The variable is used when building the service header.
`ackTimeout`	`2`	The amount of time in which a receipt acknowledgement must be received. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
`workflowTimeout`	`24`	The maximum amount of time the Customer workflow will wait for the Response message before timing out. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
`maxRetries`	`3`	The maximum number of times the workflow can try to resend the message. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
`timeUnit`	`h`	The time unit used when specifying timeouts. The variable is used in the Wait for Timeout task.
`actionVersionId`	`1.3`	Version identifier of the action. The variable is used when building the service header.

Task: Build Query Header

This task checks whether this is a retry to build this message. If this is the first time, the transaction identifier in the service header is initialized. If this is a retry the transaction identifier is unchanged. The service header XML of the message is built as an XML structure and assigned to the workflow string variable `actionHeader`. The document type is added, using the Add Document Type business operation, because building an XML document in WebLogic Process Integrator does not allow a document type to be entered. The header is then validated, using the Validate Header business operation, against an XML schema that enforces the Service Header message guidelines.

Decision: validHeader?

The decision node determines whether the service header was validated successfully against the schema. If the header is not valid, the reason is logged, using the Log business operation, and the workflow ends.

Decision: retry = 0

This variable represents the number of times the c-enabler for RosettaNet has tried to send the message. If the value of the retry variable is 0, this indicates that this is the first time the c-enabler for RosettaNet has tried to send the message, and this is when the service context is created. If value of the retry variable is not 0, the previous context is reused.

Task: Build Query Content

The task loads the Query message from the `<WLC_HOME>/rosettanet/enabler/3A2PriceAndAvailabilityQueryMessage.xml` file, using the Load XML Doc String business operation. In a real deployment this step could be replaced with a business operation that retrieves the service content from a backend system. The task calls the business operation PrepareQuery, which saves data from the Query message into workflow variables that will be used for later messages. It also generates the document id and timestamp for the message. The service content is then validated, using the Validate Content business operation, against the XML schema that enforces the PIP3A2 query message guidelines.

The `com.bea.b2b.rosettanet.pip3a2.PrepareQuery` business operation is specific to this PIP and role. If you define a workflow for a different PIP, you need to develop a new business operation to perform the same functions as the PrepareQuery business operation for the data specific to the new PIP.

Decision: validContent?

This decision node determines whether or not the service content was validated successfully against the schema. If the message content is not valid the reason is logged, using the Log business operation and the workflow ends.

Decision: isPrepareQuerySuccess

This decision node determines whether the result of the Prepare Query business operation was successful. If it was successful, the flow continues to the Send Query Message task. If the Prepare Query business operation was not successful, then the workflow ends with a failure.

Task: Send Query Message

The RosettaNet message is sent using the built service header and service content by the Send RosettaNet Message action.

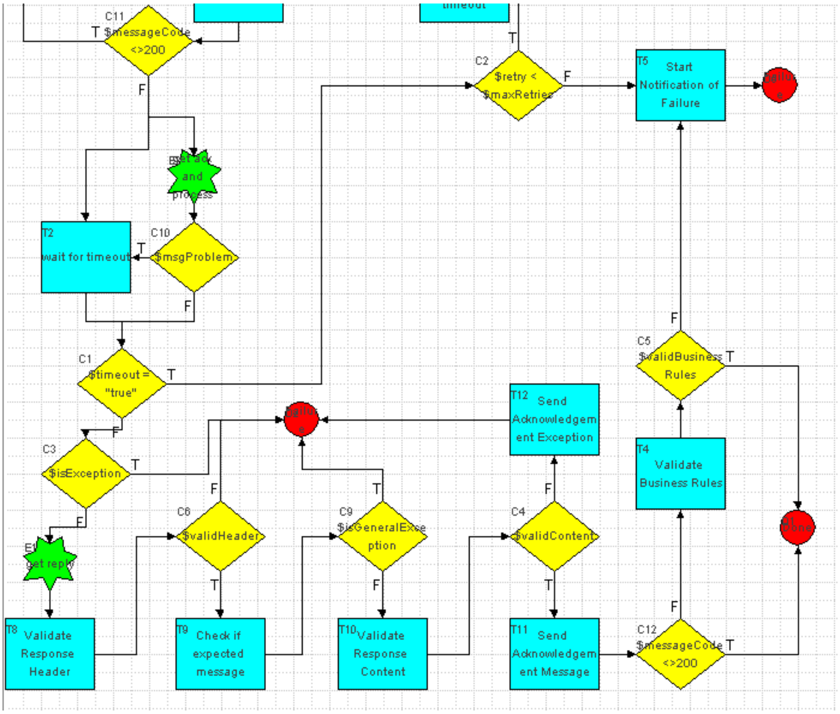
Task: Wait for Workflow to Timeout

This task sets a task due date to the time specified by the *`workflowTimeout`* variable. In the example template this is set to 24 hours in the Start node. According to the PIP Specification, the supplier has 24 hours to respond to the query message that was sent. This value is defined in the PIP technical specification and may be different or nonexistent for other PIPs.

Decision: messageCode<>200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation, and the workflow ends.

The following figure shows the second part of the PIP3A2_Customer workflow.



Task: Wait for Timeout

If the query message was sent successfully, according to the PIP specification, the supplier has two hours to send a receipt of acknowledgement message. This task sets a task due date to two hours. This value is defined in the PIP technical specification and may be different or nonexistent for other PIPs.

Event: Get Ack and Process

This event waits for an acknowledgement from the supplier. This message could be a Receipt Acknowledgement, a Receipt Acknowledgement Exception, or a General Exception. When the message is received, the business operation Process Receipt Acknowledgement is called to determine the kind of message and validate the message according to the appropriate message guidelines. The business operation returns an indication of whether the message has validation problems. This result is assigned to the *msgProblem* workflow variable.

Decision: msgProblem?

This decision evaluates, based on the outcome of the previous task, whether there is a validation problem with the message. If the message received cannot be validated, then according to the *RNIF Technical Advisory #1*, it should be treated like a timeout. In this case, the workflow enters the Wait for Timeout task.

Decision: timeout=”true”?

This decision determines whether the two-hour timeout task has expired. If a timeout occurred, a message is logged using the Log business operation and the workflow continues to check whether the message should be retried.

Decision: retry < $maxRetries?

This decision node determines whether the retry variable has reached the *maxRetries* variable value. For this PIP, the message can be sent up to four times because the *retry* variable is initialized to 0 and the *maxRetries* variable is initialized to 3 in the Start node. If the message has not been retried the maximum number of times, the retry count is incremented and the workflow goes back to rebuild and resend the query message. If the maximum number of retries is reached the *reason* variable is updated with an error message. The maximum number of retries is specified in the PIP technical specification and may be different or nonexistent for other PIP workflows.

Decision: isException?

This decision node evaluates the boolean variable `isException` set in the Process Receipt Acknowledgement business operation, which indicates whether the message received from the supplier was a Receipt Acknowledgement Exception, or a General Exception. If the message was an exception message, the workflow ends. If the message was a Receipt Acknowledgement, the workflow continues.

Event: Get Reply

This event waits for the Reply message from the supplier. This event also saves the response service content, represented by the `serviceContent` workflow variable, in the workflow variable `responseContent`. The header is saved in the `actionHeader` workflow variable. After the workflow is complete, these variables can be accessed for processing on the reply message. For example, the `responseContent` variable can be processed to obtain the price and availability values that were requested.

Task: Validate Response Header

This task validates the header of the message received from the supplier, using the Validate Header business operation, against the XML schema, which implements the Service Header message guidelines.

Decision: validHeader?

This decision node determines whether the header conformed to the message guidelines. If the header was not valid, a message is logged using the Log business operation and this workflow ends, which causes the supplier workflow to time out waiting for an acknowledgement response.

Task: Check if Expected Message

This task checks the kind of message that was received from the supplier, using the `Is this a GeneralExceptionMessage` business operation. The supplier could have sent a reply or a General Exception message.

Decision: isGeneralException?

This decision node checks whether the message received from the supplier was a General Exception message. If the message is an exception message a message is logged using the Log business operation and the workflow ends. If not, the workflow continues.

Task: Validate Response Content

This task validates the reply message, using the Validate Content business operation, against the XML schema that implements the message guidelines. It also saves the business action code from the header of the reply message into the workflow variable `responseBusinessActionCode`. The value is obtained from the `actionHeader` variable by using an XPath expression. This value is used in the receipt acknowledgement messages.

Decision: validContent?

This decision node checks whether the reply message was valid. If the reply is not valid, a message is logged using the Log business operation, the `reason` variable is updated with an error message and the workflow sends a Receipt Acknowledgement Exception. If the message is valid, the workflow sends a Receipt Acknowledgement.

Task: Send Acknowledgement Exception

This task updates the `reason` variable with an error message. The task then builds the header for the Receipt Acknowledgement Exception and assigns it to the workflow variable `signalHeader`. It then calls the business operation Prepare Receipt Exception to build the message. The task then sends the Acknowledgement Receipt Exception message to the supplier. When this task completes, the workflow is finished.

Task: Send Acknowledgement Message

This task saves the `fromDocId` and `fromDocTimeStamp` values from the service content to be used in the acknowledgement message. The values are obtained from the `serviceContent` variable by using XPath expressions. It then builds the header for the receipt acknowledgement and assigns it to the workflow variable `signalHeader`. It then calls the business operation Prepare Receipt Acknowledgement, which continues to build the message. The message is then sent to the supplier.

Decision: messageCode<>200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation and the workflow ends.

Task: Validate Business Rules

This task is a placeholder for any additional business rules that may need to be checked against the reply message. Currently only the `validBusinessRules` variable is set to true.

Decision: validBusinessRules?

This decision determines whether the business rules are valid. If the business rules are valid, the workflow is complete. If not, a message is logged using the Log business operation and the `reason` variable is updated with an error message. Then according to the *RNIF Technical Advisory #1*, the Notification of Failure PIP is started.

Task: Send Notification of Failure

This task calls the sub-workflow PIP0A1_Notifier with the data appropriate for sending a failure message. The following table lists the parameters that are passed to the PIP0A1_Notifier workflow and populated with the PIP3A2_Customer workflow variable values.

Table 9 Parameters for Send Notification of Failure

PIP0A1_Notifier Parameter	PIP3A2_Customer Variable
`actionInstanceId`	`actionInstanceId`
`actionVersionId`	`actionVersionId`
`businessActionCode`	`businessActionCode`
`failureReason`	`reason`
`fromDUNS`	`fromDUNS`

Table 9 Parameters for Send Notification of Failure (Continued)

PIP0A1_Notifier Parameter	PIP3A2_Customer Variable
fromPartnerClassCode	fromPartnerClassCode
fromRoleClassCode	fromRoleClassCode
fromSupplyChainCode	fromSupplyChainCode
initiatingDUNS	initiatingDUNS
processCode	processCode
processIndicatorCode	processIndicatorCode
processInstanceId	processInstanceId
processVersionId	processVersionId
toDUNS	toDUNS
toPartnerClassCode	toPartnerClassCode
toRoleClassCode	toRoleClassCode
toSupplyChainCode	toSupplyChainCode
transactionCode	transactionCode
transactionInstanceId	transactionInstanceId

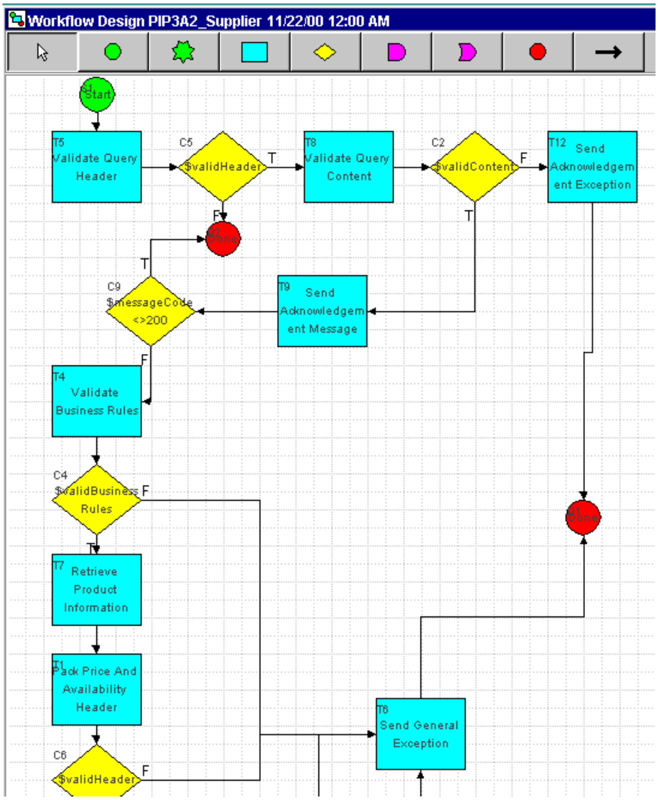
Done

This node logs the completion of the workflow, using the Log business operation.

PIP3A2_Supplier

To further understand this workflow, see *PIP Specification, PIP3A2: Query Price and Availability*. For more information about exception handling, see *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

The following illustration shows the first part of the PIP3A2_Supplier workflow.


Workflow Design PIP3A2_Supplier 11/22/00 12:00 AM
S1 Start
T5 Validate Query Header
C5 $validHeader
T
F
T8 Validate Query Content
C2 $validContent
F
T
T12 Send Acknowledgement Exception
Done
T
C9 $messageCode <>200
T9 Send Acknowledgement Message
F
T4 Validate Business Rules
C4 $validBusinessRules
F
T
Done
T7 Retrieve Product Information
T1 Pack Price And Availability Header
C6 $validHeader
F
T6 Send General Exception

Start

The workflow is started upon receipt of a RosettaNet message. The start node logs the workflow started, using the Log business operation, and initializes several workflow variables that are used throughout the workflow. The values of the following variables should be modified to accommodate the supplier:

- `fromDUNS`
- `fromContactName`
- `fromEmailAddress`

- `fromTeleNo`
- `fromPartnerClassCode`

The `fromDUNS` variable should be set to a value appropriate for the actual trading partner, as defined in the c-enabler for RosettaNet configuration files. See "Configuring the C-Enabler for RosettaNet" on page 9. The variables shown in all upper case are constants that refer to schemas and XML files. They should not be modified. Many other variables are specific to PIP3A2 and do not need to be modified. The workflow template provides a description of every variable.

The following table describes all the variables initialized in the Start node and shows the values with which they are initialized.

Table 10 Variables Initialized in the Start Node

Variable	Initialization Value	Description
`fromDUNS`	`987654321`	The unique business identifier of the trading partner sending the message. The number is specified by RosettaNet. This number must match the trading partner value in the c-enabler for RosettaNet configuration file. The variable is used when building the service header.
`fromContactName`	`Jim Smith`	Contact name. This variable is used by the Prepare Receipt Exception business operation.
`fromEmailAddress`	`jsmith@mycompany.com`	Contact email address. This variable is used by the Prepare Receipt Exception business operation.
`fromTeleNo`	`9871234567`	Contact telephone number. This variable is used by the Prepare Receipt Exception business operation.
`fromRoleClassCode`	*`Product Supplier`*	Code identifying the party's role in the supply chain. This value is specified in the PIP technical specification. The variable is used when building the service header.
`fromPartnerClassCode`	`Manufacturer`	Code that identifies a partner's function in the supply chain. The partner must determine the code, it is not specified in the PIP technical specification. The variable is used when building the service header.

Table 10 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
fromSupplyChainCode	Information Technology	Code identifying the supply chain for the partner's function. This variable is not used in the workflow.
toRoleClassCode	Customer	Code identifying the party's role in the supply chain. This value is specified in the PIP technical specification. The variable is used when building the service header.
processCode	Query Price and Availability	This code is the name of a PIP specification document. The variable is used when building the service header.
timeout	false	Timeout status. The variable is used by a decision node in the workflow.
reason	SUCCESS	Reason for failure. Updated by various nodes in the workflow if a failure occurs.
EXCEPTION_RECEIPT_SCHEMA	ReceiptAcknowledgementExceptionMessageGuideline.xsd	The name of the Receipt Acknowledgement Exception Message Guideline schema file. The variable is not used by this example workflow.
DOCUMENT_TYPE	<!DOCTYPE ServiceHeader SYSTEM \"ServiceHeaderPartMessageGuideline.dtd\">	The DOCTYPE line that needs to be inserted in the service header XML string. The service header is a String variable that contains the service header XML structure. There is no way to put the DOCTYPE in the structure via WebLogic Process Integrator so it is inserted with the Add Document Type business operation after the structure is built. The variable is used by the Send General Exception and Pack Price and Availability Header tasks.
HEADER_SCHEMA	ServiceHeaderPartMessageGuideline.xsd	The name of the Service Header Part Message Guideline schema file. The variable is used by the Process Receipt Acknowledgement business operation and by the Validate Query Header and Pack Price and Availability Header tasks.

Table 10 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
PRICE_QUERY_SCHEMA	3A2PriceAndAvailabilityQueryMessageGuideline_v1_3.xsd	The name of the Query Message schema file. The variable is used by the Validate Query Content task.
PRICE_RESPONSE_3A2	3A2PriceAndAvailabilityResponseMessage.xml	The name of the ResponseMessage file. The variable is used by the Pack Price and Availability Content task.
PRICE_RESPONSE_SCHEMA	3A2PriceAndAvailabilityResponseMessageGuideline_v1_3.xsd	The name of the Response Message schema file. The variable is used by the Pack Price and Availability Content task.
QUERY_PRICE_3A2	3A2PriceAndAvailabilityQueryMessage.xml	The name of the Query Message file. The variable is not used by this example workflow.
RECEIPT_ACKNOWLEDGE	ReceiptAcknowledgementMessage.xml	The name of the Receipt Acknowledgement Message Guideline schema file. The variable is not used by this example workflow.
RECEIPT_EXCEPTION	ReceiptAcknowledgementExceptionMessage.xml	The name of the Receipt Acknowledgement Exception Message file. The variable is not used by this example workflow.
RECEIPT_SCHEMA	ReceiptAcknowledgementMessageGuideline.xsd	The name of the Receipt Acknowledgement Exception Message Guideline schema file. The variable is not used by this example workflow.

Table 10 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`EXCEPTION_DTD`	`<!DOCTYPE ServiceHeader SYSTEM \”GeneralExceptionMessageGuideline.dtd\”>`	The DOCTYPE line for the General Exception Message. The variable is used in the Send General Exception task.
`processIndicatorCode`	`3A2`	This is the classification of the PIP specification by cluster identifier and segment identifier. The variable is used when building the service header.
`processVersionId`	`1.3`	Version of the PIP. The variable is used when building the service header.
`fromBusinessServiceCode`	`Product Supplier Service`	The unique identity of a business service network component. The code is defined in the PIP technical specification. The variable is used when building the service header.
`toBusinessServiceCode`	`Customer Service`	The unique identity of a business service network component. The code is defined in the PIP technical specification. The variable is used when building the service header.
`transactionCode`	`Query Price and Availability`	The code is the name of the business activity and the transaction dialog in the PIP technical specification. The variable is used when building the service header.
`businessActionCode`	`Price and Availability Query`	A phrase that specifies a business action. It is specified in the PIP technical specification. The variable is used when building the service header.
`documentFunctionCode`	`Response`	Code identifying the function of a document as either a request or a response. The variable is used when building the service header.

Table 10 Variables Initialized in the Start Node (Continued)

Variable	Initialization Value	Description
`ackTimeout`	`2`	The amount of time a receipt acknowledgement must be received in. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
`msgProblem`	`false`	Result of the Process Receipt Acknowledgement business operation. The variable is used by a decision node in the workflow.
`actionVersionId`	`1.3`	Version identifier of the action. The variable is used when building the service header.

Task: Validate Query Header

This task validates the header of the message received, using the Validate Header business operation, against the XML schema that enforces the Service Header message guidelines.

Decision: validHeader?

This decision node determines whether the header was validated successfully. If the header is not valid, then the workflow logs a message, using the Log business operation and ends, which means that the customer workflow will time out waiting for a response. If the header is valid, the workflow continues.

Task: Validate Query Content

This task saves some values from the query header into the following workflow variables:

- `responseBusinessActionCode`
- `toDUNS`
- `toPartnerClassCode`
- `initiatingDUNS`
- `retry`

- `transactionInstanceId`
- `actionInstanceId`

The variable values are obtained by using XPath expressions on the *`actionHeader`* workflow variable. The variables are used when building the service header. This task then validates the content of the message received, using the Validate Content business operation, against the XML schema that enforces the Query Message guidelines.

Decision: validContent?

This decision node determines whether the content was validated successfully. If the content is not valid, the *`reason`* variable is updated with an error message and a Receipt Acknowledgement Exception is sent. If the content is valid, a Receipt Acknowledgement is sent.

Task: Send Acknowledgement Exception

This task updates the *`reason`* variable with an error message and then builds the header for the Receipt Acknowledgement Exception. It then calls the business operation Prepare Receipt Exception, which continues to build the message. The message is then sent to the customer. When this task completes, the workflow is done.

Task: Send Acknowledgement Message

This task saves the query message in the `queryContent` workflow variable. It then calls the business operation Validate Query to save values from the content of the query message into workflow variables. The task then builds the header for the Receipt Acknowledgement and calls the business operation Prepare Receipt Acknowledgement, which continues to validate the header, build, and validate the content. The RosettaNet message is then sent to the supplier.

The `com.bea.b2b.rosettanet.pip3a2.ValidateQuery` business operation is specific to this PIP and role. If you define a workflow for a different PIP, you need to develop a new business operation to perform the same functions as the Validate Query business operation for the data specific to the new PIP.

Decision: messageCode<>200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation, and the workflow ends.

Task: Validate Business Rules

This task is a placeholder for any additional business rules that may need to be checked against the reply message. Currently only the `validBusinessRules` variable is set to true.

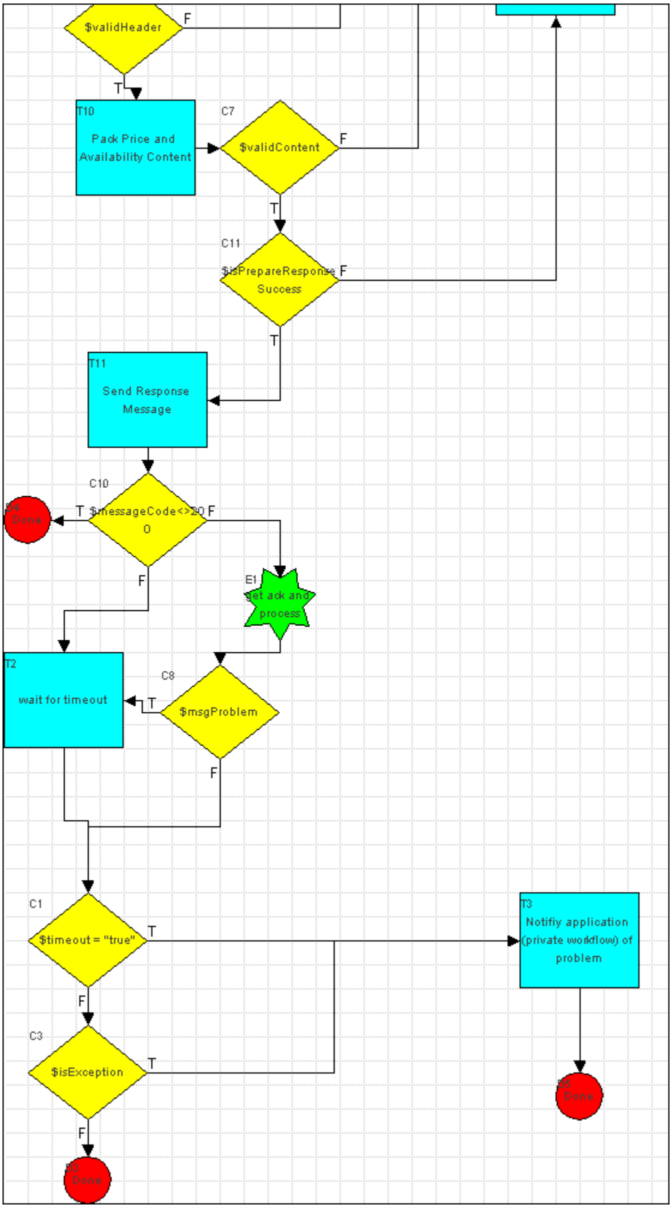
Decision: validBusinessRules?

This decision determines whether the business rules where valid. If not, the `reason` variable is updated with an error message. According to the *RNIF Technical Advisory #1*, a General Exception message is then sent.

Task: Retrieve Product Information

This task is a placeholder for the product information to be retrieved from a company database or other source. The received message is available in the workflow variables `actionHeader` and `queryContent`. These variables are the XML documents in String format. In the example template the product information is retrieved from the following XML file: `<WLC_HOME>/rosettanet/enabler/3A2PriceAndAvailabilityResponseMessage.xml`.

The following illustration shows the second part of the PIP3A2_Supplier workflow.



Task: Pack Price and Availability Header

The service header XML of the message is built and the document type is added on, using the Add Document Type business operation, because building an XML document in WebLogic Process Integrator does not allow a document type to be entered. The header is then validated, using the Validate Header business operation, against an XML schema that enforces the Service Header message guidelines.

Decision: validHeader?

The decision checks whether the service header was validated successfully against the schema. If the header is not valid, the problem is logged, using the Log business operation and the *reason* variable is updated with an error message.

Task: Pack Price and Availability Content

The task loads the Response message from the `<WLC_HOME>/rosettanet/enabler/3A2PriceAndAvailabilityResponseMessage.xml` file using the Load XML Doc String business operation. This file should contain the product items for which the supplier is returning information. The business operation Prepare Response is called to set the date timestamps and identifiers in the response document. The service content is then validated, using the Validate Content business operation, against the XML schema that enforces the PIP3A2 Response Message guidelines.

Decision: validContent?

The decision checks whether the service content was validated successfully against the schema. If the message content is not valid the problem is logged, using the Log business operation and the *reason* variable is updated with an error message.

Decision: isPrepareResponseSuccess?

The decision checks whether the Prepare Response business operation was successful. If it was, the response message is sent.

Task: Send General Exception

The signal header XML of the message is built and the document type is added on, using the Add Document Type business operation, because building an XML document in WebLogic Process Integrator does not allow a document type to be entered. The signal content XML is built and the document type is added on using the Add Document Type business operation. The general exception message is then sent to the customer.

Task: Send Response Message

The task sends the Response message using the built service header represented by the workflow variable `actionHeader` and service content represented by the workflow variable `serviceContent`.

Decision: messageCode <> 200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation, and the workflow ends.

Task: Wait for Timeout

If the response message is sent successfully, according to the PIP 3A2 specification, the supplier has two hours to send a Receipt of Acknowledgement message. This task sets a task due date of two hours using the workflow variable `ackTimeout`.

Event: Get Ack and Process

This event waits for an acknowledgement from the customer about the response message. This message could be a Receipt Acknowledgement, a Receipt Acknowledgement Exception, or a General Exception. When the message is received, the business operation Process Receipt Acknowledgement is called to determine the kind of message and validate the message according to the appropriate message guidelines. The business operation returns an indication of whether the message has validation problems. This result is assigned to the `msgProblem` workflow variable.

Decision: msgProblem?

This decision checks whether the message has a validation problem. If the message received cannot be validated, it should be treated like a timeout, according to the *RNIF Technical Advisory #1*. If a problem exists, the workflow enters the Wait for Timeout task.

Decision: timeout=”true”?

This decision checks whether the two-hour timeout task expired. If a timeout occurs, the `reason` variable is updated with an error message, and a message is logged, using the Log business operation.

Decision: isException?

This decision checks whether the message received from the supplier was a General Exception message. If the message is an exception message, a problem is logged, using the Log business operation. If not, the workflow ends.

Task: Notify Application (Private Workflow) of Problem

This task is a placeholder that can be used to notify any other applications of a problem. If this workflow is extended for use in a real world environment, additional actions should be added to handle the workflow problems.

Done

This node logs the completion of the workflow, using the Log business operation.

PIP0A1_Notifier

For more information about this workflow, see *PIP Specification, PIP0A1: Notification of Failure*. For details about exception handling, see *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*. This Technical Advisory specifies when a PIP should initiate PIP0A1.

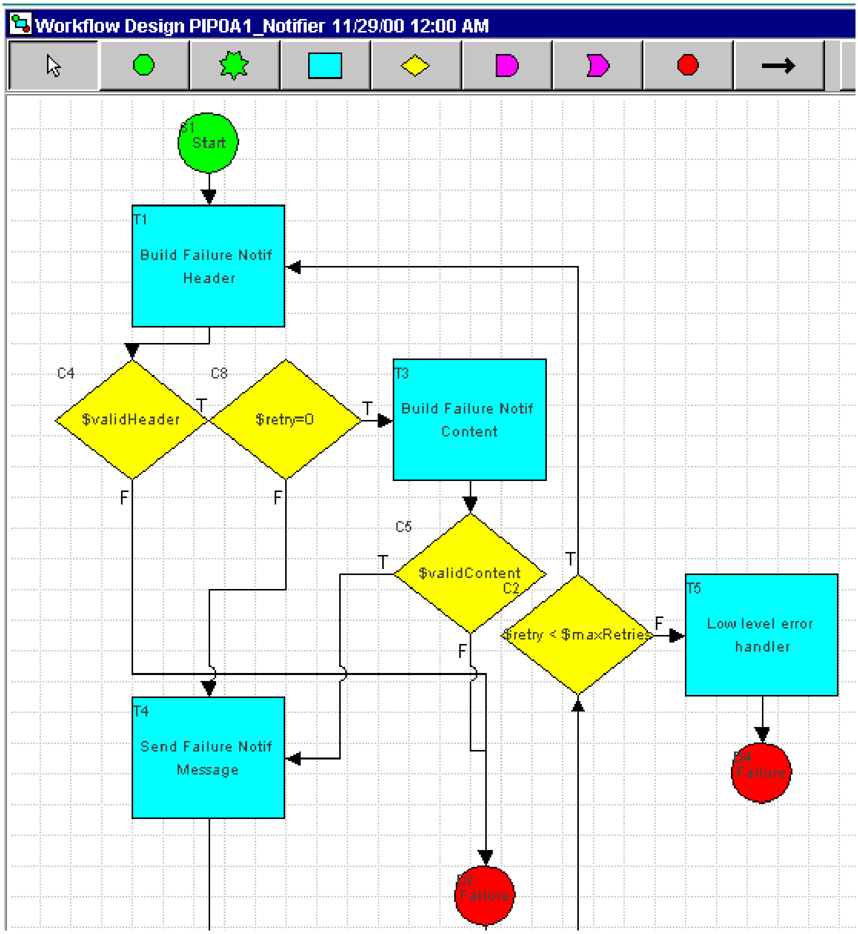
Workflow Inputs

This workflow is always started by another workflow. The initiating workflow must set the following variables in the Start Workflow action:

- `actionInstanceId`
- `actionVersionId`
- `businessActionId`
- `failureReason`
- `fromDUNS`
- `fromPartnerClassCode`
- `fromRoleClassCode`
- `fromSupplyChainCode`
- `initiatingDUNS`
- `processCode`
- `processIndicatorCode`
- `processInstanceId`
- `processVersionId`
- `toDUNS`
- `toPartnerClassCode`
- `toRoleClassCode`
- `toSupplyChainCode`
- `transactionCode`
- `transactionInstanceId`

These variables are used when the failure notification message header and content are built. Descriptions of them are provided in the workflow template. An example Start Workflow action is available in the PIP3A2_Customer workflow task Start Notification of Failure.

The following illustration shows the first part of the PIP0A1_Notifier workflow.


Workflow Design PIP0A1_Notifier 11/29/00 12:00 AM
S1
Start
T1
Build Failure Notif Header
C4
$validHeader
C8
$retry=0
T
T
F
F
T3
Build Failure Notif Content
C5
$validContent
C2
T
F
T
$retry < $maxRetries
F
T5
Low level error handler
S4
Failure
T4
Send Failure Notif Message
S2
Failure

Start

The start node logs the start of the workflow, using the Log business operation, and initializes workflow variables that are used throughout the workflow. The variable `adminDUNS` should be modified to the appropriate value. The variables shown in all upper case are constants that refer to XML schema files. Many of the other variables are specific to PIP0A1 and do not need to be modified. The following table describes all the variables initialized in the Start node and shows the values with which they are initialized.

Table 11 Variables Initialized in Start Node

Variable	Initialization Value	Description
`retry`	`0`	The number of times the message has tried to be sent. The variable is used by a decision node in the workflow.
`maxRetries`	`3`	The maximum number of times the workflow can try to resend the message. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
`timeout`	`false`	Timeout status. The variable is used by a decision node in the workflow.
`adminDUNS`	`999999993`	The unique business identifier of the trading partner sending the message. The number is specified by RosettaNet. This number must match the trading partner value in the c-enabler for RosettaNet configuration file. The variable is used when building the service header.
`DOCUMENT_TYPE`	`<!DOCTYPE ServiceHeader SYSTEM \”ServiceHeaderPartMessageGuideline.dtd\”>`	The DOCTYPE line that needs to be inserted in the service header XML string. The service header is a String variable that contains the service header XML structure. There is no way to put the DOCTYPE in the structure via WebLogic Process Integrator so it is inserted with the Add Document Type business operation after the structure is built. The variable is used by the Build Failure Notification Header task.
`HEADER_SCHEMA`	`ServiceHeaderPartMessageGuideline.xsd`	The name of the Service Header Part Message Guideline schema file. The variable is used by the Process Receipt Acknowledgement business operation and by the Build Failure Notification Header task.

Table 11 Variables Initialized in Start Node (Continued)

Variable	Initialization Value	Description
FAILURE_NOTIF_SCHEMA	0A1FailureNotificationMessageGuideline.xsd	The name of the Failure Notification Message schema file. The variable is used by the Build Failure Notification Content task.
ackTimeout	2	The amount of time a receipt acknowledgement must be received in. The value is defined in the PIP technical specification. The variable is used by a decision node in the workflow.
timeUnit	h	The time unit used when specifying timeouts. The variable is used in the Wait for Timeout task.
transactionInstanceId	Date()	A unique alpha-numeric identifier for the transaction instance. This variable is initialized with the Date() function in Studio. The variable is used when building the service header.
actionInstanceId	Date()	A unique alpha-numeric identifier for the action instance. This variable is initialized with the Date() function in Studio. The variable is used when building the service header.
processInstanceId	Date()	A unique alpha-numeric identifier for the process instance. This variable is initialized with the Date() function in Studio. The variable is used when building the service header.

Task: Build Failure Notif Header

This task first checks whether this is a retry to build the notification message. If this is the first time, then the transaction identifier in the service header is initialized. If this workflow is a retry, the task leaves the transaction identifier the same. The service header XML of the message is built and the document type is added, using the Add Document Type business operation, because building an XML document in WebLogic Process Integrator does not allow a document type to be entered. Some of the data needed to build this message comes from the calling PIP. (See “Workflow Inputs” on page 67 for the data provided by the calling PIP.) After the service header XML of the message is built and the document type is added, the header is validated, using the Validate Header business operation, against an XML schema that enforces the Service Header message guidelines.

Decision: validHeader?

The decision checks whether the service header was validated successfully against the schema. If the header is not valid, a message is logged using the Log business operation and the workflow ends because this is the first message of this PIP instance. This action is specified in the *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

Decision: retry = 0

This variable represents the number of times the c-enabler for RosettaNet has tried to send the message. If the value of the retry variable is 0, this indicates that this is the first time the c-enabler for RosettaNet has tried to send the message, and this is when the service context is created. If value of the retry variable is not 0, the previous context is reused.

Task: Build Failure Notif Content

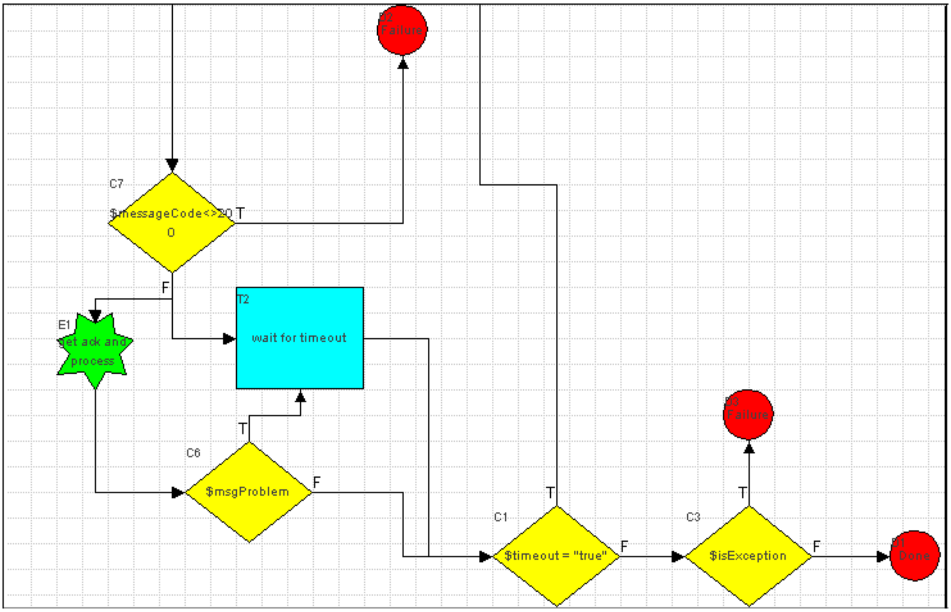
The service content of the Failure Notification message is built and assigned to the `failureNotifContent` workflow variable. Some of the data needed to build this message comes from the calling PIP. (See “Workflow Inputs” on page 67 for data provided by the calling PIP.) This service content is then validated, using the Validate Content business operation, against the XML schema that enforces the PIP_0A1 Failure Notification message guidelines.

Decision: validContent?

The decision checks whether the service content was validated successfully against the schema. If the message content is not valid, a message is logged using the Log business operation and the workflow ends because this is the first message of this PIP instance. This action is specified in the *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

Task: Send Failure Notification Message

The task sends the Failure Notification message to the PIP0A1_Admin workflow. The following illustration shows the second part of the PIP0A1_Notifier workflow.



Decision: messageCode<>200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation, and the workflow ends.

Task: Wait for Timeout

If the Failure Notification message is sent successfully, according to the PIP0A1 specification, the administrator has two hours to send a Receipt of Acknowledgement message. This task sets a task due date to two hours using the `ackTimeout` workflow variable.

Event: Get Ack and Process

This event waits for an acknowledgement from the administrator (the other role in this PIP). This message could be a Receipt Acknowledgement or a Receipt Acknowledgement Exception. When the message is received, the business operation Process Receipt Acknowledgement is called to determine the kind of message and to validate the message according to the appropriate message guidelines. The business operation returns an indication of whether there is a validation problem with the message and assigns it to the `msgProblem` workflow variable.

Decision: msgProblem?

This decision checks whether the message has a validation problem. If the message received cannot be validated, then according to the *RNIF Technical Advisory #1*, it should be treated like a timeout. So if there is a problem, the workflow enters the Wait for Timeout task.

Decision: timeout=”true”?

This decision checks whether the two-hour timeout task has expired. If a timeout occurred, then it continues to the check to see whether the message should be retried.

Decision: retry < $maxRetries?

This decision checks whether the number of retries has reached the value specified by the `maxRetries` variable. For this PIP the message can be sent up to four times because the `retry` variable is initialized to 0, and the `maxRetries` variable, to 3. If the message has not been retried the maximum number of times, the `retry` variable is incremented by one and the workflow resumes rebuilding and resending the Failure Notification message. If the maximum number of retries has been reached, the `reason` variable is updated with an error message.

Decision: isException?

This decision looks at the `isException` variable set in the Process Receipt Acknowledgement business operation, which indicates whether or not the message received from the administrator was a Receipt Acknowledgement Exception or a General Exception. If the message was an exception message then the workflow ends. If the message was a Receipt Acknowledgement then the workflow continues.

Task: Low Level Error Handler

This task is provided as an extension point in the workflow to handle low-level failures as described in the *RosettaNet PIP0A1: Notification of Failure specification.*

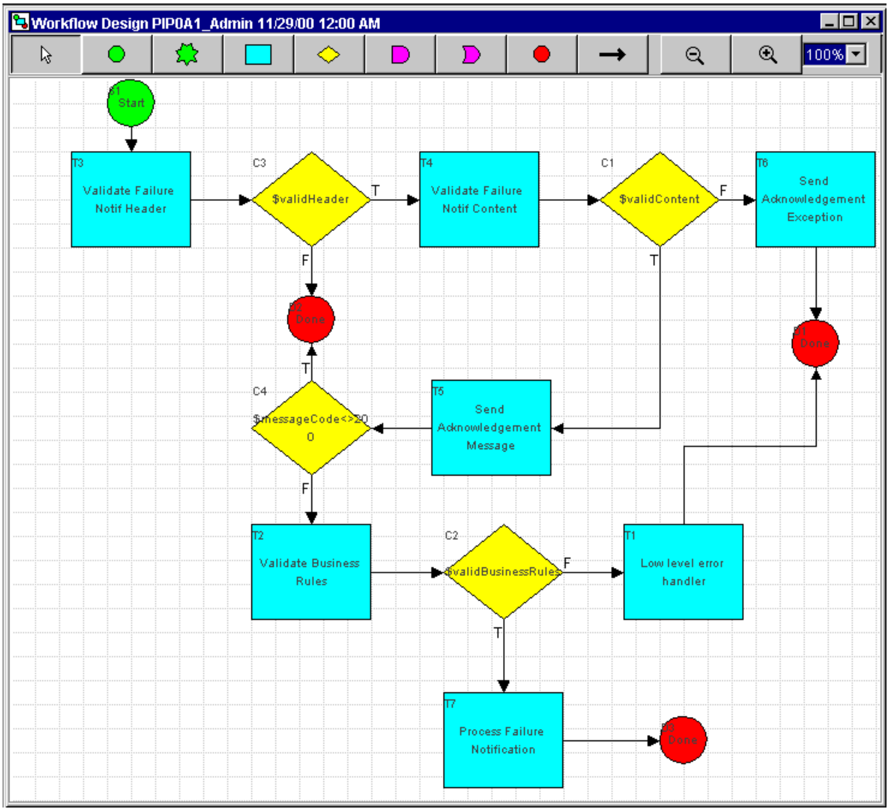
Done

This node logs the completion of the workflow, using the Log business operation.

PIP0A1_Admin

For more information about this workflow, see *PIP Specification, PIP0A1 Notification of Failure*. For details about exception handling, see *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

The following illustration shows the PIP0A1_Admin workflow.


Workflow Design PIP0A1_Admin 11/29/00 12:00 AM
100%
S1
Start
T3
Validate Failure Notif Header
C3
$validHeader
T
F
D2
Done
T4
Validate Failure Notif Content
C1
$validContent
F
T
T6
Send Acknowledgement Exception
D1
Done
C4
$messageCode<>200
T
F
T5
Send Acknowledgement Message
T2
Validate Business Rules
C2
$validBusinessRules
F
T
T1
Low level error handler
T7
Process Failure Notification
D3
Done

Start

The workflow is started upon receipt of a RosettaNet message. The Start node logs the start of the workflow, using the Log business operation, and initializes workflow variables that are used throughout the workflow. The variables `fromContactName`, `fromEmailAddress`, and `fromTeleNo` should be modified to the appropriate values for the administrator. The variables shown in all upper case are constants that refer to XML schema files. The following table describes all the variables initialized in the Start node and shows the values with which they are initialized.

Table 12 Variables Initialized in the Start Node

Variable	Initialization Value	Description
`HEADER_SCHEMA`	`ServiceHeaderPartMessageGuideline.xsd`	The name of the Service Header Part Message Guideline schema file. The variable is used by the Validate Failure Notification Header task.
`FAILURE_NOTIF_SCHEMA`	`0A1FailureNotificationMessageGuideline.xsd`	The name of the Failure Notification Message schema file. The variable is used by the Validate Failure Notification Content task.
`DOCUMENT_TYPE`	`<!DOCTYPE ServiceHeader SYSTEM \”ServiceHeaderPartMessageGuideline.dtd\”>`	The DOCTYPE line that needs to be inserted in the service header XML string. The service header is a String variable that contains the service header XML structure. There is no way to put the DOCTYPE in the structure via WebLogic Process Integrator so it is inserted with the Add Document Type business operation after the structure is built. The variable is not used by the workflow.
`fromContactName`	`John Doe`	Contact name. This variable is used by the Prepare Receipt Exception business operation.
`fromEmailAddress`	`jdoe@company.com`	Contact email address. This variable is used by the Prepare Receipt Exception business operation.
`fromTeleNo`	`9876543214`	Contact telephone number. This variable is used by the Prepare Receipt Exception business operation.

Task: Validate Failure Notif Header

This task validates the header of the message received, using the Validate Header business operation, against the XML schema that enforces the Service Header message guidelines.

Decision: validHeader?

This decision checks whether the header was validated successfully. If the header is not valid, then the workflow logs a message, using the Log business operation and ends, which means that the Notifier workflow will time out waiting for a response. If the workflow is valid, then the workflow continues.

Task: Validate Failure Notif Content

This task saves some values from the Failure Notif header in the following workflow variables:

- `fromDUNS`
- `fromPartnerClassCode`
- `fromRoleClassCode`
- `toDUNS`
- `toRoleClassCode`
- `toPartnerClassCode`
- `initiatingDUNS`
- `transactionInstanceId`
- `actionInstanceId`
- `processInstanceId`

The variable values are obtained by using XPath expressions on the `actionHeader` workflow variable. The variables are used when the service header is built. This task then validates the content of the message received, using the Validate Content business operation, against the XML schema that enforces the Failure Notification message guidelines.

Decision: validContent?

This decision checks whether the content was validated successfully. It also saves some values from the Failure Notif content in the following workflow variables:

- `fromDocid`
- `fromDocTimeStamp`

- fromSupplyChainCode
- toSupplyChainCode

The variable values are obtained by using XPath expressions on the serviceContent workflow variable. The variables are used when the service header is built. If the content is not valid, then a message is logged, using the Log business operation, and a Receipt Acknowledgement Exception is sent. If the content is valid, then a Receipt Acknowledgement is sent.

Task: Send Acknowledgement Exception

This task updates the reason variable with an error message and then builds the header for the Receipt Acknowledgement Exception. It then calls the business operation Prepare Receipt Exception, which continues to build the message. The message is then sent to the Notifier. When this task completes, the workflow is done.

Task: Send Acknowledgement Message

This task saves the failure notification message in the failureContent workflow variable. The task then builds the header for the Receipt Acknowledgement and calls the business operation Prepare Receipt Acknowledgement, which continues to validate the header, and to build and validate the content. The RosettaNet message is then sent to the Notifier.

Decision: messageCode <>200?

This decision node determines whether the RosettaNet message was sent successfully to the URL specified in the c-enabler for RosettaNet configuration file. The message code is set to the HTTP status by the Send RosettaNet Message action. Any value other than 200 indicates a problem. If there is a problem sending the message, the problem is logged using the Log business operation, and the workflow ends.

Task: Validate Business Rules

This task is a placeholder for any additional business rules that may need to be checked against the reply message. Currently only the validBusinessRules variable is set to true.

Decision: validBusinessRules?

This decision checks whether the business rules are valid. If the business rules are valid, then the workflow is complete. If not, the workflow continues to the low-level error handler.

Task: Process Failure Notification

This task is provided as an extension point in the workflow to handle the Failure Notification processing. Example actions: sending alerts to an administrator's console, sending e-mail notifications, and logging failures in a file.

Task: Low Level Error Handler

This task is provided as an extension point in the workflow to handle low-level failures, as described in the *RosettaNet PIP0A1: Notification of Failure specification*.

Done

This node logs the completion of the workflow using the Log business operation.

Descriptions of Business Operations

Business operations are called from the workflows. The following sections provide an overview of the actions performed by these business operations:

- General Business Operations
- PIP3A2-Specific Business Operations

For additional information, see the WebLogic Process Integrator Javadoc, which is provided with BEA WebLogic Collaborate.

General Business Operations

This section describes business operations that may be used, without modification, by all PIP workflows to perform actions common to all PIP workflows.

Log

This business operation logs information to the `<WLC_HOME>/rosettanet/enabler/wlc.log` file. The log provides some feedback on what happened in the workflow without having to look at the workflow status in WebLogic Process Integrator Studio.

Add Document Type

This business operation adds the document type to an XML String document from WebLogic Process Integrator. The document type should be specified in the following form:

```
<!DOCTYPE ServiceHeader SYSTEM "ServiceHeaderPartMessageGuideline.dtd">
```

Validate Header

This business operation validates an XML document passed in as a string against an XML schema that was created to enforce the message guidelines for the header. The XML string and the name of the XML schema are passed as parameters.

Validate Content

This business operation validates an XML document passed in as a string against an XML schema that was created to enforce the message guidelines for the service content. The XML string and the name of the XML schema are passed as parameters.

Prepare Receipt Exception

This business operation completes building the service header by adding the document type to the header and builds the service content for the Receipt Acknowledgement Exception message. The service header should be built before this business operation is called. No validation of the built message is done because this signal message is built correctly in the workflow. The workflow instance id is passed in so workflow variables can be used to build the message. The workflow variables `signalHeader` and `serviceContent` are also set. These variables are used when the workflow sends the message. This operation expects the following workflow variables to be defined and fails if any of them are not:

- `fromRoleClassCode`
- `fromPartnerClassCode`
- `fromSupplyChainCode`
- `fromDUNS`
- `fromContactName`
- `fromEmailAddress`
- `fromTeleNo`
- `toRoleClassCode`
- `toPartnerClassCode`
- `toSupplyChainCode`
- `toDUNS`
- `signalHeader`
- `serviceContent`

Is This a General Exception Message?

This business operation returns an indication as to whether the message is a General Exception message by looking at the service header passed in.

Process Receipt Acknowledgement

This business operation checks whether the message received is a General Exception message, a Receipt Acknowledgement Exception message, or a Receipt Acknowledgement message. It validates the service header and content of the message and returns an indication of whether the message is invalid (`msgProblem`). It also sets the workflow variable `isException`, which is used in a Decision node later in the workflow. This operation expects the following workflow variables to be defined and fails if any of them are not:

- `serviceHeader`
- `serviceContent`
- `isException`
- `reason`

Prepare Receipt Acknowledgement

This business operation completes building the service header by adding the document type to the header and builds the service content for the Receipt Acknowledgement message. The service header should be built before this business operation is called. No validation of the built message is done because this signal message is built correctly in the workflow. The workflow instance id is passed in so workflow variables can be used to build the message. Workflow variables are also set to be able to send the message. This operation expects the following workflow variables to be defined:

- `fromRoleClassCode`
- `fromPartnerClassCode`
- `fromSupplyChainCode`
- `fromDUNS`
- `fromContactName`
- `fromEmailAddress`
- `fromTeleNo`
- `toRoleClassCode`
- `toPartnerClassCode`
- `toSupplyChainCode`
- `toDUNS`
- `signalHeader`
- `serviceContent`

Load XML Doc String

This business operation loads an XML document from the file with the filename that is passed in and returns the document as a string. In the example workflows this business operation is used to load XML files such as the following:

- `3A2PriceAndAvailabilityQueryMessage.xml`
- `3A2PriceAndAvailabilityResponseMessage.xml`

PIP3A2-Specific Business Operations

The following business operations are called by PIP3A2 workflows:

- Prepare Query
- Validate Query
- Prepare Response

The following sections describe each of these business operations in detail. The PIP3A2-specific business operations may be leveraged for other PIP workflows but Java code modifications are required.

Prepare Query

This business operation saves data from the service content (query) message in workflow variables that are used later to build messages. It also sets the document ID and timestamp in the query message in preparation for sending. This operation expects the following variables to exist in the workflow:

- `fromContactName`
- `fromTeleNo`
- `fromEmailAddress`
- `fromRoleClassCode`
- `toRoleClassCode`
- `fromSupplyChainCode`
- `toSupplyChainCode`
- `toDUNS`

- toPartnerClassCode
- fromDUNS
- fromPartnerClassCode
- serviceContent
- processInstanceId

Validate Query

This business operation saves data from the query message that has been received in workflow variables that are used later to build messages. This operation expects the following variables to be defined in the workflow and fails if any of them are not defined:

- fromDocid
- fromDocTimeStamp
- toDUNS
- toRoleClassCode
- toPartnerClassCode
- toSupplyChainCode
- fromDocid
- fromDocTimeStamp
- processInstanceId

Prepare Response

This business operation takes the serviceContent variable (which contains the response message for the Supplier workflow), updates some elements of it, and saves it. The following XML elements are modified from the Pip3A2PriceAndAvailabilityResponse message:

- *thisDocumentGenerationDateTime.DateTimeStamp*
- *thisDocumentIdentifier.ProprietaryDocumentIdentifier*
- *requestingDocumentDateTime.DateTimeStamp*
- *requestingDocumentIdentifier.ProprietaryDocumentIdentifier*

Message Validation Processes

The message validation process uses the Xerces 1.2.0 DOM parser, which supports an alpha implementation of the XML Schema specification. The Xerces 1.2.0 DOM parser is packaged with the WebLogic Collaborate Enabler for RosettaNet software.

This section describes how RosettaNet message validation is implemented in WebLogic Collaborate Enabler for RosettaNet. It also provides a bibliography for further reading about message validation.

RosettaNet Message Validation Implementation

This section explains the message validation logic used within the RosettaNet exception handling process implemented by the example workflows:

- Preamble Grammar Validation
- Service Header Grammar Validation
- Service Header Content and Message Sequence Validation
- Action Message Grammar and Schema Validation
- Action Message Content Validation
- Business Signal Grammar and Schema Validation
- Validator Utility

For an explanation of the exception handling process, see *RosettaNet Implementation Framework, v1.1: Technical Advisory #1*.

The example XML schema files are located in `<WLC_HOME>/rosettanet/enabler/schemas`. The document type definition (DTD) files from `www.rosettanet.org` are located in `<WLC_HOME>/rosettanet/enabler`.

Preamble Grammar Validation

The RosettaNet Protocol Layer validates the Preamble grammar against RosettaNet's `PreamblerPartMessageGuideline.dtd` file.

Service Header Grammar Validation

The Service Header grammar is validated against RosettaNet's `ServiceHeaderPartMessageGuideline.dtd` file.

Service Header Content and Message Sequence Validation

In the workflow examples, the Service Header content is validated against the XML schema file `ServiceHeaderPartMessageGuideline.xsd`. The XML schema file was first generated using the `ServiceHeaderPartMessageGuideline.dtd` file. It was then modified to include the constraints and validation rules defined in RosettaNet's `ServiceHeaderPartMessageGuideline.htm` file.

The message sequence validation is provided in a couple of ways. When a workflow receives an action message, the Service Header validation is performed. An out of sequence business message fails this validation. When a workflow receives a business signal, the Process Receipt Acknowledgement business operation performs sequence validation. This business operation could potentially receive a General Exception message, a Receipt Acknowledgement Exception message, a Receipt Acknowledgement message, or an action message. The Process Receipt Acknowledgement business operation handles each type of message in accordance with the RosettaNet exception handling process.

Action Message Grammar and Schema Validation

The Action Message grammar is validated against a RosettaNet PIP DTD. The Action Message schema is validated against an XML schema file with the same name as the associated PIP DTD, except that the `.dtd` filename extension is replaced with the `.xsd` filename extension. The XML schema files were generated using the PIP DTD files. They were then modified to include the constraints and validation rules defined in the associated RosettaNet message guideline HTML files.

Action Message Content Validation

Action Message content is validated against a company's business rules. These business rules have not been implemented. Validate Business Rules tasks have been inserted in the workflows in appropriate locations. The user is responsible for implementing the appropriate actions within a Validate Business Rules task. Business rules can also be incorporated into the XML schema files described in "Business Signal Grammar and Schema Validation" on page 87. For example, a business rule could be implemented in an XML schema file that specifies that a supplier will only respond to requests for orders of quantities greater than 100 units.

Business Signal Grammar and Schema Validation

The Business Signal grammar is validated against the applicable RosettaNet Business Signal DTD. The Business Signal schema is validated against an XML schema file with the same name as that of the associated Business Signal DTD, except that the `.dtd` filename extension is replaced with the `.xsd` filename extension. The XML schema files were generated using the Business Signal DTD files. They were then modified to include the constraints and validation rules defined in the associated RosettaNet message guidelines.

Validator Utility

The class `com.bea.b2b.rosettanet.Validator` is a utility that can be used to validate an XML message file against a DTD file or an XML schema file, or both.

To use this utility, enter the following command line:

```
java com.bea.b2b.rosettanet.Validator arg0 arg1
```

In this command line:

- *`arg0`* is an XML file URI (for example, `MyFile.xml`)
- *`arg1`* is an XML schema file URI (for example, `MyFile.xsd`)

Note: The XML schema file must be located in a subdirectory called `schemas` directly below the directory from which `Validator` is invoked. Also, if `MyFile.xml` contains a `DOCTYPE` line, the DTD file must reside in the path specified by the DTD file that is specified on the `DOCTYPE` line in the `MyFile.xml file`.

Recommended Reading about Message Validation

The following information is recommended reading if you want to fully understand the example XML schemas; it is required reading if you are planning to implement your own XML schemas:

- Information about XML schema tools, usage, specifications, and development is available at `http://www.w3.org/XML/Schema`. *XML Schema Part 0: Primer* provides good descriptions of the features and capabilities of XML schema.
- Information about the Xerces implementation of the XML Schema specification is available at `http://xml.apache.org/xerces-j/schema.html`.

Index