WebSphere® Partner Agreement Manager

# Channel Toolkit Configuration Guide

*Version 2 Release 2*

**BIAAAH02**

**Note:** Before using this information and the product it supports, read the information in *Notices* on page 69.

**Third Edition (July 2001)**

This edition applies to version 2, release 2 of WebSphere Partner Agreement Manager (product number 5724-A85) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can make comments on this information via e-mail at idrcf@hursley.ibm.com.

# TABLE OF CONTENTS

# Welcome to the Channel Toolkit Configuration Guide

This document describes the Partner Agreement Manager Channel Toolkit and explains how to install and configure it.

**To install and configure the Channel Toolkit, follow these general steps:**

- Learn the general concepts about the Channel Toolkit. For more information, see *Introduction* on page 1.
- Install the Channel Toolkit framework and modules. For more information, see *Installing Channel Toolkit* on page 11.
- Configure the assignments for modules. For more information, see *Configuring modules* on page 17.

## Who should use this information

The Channel Toolkit has two diverse audiences:

- Partner Agreement Manager administrators, who have special needs for communicating with their partners and need to configure a channel that addresses these special needs. For example, a customer may require a channel that sends MIME-encoded XML to partners via FTP, and receives responses via e-mail. This customer can use the Channel Toolkit along with the appropriate modules to configure a channel that does this.
- IT Developers, who are writing code to perform special purpose operations on a business object en route between Partner Agreement Manager and a partner. These operations could include transformations, encoding, encryption, and even special transport requirements. These developers write modules using specific interfaces. These modules can then be assembled into a special-purpose channel by any PAM administrator.

This guide is primarily addressed to administrators, but also describes concepts that are important to developers. Developers should refer to the Javadocs for information on the Channel Toolkit API. The Javadocs are located in <*Alliance*>\Docs\javadocs\ChannelToolkit\api\index.html.

## Related information

For additional information see the following:

- The readme.htm file. This file may contain information that became available after this guide was published. The readme.htm file is located in the root directory of the product CD-ROM.
- The StartHere.htm file. This file contains links to the Partner Agreement Manager readme.htm file and the Partner Agreement Manager Installation Guide . Before installation, the StartHere.htm file is located in the root directory of the product CD-ROM. After installation, the StartHere.htm file is located in the root directory of the Partner Agreement Manager installation.
- The *Partner Agreement Manager Installation Guide*, form number GC34-5964-00, which describes how to install Partner Agreement Manager.

- The *Partner Agreement Manager Administrator's Guide*, form number BIAAAB02, which describes how to set up, configure, and administer Partner Agreement Manager after you install it.
- The *Partner Agreement Manager User's Guide*, form number BIAAAC02, which describes how to start a Partner Agreement Manager session, design public and private processes, define element definition sets, create business objects, and manage process distribution.
- The *Partner Agreement Manager Adapter Developer's Guide*, form number BIAAAD02, which describes how to develop and administer adapters using the Partner Agreement Manager Adapter Development Environment.
- The *Partner Agreement Manager Script Developer's Guide*, form number BIAAAE02, which describes how to write scripts used in Partner Agreement Manager private processes and elsewhere.
- The *Partner Agreement Manager API Guide*, form number BIAAAF02, which describes principles behind the Partner Agreement Manager External API. See also the API Javadocs, as described in *API Javadocs*, next.
- The *Partner Agreement Manager Adapters for MQSeries User's Guide*, form number BIAAAG02, which describes how to install, configure, and run the Partner Agreement Manager Adapters for MQSeries.
- The *Partner Agreement View User's Guide*, form number GC34-5965-00, which describes how to install, configure, and use Partner Agreement View.

## API Javadocs

The Channel Toolkit Javadocs, are located in
<*Alliance*>\Docs\javadocs\ChannelToolkit\api\index.html.

IT developers will also be interested in these two sets of online Javadocs, which will help you work with the Channel Toolkit API:

- The *Partner Agreement Manager API* online documentation, which describes the programming interface for the adapters and business objects programming interface.
- The *Partner Agreement Manager External API* online documentation, which describes the programming interface for the Process Manager functionality.

You can link to both of these sets of documentation via the Alliance\StartHere.htm file.

CHAPTER

# 1

# INTRODUCTION

The Partner Agreement Manager Channel Toolkit allows you to create your own simple channels from modular components. If you are an IT developer, you can use the Channel Toolkit API to create your own components that plug into the Channel Toolkit.

Topics in this chapter include:

## Terminology

It is useful to be familiar with these terms used by the Channel Toolkit:

| Term | Definition |
|---|---|
| framework | The Channel Toolkit framework manages much of the complexity in the channel, including the interaction between Partner Agreement Manager and the modules, validation of the channel configuration, error handling, retry, and so on. |
| module | Modules plug into the framework to provide specific services required by the channel. See also definitions in this table for: *data transformation module*, *packaging module*, *unpackaging module*, and *transport module*. |
| data transformation module | A module that transforms a single business object into an arbitrary stream of data, or *vice-versa*. |
| packaging module | A module that packages one or more business object streams into a stream that can be sent by the transport module. Examples of packaging include enveloping (such as MIME encoding), encryption, grouping single business objects together, and so on.<br>In general, the term “packaging module” is used to mean both packaging and unpackaging modules. |
| unpackaging module | A module that unpackages a stream into one or more business object streams. Examples of unpackaging include recognizing business object types, de-enveloping, decryption, breaking a multi-business object message into individual business objects, and so on.<br>In general the term “packaging module” is used to mean both packaging and unpackaging modules. |
| transport module | A module that takes a stream and sends it out over a transport medium (such as HTTP), or receives a stream from a transport medium and passes it to the framework. |
| assignment | Information used by the framework to determine the default and override modules and properties used for a particular inbound or outbound transmission.<br>Given a partner ID and business object type ID, the Channel Toolkit determines which module to invoke and passes runtime configuration data to the module. |

For additional terms that encompass the Channel Toolkit API, see the Channel Toolkit Javadocs.

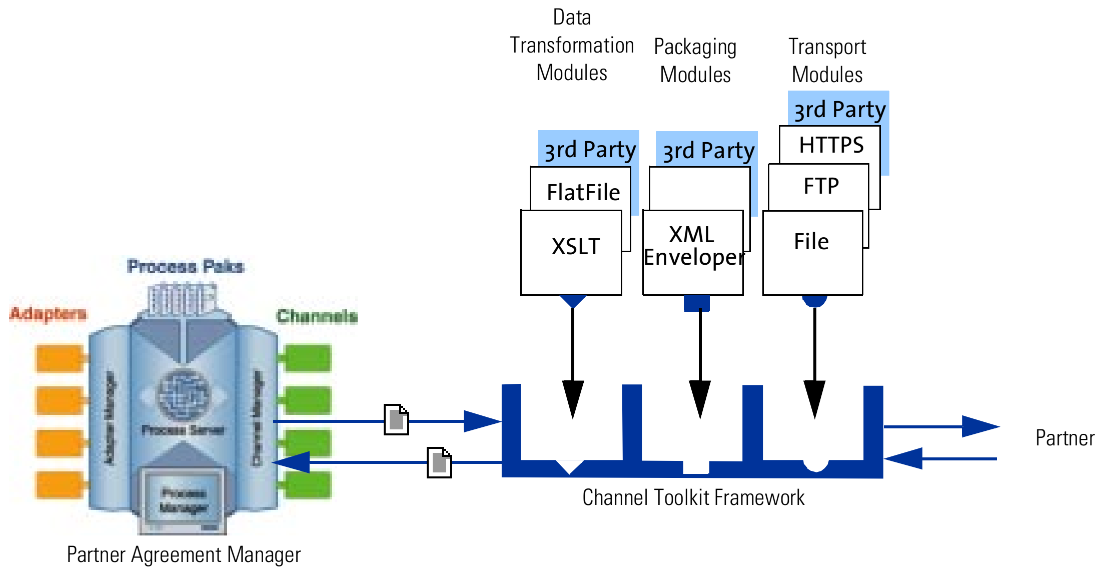
## Architecture

The Channel Toolkit design is modular, composed of a framework and modules.

The Channel Toolkit is made up of four components:

- Channel Toolkit framework
- Data Transformation modules
- Packaging modules
- Transport modules

These are illustrated in this diagram. Note that the modules shown here are examples; they don't necessarily indicate modules that are shipped with the Channel Toolkit.



The framework communicates with the Partner Agreement Manager and invokes the modules. As information passes through the framework, it is processed by an optional transformation module, an optional packaging (or unpackaging) module, and a transport module.

When you configure the Channel Toolkit, you specify the default and override modules to be invoked for inbound and outbound messages. You can specify overrides based on specific business objects, specific partners, or both. You can also specify properties to be used when invoking a module.

Before calling each module, the framework examines the message data and finds the module that best fits the default and override assignments specified for the Channel Toolkit instance. Any of the modules can split inbound data into separate pieces. Once separated, the framework routes each piece to modules as a separate message. Thus, the parts of a single message can be routed to two (or more) different modules in a subsequent step.

**NOTE:** The module assignments must specify a default inbound and outbound transport module, but packaging and transform module specifications are optional. If no packaging or transform module is specified, the framework uses internal defaults that assume well-formatted XML compatible with Partner Agreement Manager.

## FRAMEWORK

The Channel Toolkit framework is responsible for tasks that are common to all basic channels; it is also responsible for module management tasks. The common tasks include managing retry functionality, conversation threading, polling for inbound messages, and logging. The module management tasks include routing messages to the appropriate modules, splitting messages, and handling errors from modules.

### ABOUT CHANNEL TOOLKIT INSTANCES

You can install multiple instances of the Channel Toolkit. This is done by rerunning the installer and specifying a different Channel ID and Channel name. Each instance must have a unique Channel ID and name, and each has its own configuration information (for more information, see *Installing the Channel Toolkit framework* on page 13).

There is a trade-off between multiple instances of the Channel Toolkit and a single instance. It is possible to configure a single instance of the Channel Toolkit to handle all the business objects that you need to send to and receive from all your partners. However, such a configuration can become unmanageably complex.

### The framework manages transport polling

The Channel Toolkit supports polling transports, such as FTP, through a special interface that allows the framework to ask a transport module to check for new messages. The framework manages the list of polling transports configured for its channel instance and automatically polls them according to a polling interval defined for that module. If no interval is specified, the default is used (all defaults are specified in the local chanel profile). For more information, see *Setting default transport property values* on page 15.

### The framework handles retries

In the event of a non-fatal delivery failure by the transport module, the framework queues the message for redelivery. After a specified delay, the framework passes the message back to the transport module for redelivery. If the delay is not specified, the default for the channel instance is used instead.

The framework continues to resend the message until it is successful or until the retry count exceeds a specified limit. Again, if the limit is not specified, the default for the channel instance is used instead.

### The framework requires conversations

The Channel Toolkit framework must use conversation threading to maintain the process context of its messages. For more information about conversation threading, see *Creating conversations* on page 53.

## Modules

Modules plug into the framework and provide services required to:

- Prepare and send a business object to a partner.
- Receive and process messages containing one or more business objects from a partner.

The modular architecture of the Channel Toolkit makes a wide variety of modules possible. Some common modules are available from Extricity, and new ones will be available in the future. You can also create your own modules using the Channel Toolkit API, which is documented in the Channel Toolkit Javadocs. The Channel Toolkit also comes with sample code to help you understand how to create your module. If you choose to create your own modules, see *Module installer actions* on page 65 to learn how to register them with the Channel Toolkit Framework.

## Modules are invoked by the framework

As mentioned earlier, the framework moves message data through the channel, calling modules in order. For outbound messages, the order is data transformation, packaging, and transport. For inbound messages, the order is transport, unpackaging, and data transformation.

The following sections describe the modules in greater detail.

## The data transformation modules

A data transformation module is an optional component that converts business data to or from a business object. For inbound messages, the destination business object must be determined by the unpackaging or transport component, before invoking the data transformation module.

This module deals with a single business object at a time; the framework cannot call this module more than once.

Examples of data transformation modules might include XSLT or flat file.

## The packaging and unpackaging modules

A packaging module is an optional component that wraps, encrypts, or otherwise manipulates outbound data before sending. An unpackaging module similarly removes wrapping, decrypts, or manipulates inbound data. The framework cannot call packaging or unpackaging modules more than once.

Examples of packaging module might include S/MIME encoding, encryption, or packaging of multiple business objects together. Examples of unpackaging module might include recognizing business object type, decoding, decryption, or splitting one multi-business-object message into individual business objects.

## The transport modules

A transport module conveys data to or from a partner via whatever transport mechanism it supports - ftp, http, e-mail, file, message queue, and so on. For outbound data, the transport module simply conveys the data. For inbound data, there are two modes of operation: polling or listening.

**Note:** The transport module is responsible for transport-level authentication, so the framework assumes that inbound messages from a transport module have been appropriately authenticated.

Examples of transport modules might include HTTP(S), FTP, or SMTP.

## How the framework resolves overrides and defaults

The framework determines the module to use for a particular message just before it calls the module. The module used is determined by overrides and defaults specified by the channel assignments.

The overrides are based on:

- The ID of the partner that is sending or receiving the data.
- The ID of the business object type being sent or received.
- Both the ID of the partner *and* the ID of the business object type.
- Whether the data is being sent or received (inbound or outbound).

**Important:** The framework cannot predict which module will be called until the data is emitted by the previous module. This is particularly true of the inbound case, where partner and/or business object type may only be recognized after the message is handled by an unpackaging module. Sometimes messages must be split and the component parts routed to different modules as information about their content is determined. Thus, the path that a particular message takes cannot be defined ahead of time.

Before calling a module, the framework compares the message's business object type, partner ID, and direction with the assignments defined for that module. The assignments are defined in the CTAssignments.xml file, which is described in *Configuring modules* on page 17.

**After determining a message's business object type (if that information has been determined by a previous module), partner ID, and direction, the framework:**

1. Looks for an override for the module type (transport, packaging, or transform) that matches the message's direction, business object type, and partner ID. If a match is found, that override is used.
2. Looks for an override for the module type that matches the direction and business object. If a match is found, that override is used.
3. Looks for an override for the module type that matches the direction and partner ID. If a match is found, that override is used.
4. Looks for a default for the module type. If a default is found, it is used.
5. If no default is specified for the packager module, it is skipped. If no default is specified for a transform module, the Channel Toolkit uses its own internal default (which treats data as well-formatted, XML data, compatible with Process Server). The assignments file *must* define a default for transport.

## Configuration Files

The Channel Toolkit has two configuration files:

- CTModules.properties - Serves as a registry of modules installed on the system. It contains just enough of the global module properties so that a module user (for example, the Channel Toolkit framework) can find the module and use the API to query the rest of its properties. The remaining properties are obtained by loading the module and querying its module metadata interface. Administrators need not edit this file, but if you're writing your own modules, you will need to register them here. For more information, see *Module installer actions* on page 65.
- CTAssignments*NNNN*.xml - Defines the default and override modules used by this Channel Toolkit instance. Also contains the defaults and overrides for module properties. The *NNNN* in the file name is the channel ID.

  This file is stored in the local host's partner directory:
  <*PAM root*>\Partners\Partner*XXXX*\Properties\ChannelToolkit
  where *XXXX* is the Partner ID.

Throughout this document, we refer to the CTAssignments*NNNN*.xml file simply as CTAssignments.xml.

The CTAssignments.xml file is the file you will modify most frequently. When you configure a new module or define a new set of overrides, you must update CTAssignments.xml (for more information, see *Configuring modules* on page 17).

**NOTE:** If you have installed multiple instances of Channel Toolkit, each instance will have it's own CTAssignments.xml file, however CTModules.properties is shared.

A third file is the CTAssignments.dtd file, which describes the schema for the CTAssignments.xml file. The DTD file is also in <*PAM root*>\Partners\Partner*XXXX*\Properties\ChannelToolkit.

CHAPTER

2

# Installing Channel Toolkit

Read this chapter for information about installing Channel Toolkit and its modules.

Topics in this chapter include:

- *System requirements* on page 12.
- *Installation overview* on page 12.
- *Installing the Channel Toolkit framework* on page 13.
- *Setting default transport property values* on page 15.
- *Installing Modules* on page 16.

## System requirements

This section lists the Channel Toolkit system requirements for both Windows NT and UNIX. These requirements include what software you need to have installed before installing the Channel Toolkit.

### System requirements

Following are the minimum requirements for the Channel Toolkit:

- A working installation of IBM Partner Agreement Manager 2.2 with Service Pack 1. For details on the installation requirements see the Partner Agreement Manager or *Partner Agreement Manager Installation Guide*.
- 5 MB disk space.
- On UNIX, the Channel Toolkit installer requires an X-Windows interface.

## Installation overview

There are two fundamental parts to installing a Channel Toolkit channel: installing the Channel Toolkit components and configuring the module assignments. This chapter describes how to perform the first step. For more information on configuration, see *Configuring modules* on page 17.

**Step 1** Install the Channel Toolkit on the Process Server computer. This process is described in *Installing the Channel Toolkit framework* on page 13.

You can have more than one installation (that is more than one instance) of the Channel Toolkit. To do so, you simply run the installer again. For more information about multiple instances, see *About Channel Toolkit instances* on page 4.

In some cases, you might install the Channel Toolkit framework on both the Process Manager and Process Server computers:

- On UNIX, you *must* install the Channel Toolkit framework on both the Process Server and Process Manager computers.
- On Windows NT, it is optional to install the Channel Toolkit on the Process Manager computer.

However, if you install Channel Toolkit on both computers, there are some limitations. For more information, please see *Process Manager installations* on page 33.

**STEP 2** Install the modules you require. This process is described in *Installing Modules* on page 16.

You can add new modules at a later time.

**STEP 3** Configure the Channel Toolkit instances to use the new modules. This process is described in *Configuring modules* on page 17.

## INSTALLING THE CHANNEL TOOLKIT FRAMEWORK

These installation steps require that you have already installed IBM Partner Agreement Manager 2.2.

**To install the Channel Toolkit framework, perform these steps.**

1 Shut down the PAM Process Server.

2 This next step depends on whether you are installing on Windows NT or UNIX.

On Windows NT:

A. Click Start and choose Run

B. Click the Browse button and navigate to the location of the installation image for Channel Toolkit and locate `setup.bat` and click Open.

C. Click OK to run `setup.bat`.

On UNIX:

A. If you are using AIX, you must be logged in as root.

B. Untar the installation image to an empty directory.

C. Change directory to the location of the untarred installation image.

D. Change the permissions of the `setup` file to be executable.

E. Run `./setup`.

The Welcome panel appears. Click Next to continue.

3 The license panel appears. Read the license agreement and click the “I agree” option button. Click Next to continue.

4 The next panel prompts for your PAM root directory and PAM Partner ID. By default these values are filled in. If they are filled in, do not change them. If they are not filled in, provide the information requested. Click Next to continue.

5 The next panel displays the prerequisite information needed for the Channel Toolkit installation. Review the information and click Next to continue.

6 The next panel allows you to specify the Channel Instance name and the Channel ID.

The Channel ID is used by Partner Agreement Manager to distinguish between the channels you have installed, such as Channel for RosettaNet and the Channel Toolkit. Each channel must have a Channel ID that is unique within your Partner Agreement Manager installation.

Unless you have another Channel Toolkit installed, use the default Channel Instance name and Channel ID. If you need to change these defaults so they don't conflict with another Channel Toolkit installation, use a Channel ID number greater than 1000 that is not used by any other channel you have installed and enter a unique Channel Instance name. Make a note of the number and name so you don't use them when you install any other channels. You can check which channel ID values are in use by opening the Channel Manager from the Process Manager window and viewing the Channel profiles.

Unless you have another instance of your channel installed, accept the default for the channel instance name.

**IMPORTANT:** The Channel Instance name cannot start with a digit (0-9).

When you are satisfied with the values, click Next to continue.

7 The next panel prompts you for the destination folder for the Channel Toolkit components. We recommend you use the default value provided (the same directory as your PAM installation). Click Next to continue.

8 If the installer detects a previous installation of the Channel Toolkit.

9 The Installer adds the components. When it is finished, the final panel is displayed. Click Finish to end the installation process.

After you install the Channel Toolkit framework, you install modules and configure assignments for the modules. The next section (*Installing Modules* on page 16) describes how to install the modules. The following chapter (*Configuring modules* on page 17) describes how to configure assignments.

## Setting default transport property values

You use the Channel Manager to set default transport property values for a Channel Toolkit instance.

### To set the default transport property values

1. Launch PAM and open the Channel Manager.
2. Click on Channel Profiles.
3. Choose the Channel Toolkit instance you are configuring and click Edit>Properties.
4. Click on the Other tab to view the values.

This table lists the values and their meanings:

| Property | Meaning |
|---|---|
| Default_Max_Retries | The default number of times an outbound transport module attempts to retry a failed send operation. The default value is 5.<br>The assignments file can override this value for a specific business object or partner. |
| Default_Polling_Frequency | The default number of Polling_Interval units that an inbound transport module waits between polling. The default value is 3.<br>The assignments file can override this value for a specific business object or partner. The total time between polling is the polling frequency times Polling_Interval. |
| Default_Retry_Frequency | The default number of Retry_Timeout units that an outbound transport module waits between retries of a failed send operation. The default value is 3.<br>The assignments file can override this value for a specific business object or partner. The total time between retries is the retry frequency times Retry_Timeout. |
| Polling_Interval | The basic polling time unit in seconds.<br>The default value is 60 seconds. |
| Retry_Timeout | The basic retry time unit in seconds.<br>The default value is 60 seconds. |

The Update_Config_Info property is used when validating the assignments XML file. Its use is described in *Validating and using CTAssignments.xml* on page 32.

## Installing Modules

After you install the Channel Toolkit framework, you must install the modules you plan to use with the Channel Toolkit. The details of how to install a module vary, depending on who produced the module. Some modules might require you to hand edit the `CTModules.properties` file; other modules might come with installers that perform those functions.

All individual modules should come with installation documentation. Please refer to that documentation for installation instructions.

For information on the steps that a module installer must perform, see *Module installer actions* on page 65.

**Note:** If you are running on UNIX, which requires you to install channel toolkit on both the Process Server and Process Manager computers, or have just chosen to install Channel Toolkit on the both the Process Server and Process Manager computers, you must deploy modules to both computers. For more information see *Process Manager installations* on page 33.

CHAPTER

3

# Configuring modules

After installing the Channel Toolkit framework and modules, you need to configure assignments to determine how to use the modules.

This chapter includes these topics:

## Overview

After you install the Channel Toolkit framework and modules, you need to configure one or more assignments for the modules. The steps to configure assignments include:

**Step 1** Create Partner Agreement Manager partners that will use the new Channel Toolkit channel (if they don't exist already).

**Step 2** Create Partner Agreement Manager processes and business objects. This step is only necessary if you intend to create per-partner overrides.

**Step 3** Launch the Channel Manager. On start-up, Channel Toolkit creates three helper text files that you can use to cut and paste information into `CTAssignments.xml`. For more information, see *Creating helper files* on page 25.

**Step 4** Configure the defaults and overrides in `CTAssignments.xml`. The common tasks involved in modifying `CTAssignments.xml` are described in *Creating assignments* on page 29.

**Step 5** Validate and employ the modified `CTAssignments.xml` file. This is described in *Validating and using CTAssignments.xml* on page 32.

## About CTAssignments.xml

`CTAssignments.xml` contains all the run-time module assignments and properties. These definitions are too complex to fit into a flat-file schema, so a hierarchical XML schema is used instead.

As part of adding a module that will be used by a Channel Toolkit instance, you modify the corresponding `CTAssignments.xml` file to describe when to use the module and the properties for that module. (For more information on installing a module, see *Installing Modules* on page 16.)

More specifically, you:

- Define the default modules and their properties for a Channel Toolkit instance (for both inbound and outbound messages). For more information, see *Default assignments* on page 22.

- Specify the override assignments by identifying combinations of business object type, partner, and direction (inbound/outbound) and specifying the modules and properties to use in those circumstances. For example, you can specify that when a specific business object is received from a partner, a special transform module must be used on that business object.

**IMPORTANT:** If you are not familiar with editing XML files, there are many XML file editors that help you create well-formed, valid XML. You can use one of these editors to edit CTAssignments.xml. The IBM alphaWorks site has a number of good XML file editors; the URL is http://alphaworks.ibm.com.

## Processing CTAssignments.xml

The Channel Toolkit uses the CTAssignments.xml file in two different ways:

- When a Channel Toolkit instance is started, it automatically processes the CTAssignments.xml file to load the run-time properties for that instance.
- By using the Channel Manager, you can explicitly cause the Channel Toolkit to process the CTAssignments.xml file (by setting the update_config_info property to true).

Whenever a Channel Toolkit instance processes CTAssignments.xml (whether automatically or explicitly), it does three things:

- It validates the contents of CTAssignments.xml.
- It generates helper files (see *Creating helper files* on page 25).
- It updates the runtime configuration information for that Channel Toolkit instance. This allows you to change channel properties without stopping and restarting Partner Agreement Manager.

For more information on explicitly processing CTAssignments.xml, see *Validating and using CTAssignments.xml* on page 32.

## The CTAssignments.dtd file

There are a number of ways to understand the structure of the CTAssignments.xml file. The most accurate way to view the XML file is through the DTD to which the file conforms (CTAssignments.dtd, stored in <*PAM root*>\Partners\Partner*NNNN*\Properties\ChannelToolkit). Following sections show examples of the actual CTAssignments.xml file.

```
<!-- CTAssignments.dtd  ChannelToolkit Assignments -->

<!ELEMENT assignments (default_assignment, override_assignment*)>
<!ELEMENT default_assignment (inbound_default, outbound_default)>
<!ELEMENT override_assignment (constraints, modules)>
<!ELEMENT constraints ( (bo_type_id) | (partner_id) | (bo_type_id, partner_id) )>
<!ELEMENT modules ((inbound_override | outbound_override) |
(inbound_override, outbound_override))>
<!ELEMENT inbound_override  ( (transport, packager?, transform?) |  (transport?,
packager, transform?) |  (transport?, packager?, transform)) >
<!ELEMENT outbound_override ( (transport, packager?, transform?) |  (transport?,
packager, transform?) |  (transport?, packager?, transform)) >
<!ELEMENT inbound_default  (transport, packager?, transform?)>
<!ELEMENT outbound_default (transport, packager?, transform?)>
<!ELEMENT transport (module, transport_internal?)>
<!ELEMENT packager (module)>
<!ELEMENT transform (module)>
<!ELEMENT module (module_name, module_properties?)>
<!ELEMENT module_name (#PCDATA)>
<!ELEMENT module_properties (description?, property*)>
<!ELEMENT property (name, value)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT bo_type_id (#PCDATA)>
<!ELEMENT partner_id (#PCDATA)>
<!ELEMENT transport_internal (
  ((retry_frequency, max_retries)?, is_synchronous?) | (polling_frequency))>
<!ELEMENT retry_frequency (#PCDATA)>
<!ELEMENT max_retries (#PCDATA)>
<!ELEMENT is_synchronous (#PCDATA)>
<!ELEMENT polling_frequency (#PCDATA)>
```

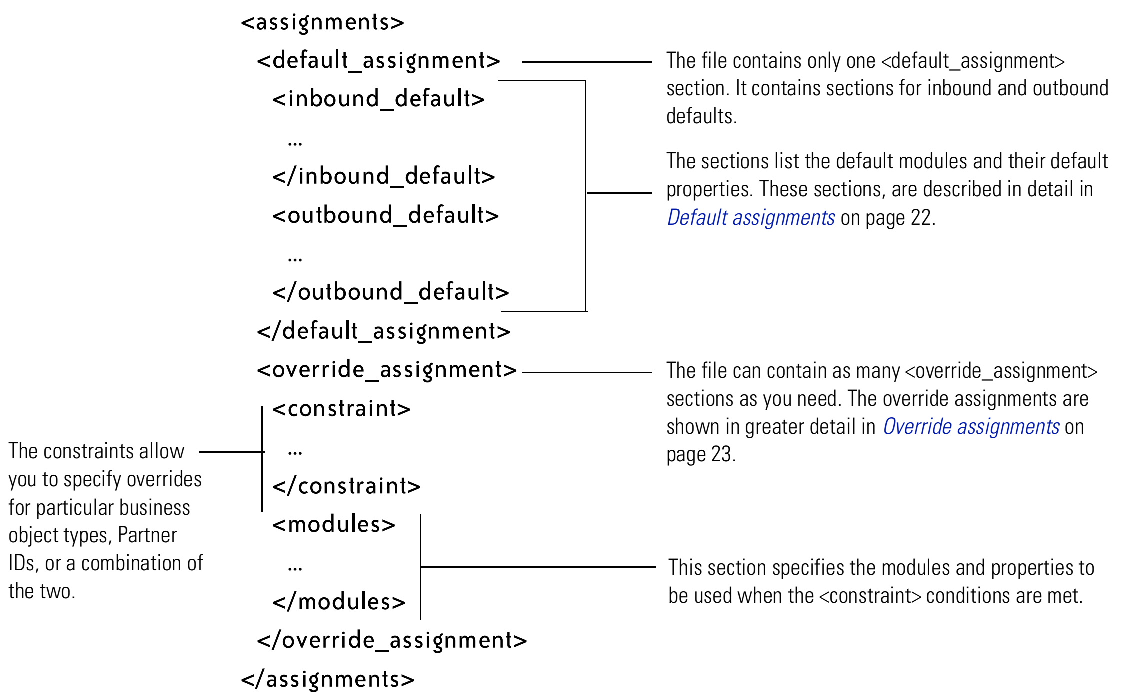
Some comments on the DTD:

- Default inbound and outbound transports are required, but both packager and transform are optional. If no packager or transform is specified, then the framework uses internal defaults that assume the data is well-formatted PAM business-object-compatible XML.

- The name-value pairs used to describe individual module properties are defined by each module.
- The transport_internal elements override the default property values set in the Channel Manager. For more information about these values, see *Setting default transport property values* on page 15.

## Structure of the CTAssignments.xml file

This figure illustrates the overall structure of the **CTAssignments.xml** file.



```
<assignments>
  <default_assignment>
    <inbound_default>
      ...
    </inbound_default>
    <outbound_default>
      ...
    </outbound_default>
  </default_assignment>
  <override_assignment>
    <constraint>
      ...
    </constraint>
    <modules>
      ...
    </modules>
  </override_assignment>
</assignments>
```

The file contains only one <default_assignment> section. It contains sections for inbound and outbound defaults.

The sections list the default modules and their default properties. These sections, are described in detail in *Default assignments* on page 22.

The file can contain as many <override_assignment> sections as you need. The override assignments are shown in greater detail in *Override assignments* on page 23.

The constraints allow you to specify overrides for particular business object types, Partner IDs, or a combination of the two.

This section specifies the modules and properties to be used when the <constraint> conditions are met.

## Default assignments

The portion of the **CTAssignments.xml** file that defines defaults has this form:

```
<assignments>
 <default_assignment>
  <inbound_default>
   <transport>
    <module>
     <module_name>NAME</module_name>
     <module_properties>
      <property>
       <name>PROP-NAME</name>
       <value>PROP-VALUE</value>
      </property>
     </module_properties>
    </module>
   </transport>
   <packager>
    ...
   </packager>
   <transform>
    ...
   </transform>
  </inbound_default>
  ...
 </default_assignment>
...
```

The default assignments <transport> section must contain one <module> section.

The <module_properties> section can contain one or more <property> sections. Each module defines its own <name>/<value> pairs. There must be one <property> section for each property specified by the module; required properties are identified in ModuleMetadata.txt.

The name and value pairs are defined by the metadata for each module. For more information, see *ModuleMetadata.txt* on page 28

You specify the default module and module properties for the <packager> and <transform> sections in the same way as shown for the <transport> section.

## Override assignments

The portion of the **CTAssignments.xml** file that defines override properties has this form:

```
<assignments>
  ...
  <override_assignment>
    <constraints>
     <bo_type_id>ECO!2.ECO.1</bo_type_id>
     <partner_id>UserCT.109654</partner_id>
    </constraints>
   <modules>
     <inbound_override>
       <transform>
         <module>
           <module_name>SimpleInXform</module_name>
           <module_properties>
             <property>
               <name>InXform1</name>
               <value>ASSIGN1_InXform1</value>
             </property>
           </module_properties>
         </module>
       </transform>
     </inbound_override>
   </modules>
  </override_assignment>
  ...
</assignments>
```

Each <override_assignments> section contains one <constraints> section, followed by one <modules> section.

This example override specifies both a business object type ID (from BOTypeIDs.txt) and a partner ID (from PartnerIDs*NNNN*.txt).

The <modules> section can contain override sections for both inbound and outbound data.

The <module_properties> section can contain multiple <property> sections; each <property> section can contain only one <name>/<value> pair. Each module defines its own <name>/<value> pairs.

The name and value pairs are defined by the metadata for each module. For more information, see *ModuleMetadata.txt* on page 28

## Order of precedence

It is possible for business object type and partner ID overrides to conflict. In cases where the constraints specify both a particular business object type and a partner ID, the business object type override takes precedence. This table illustrates the order of precedence:

| | |
|---|---|
| 1 | Business object type and partner ID override |
| 2 | Business object type override |
| 3 | Partner ID override |
| 4 | Default |

## Using channel and partner properties

The Channel Toolkit framework allows you to use special strings to refer to Partner Agreement Manager password store entries, channel properties, and partner properties from the `CTAssignments.xlm` file. These **dereference values** are replaced with the actual values from Partner Agreement Manager at runtime.

To use a dereference value, prefix the Partner Agreement Manager property name with `$CHANNEL_`, `$PARTNER_`, or `$PASSWORD_`, as appropriate.

For example:

- To include the Default_Retry_Frequency channel property, use `<value>$CHANNEL_Default_Retry_Frequency</value>`.
- To include the Basic_Authentication_Username partner property, use `<value>$PARTNER_Basic_Authentication_Username</value>`.

A password store entry is the username corresponding to a password in the Partner Agreement Manager password store. If the password store entry includes a partner ID, you can include the string "`$PARTNERID$`" in the password store entry, which is replaced with the partner ID before the lookup. For example, if you have a password store entry in the form CTInbound777, you can refer to this in the `CTAssignments.xml` file with:

```
$PASSWORD_CTInbound$PARTNERID$
```

When this value is resolved for partner TestPartner.777, the password store entry would be CTInbound777 (the $PARTNERID$ has been replaced with 777). When the partner ID is 8743, the password store entry would be CTInbound8743.

### Validation

The Channel Toolkit framework validates the property names that follow $CHANNEL_ and $PARTNER_, to ensure that the properties actually exist in the channel or partner profiles.

As a security measure, this is not done for $PASSWORD_. Otherwise someone could use this facility to check what passwords are in the password store.

## Creating helper files

When the Channel Toolkit processes CTAssignments.xml, it creates three helper files that assist you in modifying the CTAssignments.xml file:

- PartnerIDs*NNNN*.txt - lists partners using Channel ID *NNNN*.
- BOTypeIDs.txt - lists business object types.
- ModuleMetadata.txt - provides sections of XML data and metadata information used to configure modules.

These files are also created when you validate the CTAssignments.xml file through the Channel Manager; the files are described in greater detail in the following sections.

### PartnerIDsNNNN.txt

PartnerIDs*NNNN*.txt contains a list of partners using ChannelID *NNNN*. When you need to identify overrides based on partner ID, you can reduce typographical errors by copying partner IDs from this file and pasting them into the CTAssignments.xml file for override constraints.

The file looks like this:

```
----------------------------------------------
----------------------------------------------
Filename   : PartnerIDs1301.txt
Description: Describes the legal PartnerID values
         for channel: ChannelToolkit_Example1301.1301
(Note: This file was automatically generated at 4/24/01 5:52 PM)
----------------------------------------------
----------------------------------------------
SomeoneCT.109654
TestToolkitPartner.45678
```

## BOTypeIDs.txt

**BOTypeIDs.txt** contains a list of business object type IDs known to Partner Agreement Manager. When you need to identify overrides based on business object, you can reduce typographical errors by copying business object type IDs from this file and pasting them into the **CTAssignments.xml** file for override constraints.

**Note:** When copying the BOTypeID, use the Internal BOTypeID, not the External Document DefID.

The **BOTypeIDs.txt** file looks like this:

```
----------------------------------------------
----------------------------------------------
Filename   : BOTypeIDs.txt
Description: Describes the legal DocumentID values
         in this system(internal and external)
 BOTypeID    - Internal Document Type Identifier
 ExtDocDefID - External Document Type Identifier

 (Note: This file was automatically generated at 4/24/01 5:52 PM)
----------------------------------------------
----------------------------------------------
```

```
DocumentID - ActionStatus
---------------------------------------------
 - BOTypeID                : ActionStatus!1.ActionStatus.1
 - DOCTYPE for this BOTypeID   : <!DOCTYPE ActionStatus SYSTEM
"1.ActionStatus.1">
 - External Document DefID     : null

DocumentID - Operation_Status
---------------------------------------------
 - BOTypeID                : Operation_Status!3.Operation_Status.1
 - DOCTYPE for this BOTypeID   : <!DOCTYPE Operation_Status SYSTEM
"3.Operation_Status.1">
 - External Document DefID     : null
```

In cases where the business object type ID includes an external element definition set, the External Document DefID line is followed by another DOCTYPE line that gives the URL for the DTD.

The external ID is defined ahead of time by the creator of the business object, using the Element Definition Set Editor. The creator must also freeze the business object. For more information about using the Element Definition Set Editor, see the *Partner Agreement Manager User's Guide*.

Here is business object type ID that includes an External Document DefID:

```
DocumentID - OrderInquiry
---------------------------------------------
 - BOTypeID                    : OrderInquiry!1.iq39am35.8sm.1
 - DOCTYPE for this BOTypeID    : <!DOCTYPE OrderInquiry SYSTEM "1.iq39am35.8sm.1">
 - External Document DefID       : http://www.yourcompany.com/DTDs/OrderInquiry.dtd
 - DOCTYPE for this ExtDocDefID : <!DOCTYPE OrderInquiry SYSTEM "http://www.yourcompany.com/
DTDs/OrderInquiry.dtd">
```

## ModuleMetadata.txt

ModuleMetadata.txt contains property information for each of the modules listed in the CTModules.properties file. It is generated by the framework at start-up and update time. The information takes the form of XML sections that define assignments; you can cut and paste these into CTAssignments.xml. Because the file contains information for *all* modules listed in the CTModules.properties file, it might include modules that aren't used by the instance you're configuring.

The ModuleMetadata.txt file typically looks like this:

```
----------------------------------------
ModuleMetadata - (SimplePollXport)
----------------------------------------
Module Name      : SimplePollXport
Metadata Class   : com.yourcompany.SimplePollingMetadata
Module Type      : TRANSPORT
Supports Inbound  : TRUE
Supports Outbound : FALSE
Externally Loaded : FALSE
Polling Transport : TRUE
Supports Async   : TRUE
Supports Sync    : FALSE
Supports Secure  : FALSE
Property Names   : File_Extension Inbox_Directory Partner_ID Outbox_Directory
CTAssignments.xml XML fragment :

    <transport>
      <module>
        <module_name>SimplePollXport</module_name>
        <module_properties>
          <description></description>
          <property>
            <name>File_Extension</name>
            <value>__REPLACE_VALUE__</value>
          </property>
          <property>
            <name>Inbox_Directory</name>
            <value>__REPLACE_VALUE__</value>
          </property>
          <property>
            <name>Partner_ID</name>
            <value>__REPLACE_VALUE__</value>
          </property>
          <property>
            <name>Outbox_Directory</name>
            <value>__REPLACE_VALUE__</value>
          </property>
        </module_properties>
      </module>
      <transport_internal>
        <polling_frequency>3</polling_frequency>
      </transport_internal>
    </transport>
```

After cutting and pasting into `CTAssignments.xml`, you replace all the "`__REPLACE_VALUE__`" placeholders with the appropriate information.

## Creating assignments

The following sections describe how to perform common tasks in creating or updating default and override assignments.

### Defining a default assignment

**To define a default assignment, you perform these steps:**

1. Edit the appropriate `CTAssignments.xml` file.
2. Copy a module section for each of these three sections from the `ModuleMetadata.txt` file to the appropriate section within the <inbound_default> section:
   - transport - required
   - packager - optional
   - transformation - optional

   In each copied section, replace the "`__REPLACE_VALUE__`" placeholders with the appropriate values or strings.

3 Copy a module section for each of these three sections from the ModuleMetadata.txt file to the appropriate section within the <outbound_default> section:

- transport - required
- packager - optional
- transformation - optional

In each copied section, replace the "__REPLACE_VALUE__" placeholders with the appropriate values or strings.

## Defining an override assignment

You can create overrides based on:

- Business object type
- Partner
- Both business object type and partner

For either of these constraints, you can further specify a direction (inbound, outbound, or both).

**To define an override:**

1 Edit the appropriate CTAssignments.xml file.

2 Add a new <override_assignment> section to the file.

3 Create a <constraints> section.

- To create an override for a particular business object, add a <bo_type_id> section to the <constraints> section.

  To avoid typos, you can copy the business object type ID from BOTypeIDs.txt (for more information, see *BOTypeIDs.txt* on page 26).

- To create an override for a particular partner ID, add a <partner_id> section to the <constraints> section. To avoid typos, you can copy the partner ID from PartnerIDs*NNNN*.txt (for more information, see *PartnerIDsNNNN.txt* on page 25).

- You can specify both a business object type ID and a partner ID in the <constraints> section:

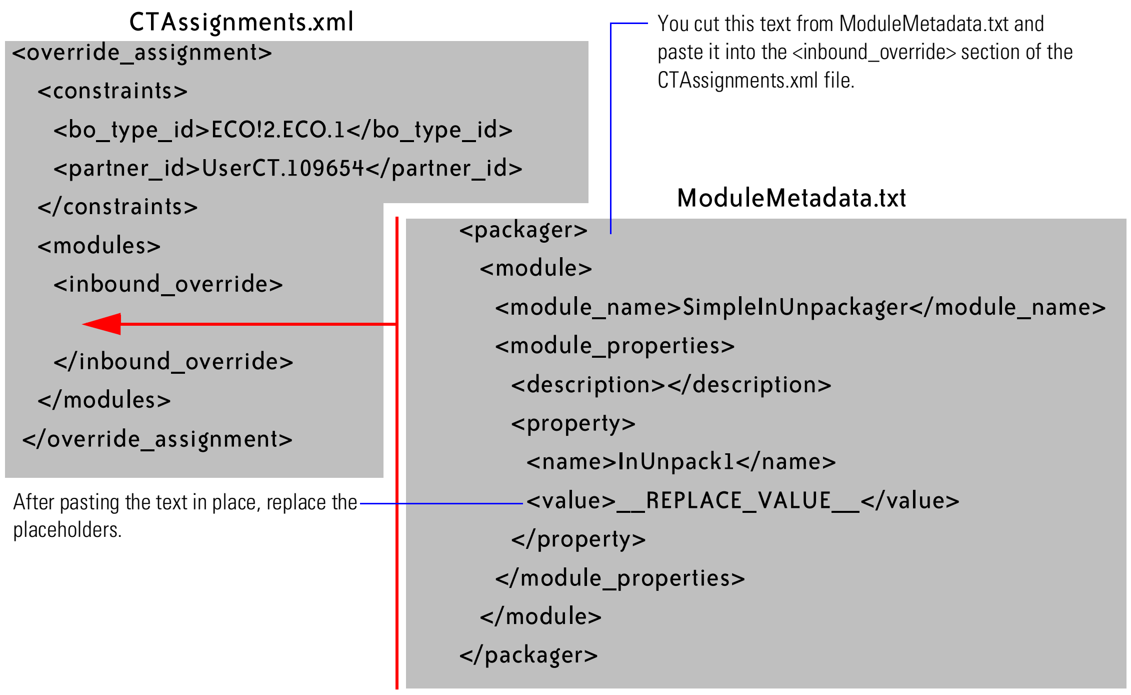
```
<constraints>
 <bo_type_id>ECO!2.ECO.1</bo_type_id>
 <partner_id>UserCT.109654</partner_id>
</constraints>
```

4 Create a <modules> section following the <constraints> section.

5 For inbound assignments, add an <inbound_override> section to the <modules> section; for outbound assignments, add an <outbound_override> section. To define an override that applies to both inbound and outbound data, you must create both an <inbound_override> and an <outbound_override> section and duplicate the tags enclosed in those sections.

```
<modules>
  <inbound_override>
  </inbound_override>
</modules>
```

6 Copy the appropriate module section (<transform>, <packager>, or <transport>) from the **ModuleMetadata.txt** file and insert it in the appropriate <inbound_override> or <outbound_override> section (for more information, see *ModuleMetadata.txt* on page 28). This figure shows an inbound override assignment being added to **CTAssignments.xml**:



CTAssignments.xml

```
<override_assignment>
  <constraints>
    <bo_type_id>ECO!2.ECO.1</bo_type_id>
    <partner_id>UserCT.109654</partner_id>
  </constraints>
  <modules>
    <inbound_override>

    </inbound_override>
  </modules>
</override_assignment>
```

You cut this text from ModuleMetadata.txt and paste it into the <inbound_override> section of the CTAssignments.xml file.

ModuleMetadata.txt

```
<packager>
  <module>
    <module_name>SimpleInUnpackager</module_name>
    <module_properties>
      <description></description>
      <property>
        <name>InUnpack1</name>
        <value>__REPLACE_VALUE__</value>
      </property>
    </module_properties>
  </module>
</packager>
```

After pasting the text in place, replace the placeholders.

7 In each copied section, replace the "__REPLACE_VALUE__" placeholders the appropriate values or strings.

## Validating and using CTAssignments.xml

After modifying `CTAssignments.xml` for a particular Channel Toolkit instance, you must process the file to validate it and update the instance's run-time properties.

**To process CTAssignments.xml:**

1. Launch PAM and open the Channel Manager.
2. Click on Channel Profiles.
3. Choose the Channel Toolkit instance you are configuring and click Edit>Properties.
4. Click on the Other tab.
5. Ensure Update_Config_Info is set to True and click OK.

When you click OK, the Channel Toolkit generates new versions of the three helper files (described in *Creating helper files* on page 25) and validates both `CTAssignments.xml` and `CTModules.properties` for the Channel Toolkit instance you selected.

If both files are valid, it Channel Toolkit updates the run-time configuration data for that instance. If either file is invalid, the Channel Toolkit displays an error.

For a remote client installation (Channel Toolkit is installed on the Process Manager computer) , Channel Toolkit displays an error if the Process Manager computer's file are incorrect. If the files on the Process Server and Process Manager computers are out of sync, you might get a Process Server side exception, even though you did not get an error dialog on the Process Manager computer.

In summary, you use the steps listed above to validate your configuration:

- If validation of the configuration files on the Process Manager computer fails, you'll get an error dialog.
- If validation of the configuration files on the Process Server computer fails, you'll get an exception on the Process Server console.

### Returning to the Channel Manager

If you need to perform other tasks from Channel Manager before you can create a valid CTAssignments.xml file (such as add a partner), you can either:

- Set Update_Config_Info to False and click OK.

  When Update_Config_Info is False, no validation is performed.
- Click Cancel.

## Process Manager installations

On UNIX, you must install and run the Channel Toolkit on a computer running Process Manager, in addition to the Process Server computer. (This is an option on Windows NT; it allows you to provide a local facility on which users can edit and validate the CTAssignments.xml and CTModules.properties file.)

Because the Channel Toolkit user interface has no facility for sending the modified CTAssignments.xml, CTModules.properties, and other files from the Process Manager computer to the Process Server computer, you must synchronize these files yourself.

**To do this:**

1. Copy CTAssignments.xml and CTModules.properties to the Process Server computer by hand.
2. On the computer running Process Server, perform the steps listed in *Validating and using CTAssignments.xml* on page 32.

CHAPTER

4

# Designing processes using Channel Toolkit

Read this chapter for specific information on how to design processes that will be used with Channel Toolkit. Sections in this chapter include:

- *About public and private processes* on page 36.
- *About Channel Toolkit and public processes* on page 36.
- *Synchronous partners and public processes* on page 44.

## About public and private processes

Partner Agreement Manager's two-level process model is designed to automate transactions and the exchange of information between partners in an extended enterprise. This model promotes cooperation between partners, yet maintains total security and flexibility for each partner's internal activities.

The first level is the **public process**, which defines the step-by-step flow of information and actions between two or more partners in an extended enterprise. One partner develops the public process, and all partners involved review and accept the process before it is implemented. The partner who designs a process is its owner.

The second level is the **private process**, which defines each partner's internal sequence of actions for its steps in the public process. Although all partners in a public process see and agree to its flow, each partner's private process is just that—private. The partner who develops a private process is the only one who can ever see it.

See the *Partner Agreement Manager User's Guide* for design guidelines and a complete explanation of how to design and develop processes.

## About Channel Toolkit and public processes

Partner Agreement Manager supports Channel Toolkit partners in public processes. A Channel Toolkit partner is a partner that exchanges data with a Partner Agreement Manager public process via a Channel Toolkit channel. As a host partner, you can construct a variety of processes that use Channel Toolkit.

### Processes you can create using Channel Toolkit

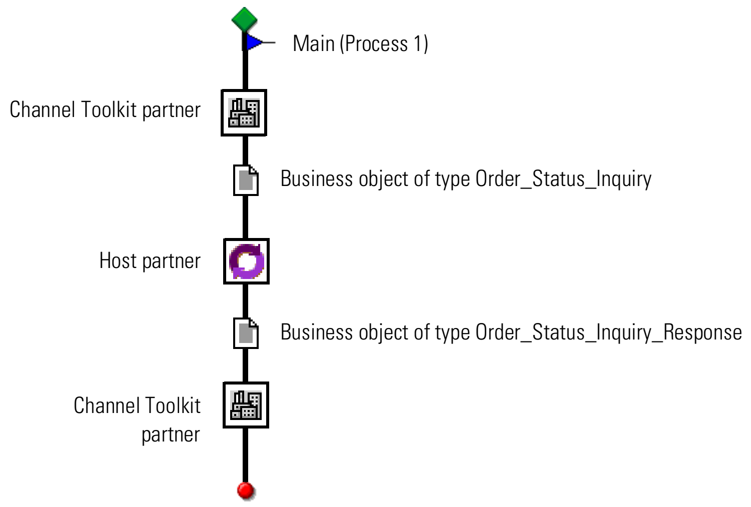
You can create a large variety of processes using Channel Toolkit, utilizing all the richness of the Partner Agreement Manager process development tools. These processes fall into four very general types:

- the Channel Toolkit partner initiates the process
- the host partner initiates the process
- the Channel Toolkit partner completes the process

- the host partner completes the process

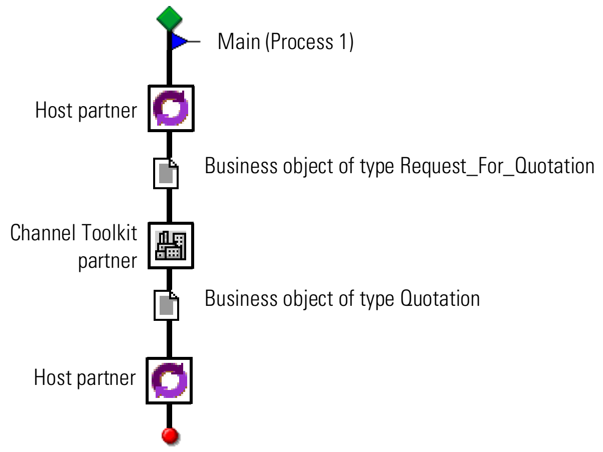
## The Channel Toolkit partner initiates the process

The Channel Toolkit partner can initiate a process. For example, if the Channel Toolkit partner is buying something from the host partner, the Channel Toolkit partner might issue an order status inquiry:


Main (Process 1)
Channel Toolkit partner
Business object of type Order_Status_Inquiry
Host partner
Business object of type Order_Status_Inquiry_Response
Channel Toolkit partner

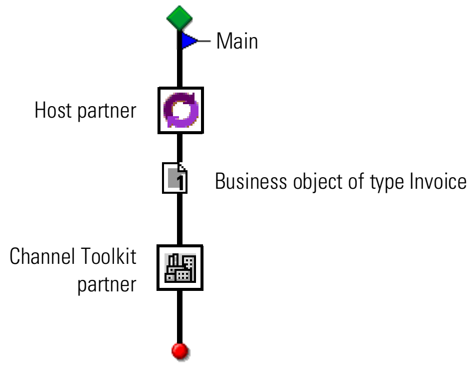
## The host partner initiates the process

The host partner can initiate a process that includes a Channel Toolkit partner. For example, if the host partner is buying something from the Channel Toolkit partner, the host partner might start by asking for a quotation:


Main (Process 1)
Host partner
Business object of type Request_For_Quotation
Channel Toolkit partner
Business object of type Quotation
Host partner

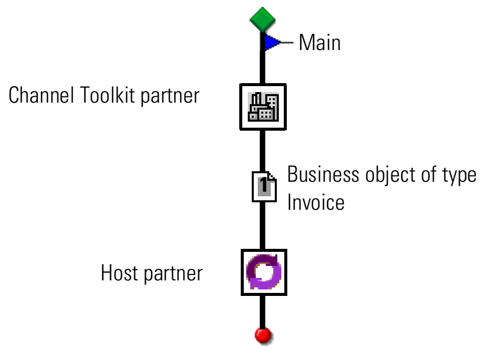
## The Channel Toolkit partner completes the process

The Channel Toolkit partner can be the terminal step of a process. This is shown in the first example (*The Channel Toolkit partner initiates the process* on page 37). In that example, not only did the Channel Toolkit partner initiate the process, they also completed it. Another example would be if the Channel Toolkit partner were the buyer and the host partner sent an invoice:



### The host partner completes the process

The host partner can be the terminal step of a process. For example, if the host partner was a buyer and the Channel Toolkit partner sent an invoice:



## Invalid processes detected by verifier

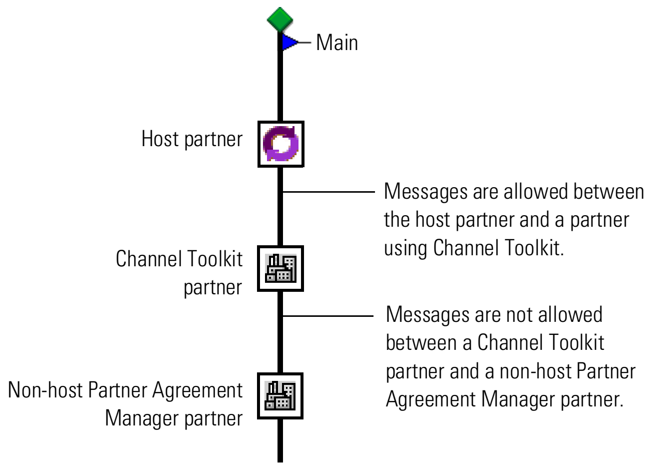
When designing processes that include a Channel Toolkit partner, there are a few design considerations you must take into account. These considerations reflect a few corner cases and are unlikely to be present in your process design. The process verifier rejects any process that does not conform to these design considerations.

**Note:** The consideration listed in this section apply to partners that communicate asynchronously. There are more invalid processes that can exist for partners that communicate synchronously. For the synchronous partner cases, see *Synchronous partners and public processes* on page 44.

### A Channel Toolkit partner can communicate only with the process owner

The process owner cannot author a process in which a non-host Partner Agreement Manager partner communicates directly with a partner using Channel Toolkit.

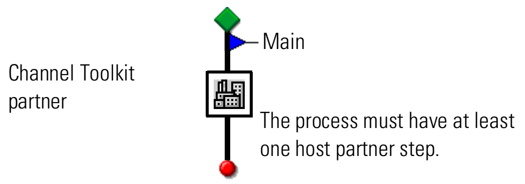
For example, this process flow is not valid:


Main
Host partner
Messages are allowed between the host partner and a partner using Channel Toolkit.
Channel Toolkit partner
Messages are not allowed between a Channel Toolkit partner and a non-host Partner Agreement Manager partner.
Non-host Partner Agreement Manager partner

## You cannot author processes that contain only a Channel Toolkit partner

A process that contains a Channel Toolkit partner must also contain at least one host partner step. The single step of a single step process cannot contain only a Channel Toolkit partner, either as a static partner or a group member.

For example, the following configuration is not valid:



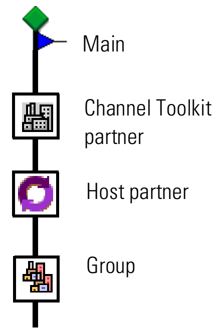
A Channel Toolkit partner step cannot be:

- immediately preceded or followed by a step assigned to anyone except the process owner or
- the single step of a single step process.

## A Channel Toolkit partner cannot initiate a parametric process

A Channel Toolkit partner may not be the non-group initiator of a process which contains a group. In such a case, Partner Agreement Manager will not be able to determine which group member should be selected to participate in this process instance.

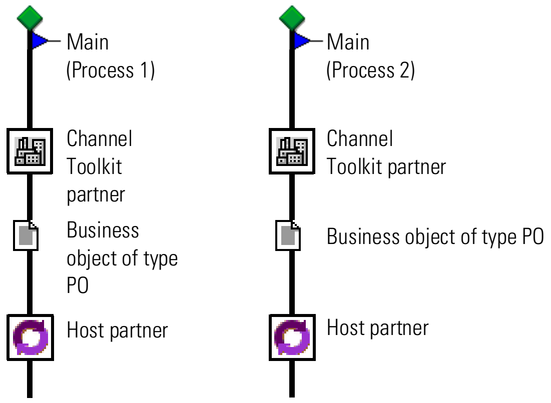
For example, this process is not valid:



## A Channel Toolkit partner cannot initiate multiple processes with the same business object type

One Channel Toolkit partner cannot initiate multiple processes with the same business object type. In such a case, Partner Agreement Manager cannot determine which process to initiate.

For example, these two processes cannot both be installed:



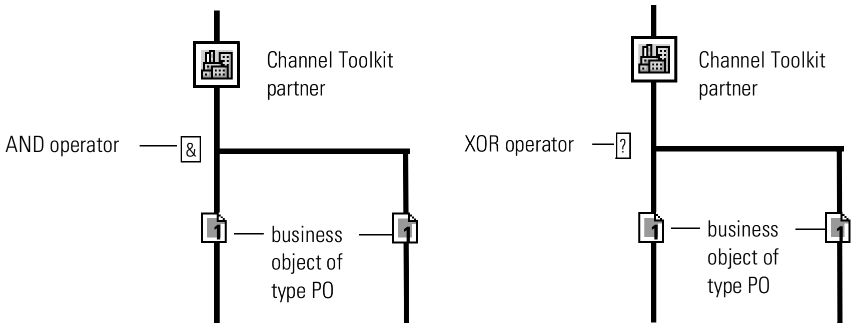
Note that two different Channel Toolkit partners could initiate different processes with the same business object type.

The default configuration includes this design consideration. If you need this functionality, contact IBM customer support about changing the configuration.

### A Channel Toolkit partner cannot output two or more business objects of the same type

A Channel Toolkit partner cannot send two or more business objects of the same type from a single step. In such a case, the channel cannot determine the process path for each business object. This applies to both conjunctive and disjunctive out paths.

For example, neither of these processes is valid:

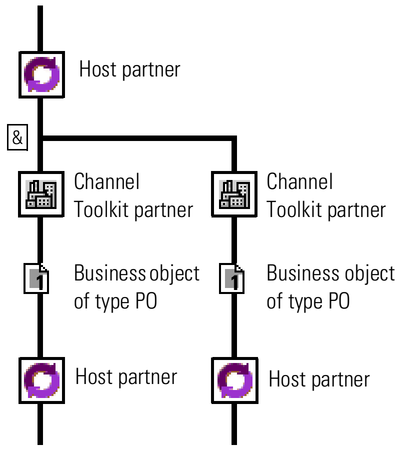


The default configuration includes this design consideration. If you need this functionality, contact IBM customer support about changing the configuration.

### A Channel Toolkit partner cannot output the same business object type on concurrent paths

A Channel Toolkit partner cannot output the same business object type on concurrently executing paths. In such a case, the channel cannot identify the path to which a newly arrived business object belongs. This applies only to conjunctive paths.

For example, this process is not valid:

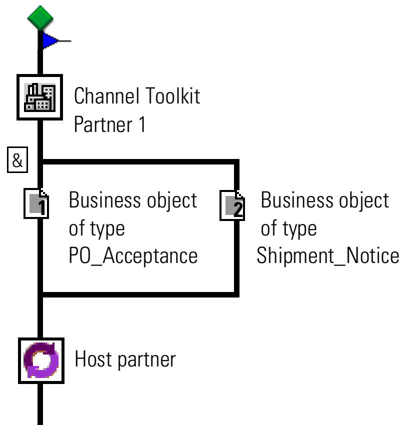


The default configuration includes this design consideration. If you need this functionality, contact IBM customer support about changing the configuration.

## A Channel Toolkit partner cannot initiate a process with two or more business objects on conjunctive paths

If a Channel Toolkit partner is the initiating step, it cannot output two or more business objects on conjunctive paths.

For example, this process is not valid:


Channel Toolkit Partner 1
&
Business object of type PO_Acceptance
Business object of type Shipment_Notice
Host partner

The default configuration includes this design consideration. If you need this functionality, contact IBM customer support about changing the configuration.

## Synchronous partners and public processes

Previous examples in this chapter have involved partners that communicate asynchronously; that is, each request and response is handled through a separate connection. The processes introduced in this section involve partners that communicate synchronously; that is, one connection (either inbound or outbound) is used for both a request and a response. In the following section, partners that communicate synchronously are called **synchronous partners.**

Synchronous partners have one input business object to a public node and one output business object from a public node.

### Invalid processes not detected at design time

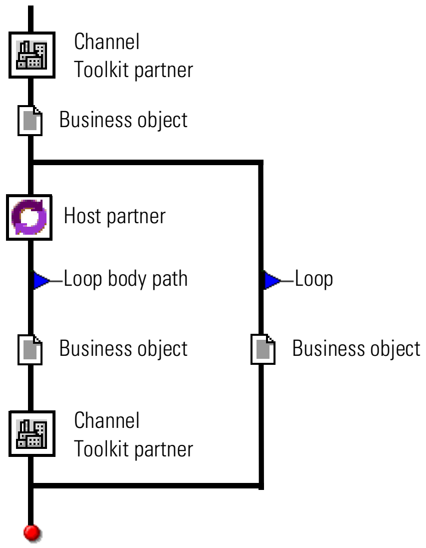
Some asynchronous process configurations are invalid, but cannot be caught by the process verifier because there is nothing in the process definition that specifies whether the partners communicate in synchronous or asynchronous mode. The partners communication mode is set as a partner property.

The errors in these configurations will show up at run-time when testing your process.

#### Non-Partner Agreement Manager Channels should not allow a terminating loop

Channels that do not pass outbound control messages cannot have a loop branch coming out of them if they are the last node. Partner Agreement Manager is not able to tell when to terminate the node; it's ambiguous.

For example, this process is not valid:



### Synchronous partners can only have one input, one output

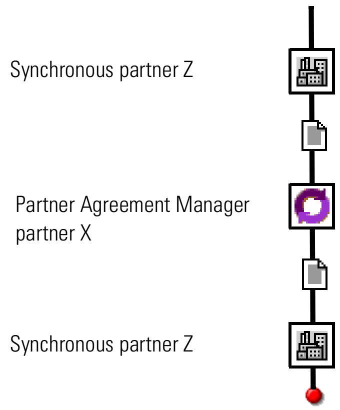
When a process includes a synchronous partner, there can be only one business object input to the synchronous partner and a maximum of one business object output via the completion set.

### The same synchronous partner must precede and follow an Partner Agreement Manager partner

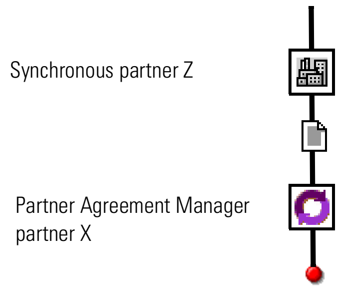
A synchronous partner must either:

- Both precede and follow a Partner Agreement Manager partner. Both synchronous partners must have the same PartnerID.
- Precede a Partner Agreement Manager partner, which must be the terminal step in the process.

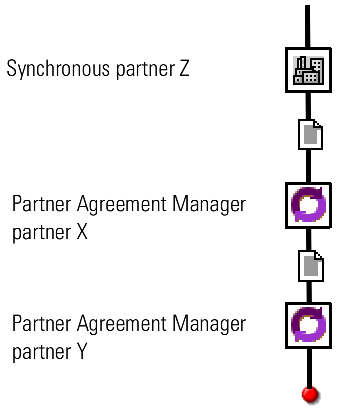
This first example shows a valid process using a synchronous partner. In this case, the Partner Agreement Manager partner is sandwiched by a synchronous partner.



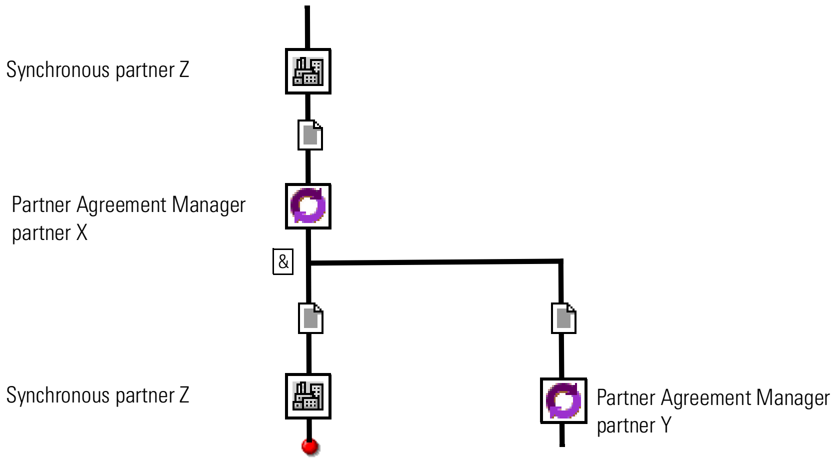
This example also shows a valid process using a synchronous partner. In this case, the Partner Agreement Manager partner is preceded by a synchronous partner and is also the terminal step.



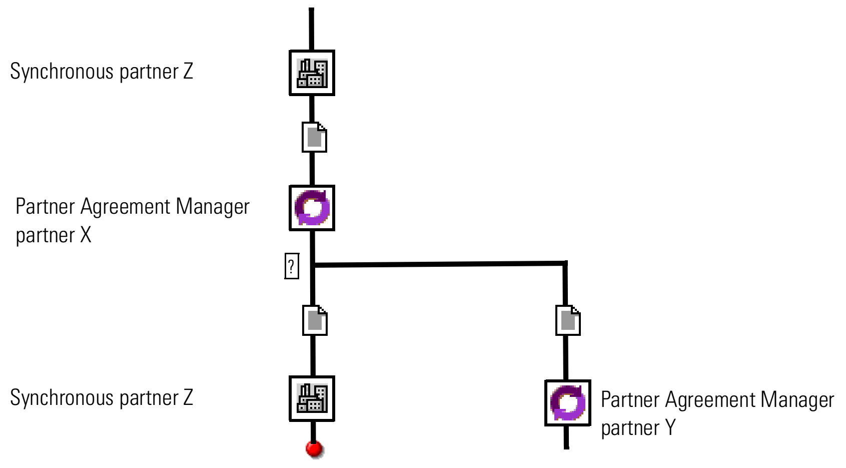
The following is a valid variation of the previous example. In this case, the Partner Agreement Manager partner is preceded by a synchronous partner and is followed by a second Partner Agreement Manager partner, which is then followed by the terminal step.


Synchronous partner Z
Partner Agreement Manager partner X
Partner Agreement Manager partner Y

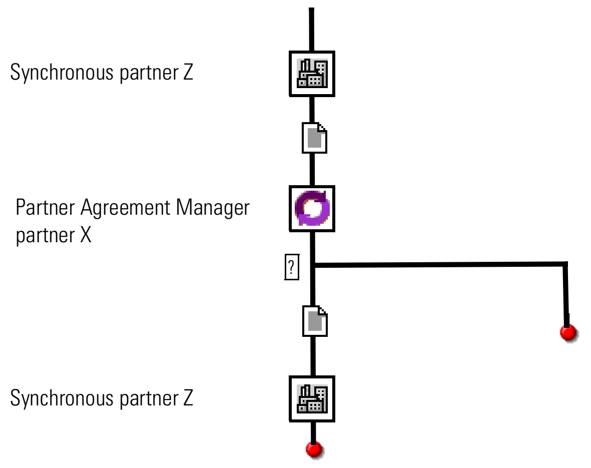
The following is a valid variation of the first and third examples. In this case, the Partner Agreement Manager partner is preceded by a synchronous partner and is followed, using an AND branch, by both the same synchronous partner and a second Partner Agreement Manager partner. The important thing here is that the AND branch makes the second synchronous partner Z required.


Synchronous partner Z
Partner Agreement Manager partner X
&
Synchronous partner Z
Partner Agreement Manager partner Y

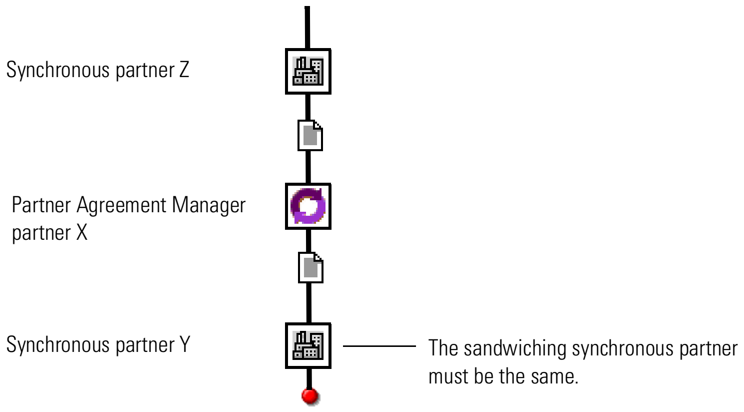
The following process is invalid for synchronous partners, because the OR doesn't require execution of the process to follow the synchronous partner branch.


Synchronous partner Z
Partner Agreement Manager partner X
?
Synchronous partner Z
Partner Agreement Manager partner Y

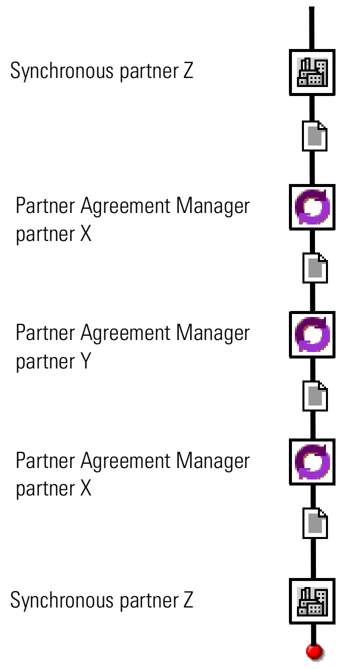
In this example, the previous example is made valid by having synchronous partner Z on one path and a termination on the other.



This process is invalid, because the Alliance Manager partner is not sandwiched by the same synchronous partner.


Synchronous partner Z
Partner Agreement Manager partner X
Synchronous partner Y
The sandwiching synchronous partner must be the same.

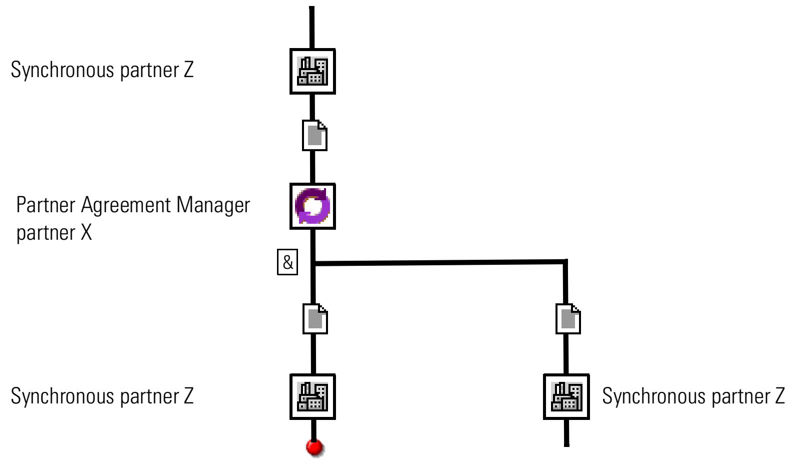
This final example of invalid processes that involve synchronous partners shows a synchronous partner that does not directly sandwich an Partner Agreement Manager partner. A possible workaround is to have Partner Agreement Manager partner X call Partner Agreement Manager partner Y as a subprocess.


Synchronous partner Z
Partner Agreement Manager partner X
Partner Agreement Manager partner Y
Partner Agreement Manager partner X
Synchronous partner Z

## Synchronous partner must receive exactly one response object

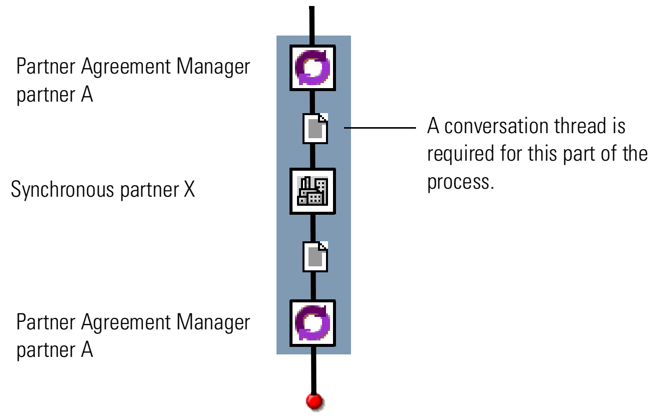
If an Partner Agreement Manager partner is sandwiched by a synchronous partner, the Partner Agreement Manager partner must emit exactly one response object, unless the process terminates after the Partner Agreement Manager partner.

This examples shows an invalid process, because synchronous partner Z receives two response objects.



## Conversation thread required

If X is a client partner and A is an Partner Agreement Manager partner, you must define a conversation for processes of the form A-X-A.


Partner Agreement Manager partner A
A conversation thread is required for this part of the process.
Synchronous partner X
Partner Agreement Manager partner A

For more information on conversation threads, see *Creating conversations* on page 53.

CHAPTER

5

# Creating conversations

Partner Agreement Manager matches up responses to requests by associating business objects it receives with those that it has sent. Usually this information is maintained as process metadata between Partner Agreement Manager and its partners, but in some cases, (such as Channel Toolkit channels), the partner has no provision for maintaining this process metadata. Thus, Channel Toolkit uses a feature called conversation threading to maintain the process context of its messages.

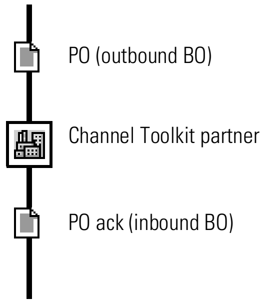
Read this chapter for information about creating conversations that allow Partner Agreement Manager to identify responses from a Channel Toolkit partner.

- *About conversations* on page 54.
- *About the conversation editor* on page 56.
- *Creating conversations* on page 59.

## About conversations

The *conversation threading* feature in Channel Toolkit allows you to use unique information shared by outbound and inbound business objects (such as a PO number) to route inbound business objects to the appropriate Partner Agreement Manager process.

When a public process includes a Channel Toolkit partner, you usually pass a business object to the partner and, in many cases, the Channel Toolkit partner returns another business object.



The conversation associates fields in the outbound business object (in this case a purchase order) and uses the values in these fields to associate the inbound message (a purchase order acknowledgement) with the appropriate process.

**Note:** You cannot create two conversations that use the same inbound and outbound business objects, even if the fields used in conversations are different.

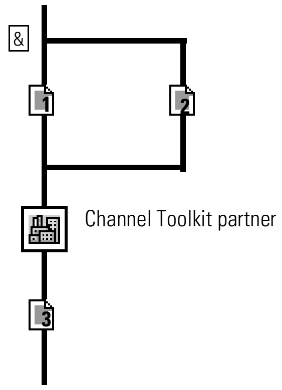
If conversation threading is turned on an you do not create a conversation for the outbound and inbound business objects used by a Channel Toolkit partner, the Partner Agreement Manager process verifier issues an error, stating that the conversation is not defined.

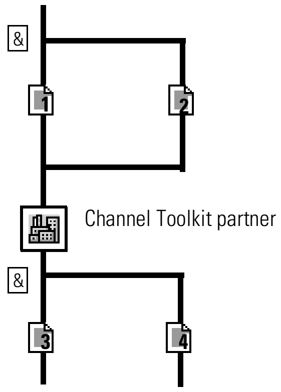
A conversation is global. Once you have created a conversation, it is used to evaluate all inbound messages routed through the channel.

If Partner Agreement Manager receives an inbound business object from a Channel Toolkit partner, but cannot match the business object to a running process, Partner Agreement Manager looks for a process that can be started by the Channel Toolkit partner using that business object. If Partner Agreement Manager finds such a process, it starts it; if no process is found, Partner Agreement Manager reports an error initiating a process.

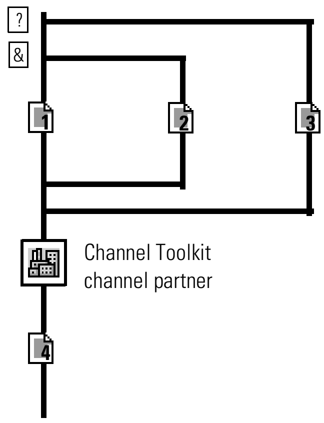
## Requests with multiple business objects

Each conversation can have one or more outbound business objects, joined with an AND, but can only have one inbound business object. For example, the process flow on the left can be identified by a single conversation (object 1 and 2 outbound with object 3 inbound). Whereas the flow on the right must be identified by two separate conversations (object 1 and 2 outbound with object 3 inbound; object 1 and 2 outbound with object 4 inbound):





If the outbound business objects are joined by an OR, each outbound set of business objects requires a separate conversation. For example, this process flow requires two conversations (object 1 and 2 outbound with object 4 inbound; object 3 outbound with object 4 inbound).



## About the conversation editor

This section describes some of the general functionality found in the Conversation editor. For information on how to start the Conversation editor and create conversations, see *Creating conversations* on page 59.

When creating a conversation, you start by selecting the inbound business object type (that is, the business object returned from the Channel Toolkit channel partner). Then, you select one or more outbound business object types (the business object passed to the channel Channel Toolkit partner).

The Conversation editor is divided into two panels: the outbound panel and inbound panel.

- The outbound panel (on the left) shows the business object type that is sent to the Channel Toolkit partner.
- The inbound panel (on the right) shows the business object types that can be returned from the Channel Toolkit partner.

**NOTE:** At times the terms "outbound" and "inbound" can be confusing. Just remember that the perspective of the Conversation editor (and this documentation) is from the Partner Agreement Manager perspective. Business objects are "outbound" when they are being sent from Partner Agreement Manager to the Channel Toolkit partner; business objects are "inbound" when they are sent from the Channel Toolkit partner to Partner Agreement Manager.

The lower portions of each panel display the description of an element selected in the corresponding upper portion.

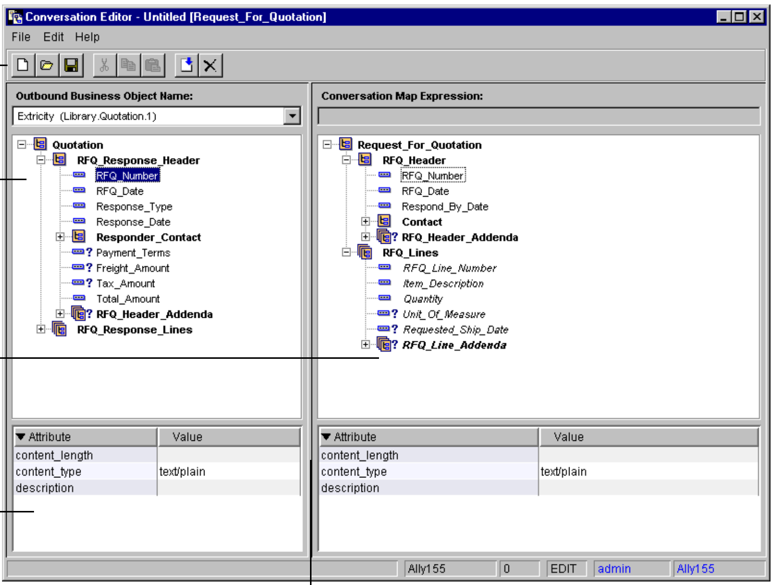
You can resize the panels by dragging their borders.

Toolbar buttons give you quick access to common commands.

This panel shows the business object type being sent to the Channel Toolkit partner.

This panel shows the hierarchy of elements in business object type being returned from the Channel Toolkit partner.

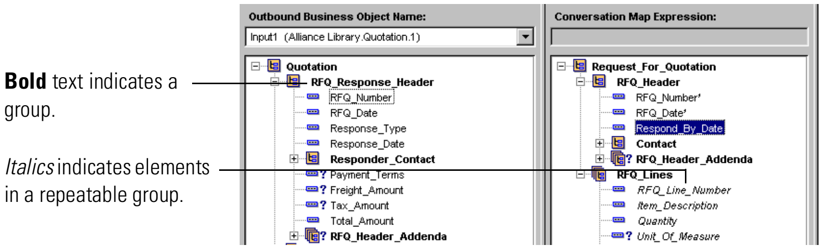
This area shows a description of the selected element.



Drag a panel's border to resize the panel.

The Conversation editor uses the same icons and graphical cues that appear in the Map editor.

These are the font cues in the outbound and inbound panels.



This table summarizes the icons that can appear in the Conversation editor.

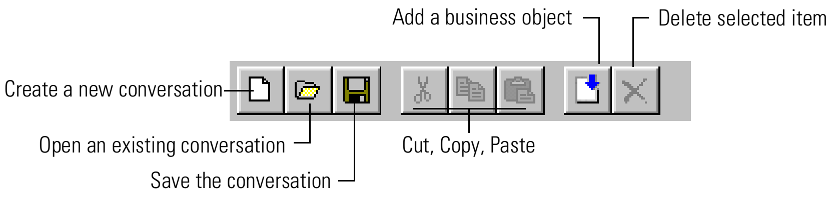
| This icon | Represents | This icon | Represents |
|---|---|---|---|
| ▭ | A required field | ▤ | A required group |
| ▭? | An optional field | ▤? | An optional group |
| ▭▭ | A required field sequence* | ▤▤ | A required group sequence* |
| ▭▭? | An optional field sequence* | ▤▤? | An optional group sequence* |

* This version of the Conversation editor does not allow you to use sequences to identify conversations.

**NOTE:** Unless they are specifically designated as optional (a question mark appears to the right of the icon), all elements are required. This means that any element without a question mark is required in its parent group. We recommend that you only use required fields to identify conversations.

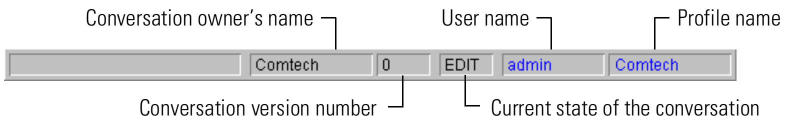
## ABOUT THE COMMAND TOOLBAR

The Command toolbar at the top of the Conversation editor gives you access to common File and Edit menu commands and lets you remove outbound business objects.



### About the status bar

The status bar at the bottom of the Conversation editor shows information about the current user and the state of the map.


Conversation owner's name
User name
Profile name
Comtech
0
EDIT
admin
Comtech
Conversation version number
Current state of the conversation

### About the menu bar

The Conversation editor menu bar appears at the top of the window.

The File menu contains commands for creating, opening, saving, and closing conversations.

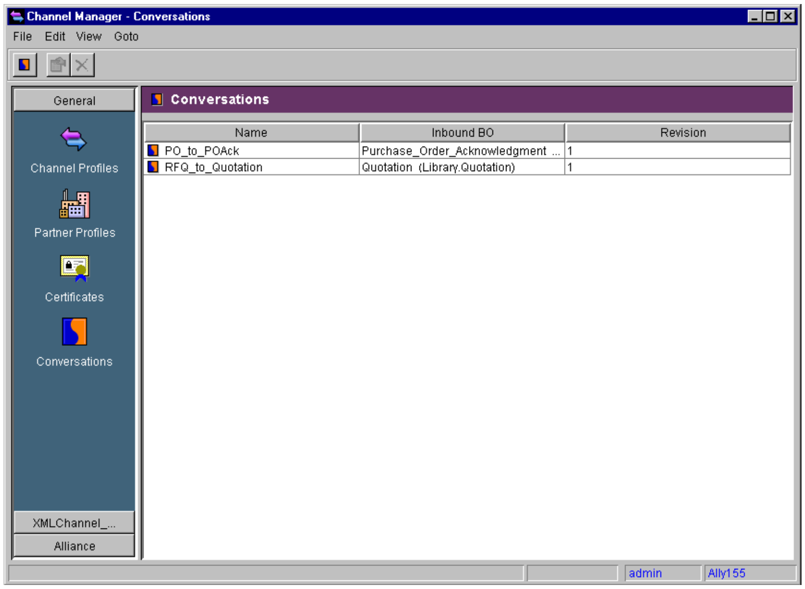
## Creating conversations

The first step in creating a conversation is to select the inbound business object type. You then add outbound business object types as required by the public process structure (described in *About conversations* on page 54) and establish associations between fields in the outbound business objects and the inbound business objects.

**You create conversations in the Channel Manager. To launch the channel manager:**

1. In the Process Manager window, click Actions and select Channel Manager from the Actions menu.

2 The Channel Manager window appears.



| Name | Inbound BO | Revision |
|---|---|---|
| PO_to_POAck | Purchase_Order_Acknowledgment ... | 1 |
| RFQ_to_Quotation | Quotation (Library.Quotation) | 1 |

## To create a conversation:

1 Click the New Conversation button in the Channel Manager’s Command toolbar. This button is only enabled when you select Conversations in the panel on the left.

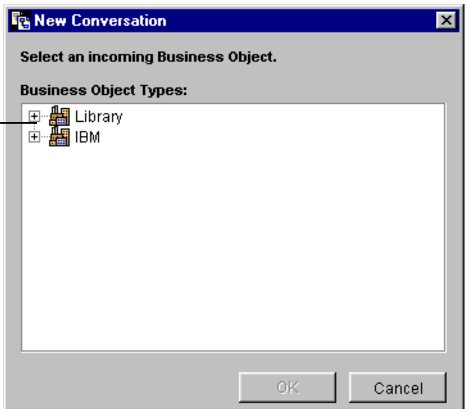
Or, you can click File on the Channel Manager menu and choose New>General>Conversation.

2 The Conversation Editor dialog appears. Click the Create a new conversation button from the command toolbar.

Or, you can click File on the Conversation Editor menu and choose New.

The New Conversation dialog box appears.

Select a business object type to map to (the output of the map).

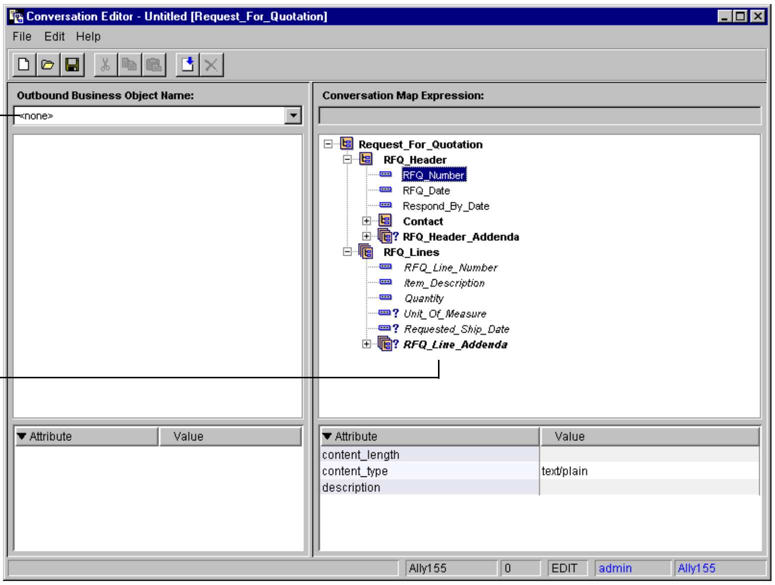


3 Select the inbound business object type that you want associate with this conversation and click OK.

The Conversation editor appears. The business object type you selected appears on the right side of the Conversation editor, in the output area.

When you have multiple outbound business objects in a conversation, you can view them in this list.

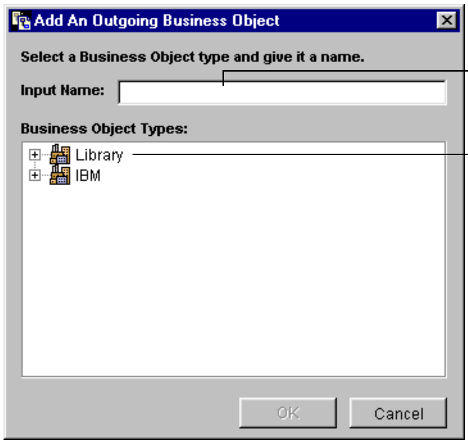
This panel displays the inbound business object type. Double-click a group element to display its contents.



| Attribute | Value |
|---|---|
| content_length | |
| content_type | text/plain |
| description | |

4 To add an outbound business object, click the Add business object button in the Conversation editor's Command toolbar.

Or, you can choose Add Business Object from the Edit menu. The Add Outbound Business Object dialog box appears.


Add An Outgoing Business Object
Select a Business Object type and give it a name.
Input Name:
Business Object Types:
Library
IBM
OK
Cancel

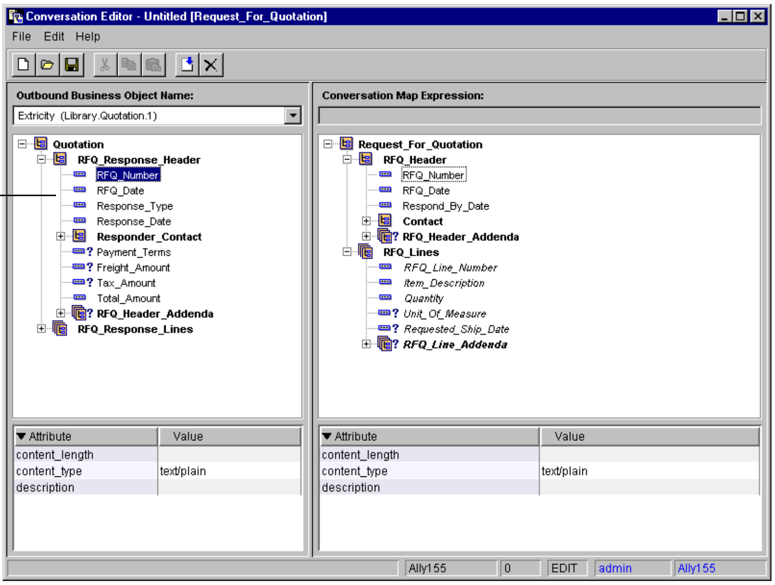
Type a name for the Outbound business object.

Select a business object type to map from (an input).

5 Type in a name for the Outbound business object type.

6 Select an outbound business object type for this conversation. Click OK.

The business object type you select appears in the input panel.


Conversation Editor - Untitled [Request_For_Quotation]
File Edit Help
Outbound Business Object Name:
Conversation Map Expression:

Extricity (Library.Quotation.1)

```
Quotation
  RFQ_Response_Header
    RFQ_Number
    RFQ_Date
    Response_Type
    Response_Date
    Responder_Contact
    ? Payment_Terms
    ? Freight_Amount
    ? Tax_Amount
    Total_Amount
    ? RFQ_Header_Addenda
  RFQ_Response_Lines
```

```
Request_For_Quotation
  RFQ_Header
    RFQ_Number
    RFQ_Date
    Respond_By_Date
    Contact
    ? RFQ_Header_Addenda
  RFQ_Lines
    RFQ_Line_Number
    Item_Description
    Quantity
    ? Unit_Of_Measure
    ? Requested_Ship_Date
    ? RFQ_Line_Addenda
```

| Attribute | Value |
|---|---|
| content_length | |
| content_type | text/plain |
| description | |

| Attribute | Value |
|---|---|
| content_length | |
| content_type | text/plain |
| description | |

Ally155 | 0 | EDIT | admin | Ally155

This panel displays the current outbound business object type. Double-click a group element to display its contents.

7 Expand the groups in the outbound and inbound business object types so that you can see their contents.

8 Associate one or more fields in the outbound business object with the inbound business object by clicking on a field in the outbound business object and dragging it to a field in the inbound business object. When you associate an inbound field, an asterisk (*) appears to the right of its name.

**Note:** In this release of the Conversation Editor, you cannot map sequences (repeatable elements).

You need to associate fields that will have identical content in both the outbound and inbound business object. Partner Agreement Manager uses these fields to identify the conversation.

The names of the fields can be different; the important thing is that the data contained in the fields is identical.

**Note:** We recommend that you always use required fields. If you use an optional field and the field is not present in either the outbound or inbound business object instance, Partner Agreement Manager will not be able to complete the conversation.

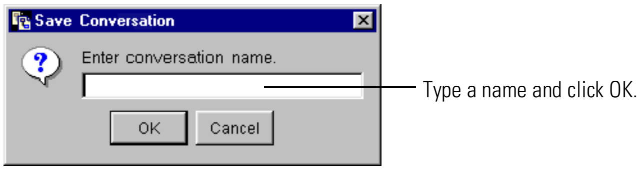
## Saving a conversation

When you save a conversation, you give it a name; the Channel Manager lists it in the Conversations folder.

### To save a conversation:

1 Choose Save from the File menu.

The Save Conversation dialog box appears.



2 Type a name and click OK.

The conversation appears in the Conversations folder in the Channel Manager.

APPENDIX
A

# MODULE INSTALLER ACTIONS

If you develop a module and provide an installer for the module, this appendix describes the actions that your installer must perform. Note that even if you don't provide an installer, you should provide instructions with your module that describe how the user is to install your module.

Sections in this appendix include:

## Location for the module

To install your module class and properties files, you must copy them to in <*Alliance*>\Partners\Partner*XXXX*\com\<*your-company*>.

**Note:** If you are running on UNIX, which requires you to install channel toolkit on both the Process Server and Process Manager computers, or have just chosen to install Channel Toolkit on the both the Process Server and Process Manager computers, you must deploy modules to both computers. For more information see *Process Manager installations* on page 33.

The class names specified in the global module properties of CTModules.properties provide the paths to the files.

## Modifying CTModules.properties

**A module installer must perform these actions:**

1. Open<*Alliance*>\Partner\Partners*XXXX*\Properties\ChannelToolkit\CTModules.properties file.
2. Locate the list of available modules, identified by "Modules=", starting in column 1.
3. Append the module name to the list of available modules.

   For example, to add SimpleOutBound Packager, you would add:

   ```
   Modules=… SimpleOutboundPackager
   ```

4. Depending on the module type (transport, packager, transform), locate the appropriate section in the file (identified by the commented text "# TRANSPORTS", "# PACKAGERS", "# TRANSFORM"), and add three lines defining the global properties for the module (these properties are described in *Global module properties*, next).

   For example, if your Java class names are com.<*your-company*>.SimplePackager and com.<*your-company*>.SimplePackagerMetadata, the #PACKERS section might be:

```
# PACKAGERS
#--------------------------
SimpleOutboundPackager.name=SimpleOutPackager
SimpleOutboundPackager.runtime.class.name=com.<your-company>.SimplePackager
SimpleOutboundPackager.metadata.class.name=com.<your-company>.SimplePackagerMetadata
```

## Global module properties

Each module must register these properties in CTModules.properties:

| Property Name | Description |
|---|---|
| *ModuleName*.name | Friendly name used in CTAssignments.xml to refer to this module. |
| *ModuleName*.runtime.class.name | The Java class name of module run-time implementation. |
| *ModuleName*.metadata.class.name | The Java class name of module metadata implementation. |

## Documenting your module

The Channel Toolkit installation includes a Microsoft Word document template that allows you to document your module for other users. The template is stored in:

<*Alliance*>\Partner\Partners*XXXX*\Properties\ChannelToolkit\CTModuleDocs\Channel_Toolkit_Module_Documentation_Template.dot

For cross-platform compatibility, we strongly suggest that you save the resuling Microsoft Word document in HTML format.

You should ensure that when your module is installed, the HTML document is installed in:

<*Alliance*>\Partners\Partner*XXXX*\Properties\ChannelToolkit\CTModuleDocs

APPENDIX

B

# Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

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

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

## Trademarks

# GLOSSARY

**action**—a task performed as part of a private process. A private process action is the equivalent of a step in a public process. See the following terms in this glossary for more information about the action types you can include in a private process:

- approval action
- extension action
- mapping action
- notification action
- output object action
- script action
- subprocess action
- termination action
- timer action

See also *private process*.

**adapter**—the software bridge between Partner Agreement Manager processes and specific end-system and business-application interfaces. Adapters manage interactions between business applications and the Adapter Server. They allow private processes to interact with external business applications while a process is running, and they allow PAM to start public processes based on events that occur in external business applications. See also *adapter implementation*, *adapter instance*, *adapter type*.

**adapter implementation**—the implementation declaration for an adapter type. It specifies the name and location of the Java source file that defines the application logic used to communicate with a specific end system through that end system's interface. The application logic is specified in the form of properties. See also *adapter*, *adapter instance*, *adapter type*.

**adapter instance**—an instance of an adapter implementation. The adapter instance is used in a private process extension action and provides the specific values to be used for the properties declared in the adapter implementation. See also *adapter*, *adapter implementation*, *adapter type, extension action*.

**adapter type**—a definition that is stored in XML format and specifies the adapter's properties as well as the operations and events it supports. A single adapter type can have multiple implementations, and each implementation can have multiple instances. See also *adapter*, *adapter implementation*, *adapter instance*.

**approval action**—a private process action that you use to ask for a response from a user before letting the process continue to run. You can use an approval action, for example, to ask for an OK when a purchase order exceeds a predetermined amount. See also *private process*.

**assignment**—information used by the Channel Toolkit framework to determine default and override properties.

**business object**—a message transmitted as part of a public process. Business objects take the form of purchase orders, acknowledgments, requests for clarification, and so on. See also *business object type*.

**business object type**—a definition that determines the types of information a message can contain. It has three properties: the top-level element in its element definition set, its key field, and whether instances of it return audit information for non-repudiation purposes. The name of the business object type is the name of the element you select as its top-level element. See also *business object*, *element definition set*, *non-repudiation*.

**business object variable**—one of the two types of variables used in Partner Agreement Manager to store information within a process. Business object variables create an instance of a business object type. They can be used to store, for example, the outputs from extension actions, the inputs for map actions, or the inputs and outputs for subprocesses. See also *business object*, *business object type*, *extension action*, *variant variable*.

**CA**—see *certificate authority*.

**certificate**—a security document that binds a public encryption key to an entity (an individual or organization) known as the principal. The security document (a digital certificate) is signed by another entity known as the issuer. A digital certificate for which both the principal and issuer are the same entity is known as a self-signed certificate. A certificate for which the principal and issuer are different entities is issued by a certificate authority (CA) like VeriSign and is known as a CA-issued (or third-party-signed) certificate. Partner Agreement Manager supports both self-signed and CA-issued certificates. PAM also supports the binding of certificates to be used for signature authentication, message encryption, and SSL authentication for channels other than Partner Agreement Manager. See also *certificate authority*, *SSL*.

**certificate authority**—a trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the certificate authority, or CA, is to authenticate the entities (individuals or organizations) involved in electronic transactions. CAs are a critical component in data security and electronic commerce because they guarantee that the two parties exchanging information are really who they claim to be. See also *certificate*.

**channel**—a communications mechanism that encapsulates all the processing information needed to send messages to a partner's system, as well as to translate data received from a partner into Partner Agreement Manager messages. PAM provides channels for RosettaNet, EDI, cXML, and other systems and protocols. See also *message*.

**data transformation module**—a Channel Toolkit module that transforms a single business object into an arbitrary stream of data, and vice-versa.

**digital certificate**—see *certificate*.

**DTD**—Document Type Definition. A type of file associated with SGML and XML documents that defines how the formatting tags should be interpreted by the application presenting the document. In Partner Agreement Manager, a DTD file contains the complete description of a business object type's element definition set. See also *business object*, *business object type*, *element definition set*.

**element definition set**—a collection of data fields (or elements) or groups of data fields that defines the structure and meaning of a business object type. See also *business object*, *business object type*.

**encryption certificate**—see *certificate*.

**event**—a piece of information that comes into Partner Agreement Manager as a message from another source (an enterprise system or business application, for example) and triggers a public process. See also *message*.

**event push**—a method that uses the HTTP POST mechanism to push events into Partner Agreement Manager as a way to trigger processes. A port on the Process Server is set to listen for events in the form of HTTP POST messages. When a message is detected, PAM uses the information in the message to generate an event. See also *event*.

**extended enterprise**—a business model under which companies that work together as partners function as efficiently as a single organization through the implementation of automated communication technologies.

**extension action**—a private process action that communicates via an adapter with an external application that is registered with Partner Agreement Manager. You can use an extension action, for example, to launch a spreadsheet application, perform calculations, and update the enterprise system, or to get information from an enterprise system or listen for an event in the enterprise system. See also *adapter*, *private process*.

**framework**—manages much of the complexity in Channel Toolkit channels, including the interaction between Partner Agreement Manager and the modules, channel configuration, error handling, retry, and so on.

**LDAP**—Lightweight Directory Access Protocol. LDAP provides a standard method for accessing information from a central directory. After user authentication is set up in the LDAP directory, applications that use the LDAP protocol can retrieve the information from that directory. An authenticated user can log in to any application that supports the LDAP protocol with the same user name and password.

**linked certificate**—see *certificate.*

**map**—a Java Script or VBScript that inserts data into fields in an output business object type generated by a private process. The map specifies which fields in the output business object type receive data, and it identifies the information source.

**map method**—a reusable logical block of code that inserts data into a particular type of element or element sequence in a business object type. Within a map method, you can write the expressions that map individual input and output fields in the sequence. Or you can create a submap and drag input fields to output fields and have Partner Agreement Manager create the appropriate mapping expressions. See also *map*, *submap*.

**mapping action**—a private process action that you use to call a map. The map specifies the fields in a business object type that will receive data extracted from another source. You use a mapping action when you want to extract data from one business object type and insert it in a different business object type. For example, you use a mapping action to transform a purchase order generated by your inventory system into a sales order in a format that your partner expects. See also *map*, *private process*.

**message**—a structured communication used to pass information and control to another partner in a public process. The action in the process passes to the partner who receives the message. The content of a message is determined by its business object type. A message can be transmitted via synchronous or asynchronous methods, as determined by its communication service type. See *business object type*.

**module**—plugs into the Channel Toolkit framework to provide specific services required by the channel. See also *data transformation module*, *packaging module*, *unpackaging module*, and *transport module*.

**non-repudiation**—a business object security feature that authenticates instances of a business object type and maintains an audit record to verify that they were received by the intended recipient. For business object instances that you receive, Partner Agreement Manager authenticates each instance and maintains an audit record to verify that the instance actually originated with the stated originator. If you disable auditing for a business object type, non-repudiation support is disabled for all messages that contain instances of that business object type.

**notification action**—a private process action that you use to send an e-mail, fax, or pager message to addressees that you specify. You use a notification action to inform someone inside or outside your organization that an event has occurred. For example, you can use a notification action to alert the order entry department when a purchase order arrives from a customer. See also *private process*.

**output object action**—a private process action that you use to bind a business object to the expected output object and path in a public process. You use an output object action at the point in a private process when you are ready to send a business object to the associated public process. This is typically the last action in the private process. See also *private process*.

**packaging module**—a Channel Toolkit module that packages one or more business object streams into a stream that can be sent by the transport module. Examples of packaging include enveloping (such as MIME encoding), encryption, grouping single business objects together, and so on.

In general, the term "packaging module" is used to mean both packaging and unpackaging modules.

**partner group**—a group of partners that perform the same role in a process at different times. Instead of duplicating a public process and substituting a different partner name, you can set up a partner group for the public process and then designate a specific partner as the participant when you start an instance of the process. For example, you might design a generic purchasing process that works equally well with any of your suppliers and then designate the appropriate partner when you start the process.

**partner profile**—information that identifies an organization, specifies a contact person in that organization, lists the communication services the organization supports, and defines the organization's security profile. When partners agree to participate in a public process, they must exchange profile information as a way to ensure authenticity before they can proceed.

**PIP**—Partner Interface Process. RosettaNet PIPs are specialized system-to-system XML-based dialogs that define business processes between supply-chain partners and provide models and documents for the implementation of e-commerce standards. 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. See also *RosettaNet*.

**post method**—the last block of code that is executed when a mapping action runs. Its only parameter is the output business object. You use the post method when you need to perform post-processing on the output business object. For example, you might use the post method to set the value of a summary field based on the number of line items in the output business object, or to examine a range of dates in a repeated group, extract the most recent date, and post that date in a header field. See also *mapping action*, *pre method*.

**pre method**—the first block of code that is executed when a mapping action runs. The pre method's parameters are the map inputs. You use the pre method to access a map's inputs and set global variables based on their content. See also *mapping action*, *post method*.

**private process**—a task or set of tasks that business partners participating in a public process perform at points where they need to take action internally. Partners participating in a public process must implement a private process for each public process step that they own. A private process begins with input from the public process and ends with output that feeds back into the public process. The input can be the receipt of a business object from a partner, or it can be a triggering event from an internal system. The output is the business object that transfers control back to the public process. See also *action*, *process*, *public process*.

**private process action**—see *action*.

**process**—the flow of actions and the exchange of business information between partners in an extended enterprise. A process operates on two levels, public and private. See *extended enterprise*, *private process*, *public process*.

**public process**—the step-by-step flow of messages, events, and actions between two or more business partners. Public processes are set up by agreement between partners, and each step in a public process has a private process associated with it. A public process is developed by one partner, and all the partners who participate in it must review and approve it before it can be implemented. The partner who designs a public process is its owner. See also *private process*, *process*.

**RosettaNet**—a consortium of major information technology, electronic components, and semiconductor manufacturing companies that is working to create and implement industry-wide, open e-business process standards. See also *PIP*.

**script action**—a private process action that consists of a script written in VBScript or JavaScript and is designed to manipulate information or set up conditional actions based on input. You use a script to establish decision-making criteria for branches or loops, to set variables, or to calculate values that are used elsewhere in the private process. See also *private process*.

**security certificate**—see *certificate*.

**self-signed certificate**—see *certificate*.

**signature certificate**—see *certificate*.

**SSL**—Secure Sockets Layer. The SSL protocol is a security protocol that provides for communications privacy and reliability over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

**submap**—a secondary level map that is called by a map method to insert data into an output element other than the top-level element. See *map*, *map method*.

**subprocess action**—a private process action you use to call an existing public process. You can call any public process in which your organization owns the first partner action. For example, you can use a subprocess to get a quote approved by a third-party supplier before responding to a customer. See also *private process*.

**termination action**—a private process action that you use to stop a process at a predetermined point for a reason that you specify. You can use a termination action to deal with errors in data that might prevent a process from completing successfully. For example, you might want to stop a process in cases where an enterprise system passes incomplete or corrupted information to it. See also *private process*.

**third-party-signed certificate**—another name for a CA-issued certificate. See *certificate*.

**timer action**—a private process action that you use to insert a pause. You can use a timer action to specify the period of time you want to elapse before the next action in the process starts. See also *private process*.

**transport module**—a Channel Toolkit module that takes a stream and sends it out over a transport medium (such as HTTP), or receives a stream from a transport medium and passes it to the framework.

**unpackaging module**—a Channel Toolkit module that unpackages a stream into one or more business object streams. Examples of unpackaging include recognizing business object types, de-enveloping, decryption, breaking a multi-business object message into individual business objects, and so on.

In general the term “packaging module” is used to mean both packaging and unpackaging modules. See also *packaging module*.

**variant variable**—single field variables. Variant variables store text strings—the type of information contained in a single field element. You can use variant variables to store the input for actions, to set flags (such as the time-out flag for an approval action), to move information within scripts, or to store the results of an approval action. See also *business object variable*.

# INDEX