SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Commerce One MarketSite User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050405222138.

Contents

Chapter 3

Configuration 24

Chapter 4

Implementation 35

Chapter 1

Introduction

This document describes how to install, configure, and implement the e*Way Intelligent Adapter for Commerce One™ MarketSite™ (Commerce One MarketSite e*Way) using the XML Portal Connector (XPC) framework.

1.1 Overview

The Commerce One MarketSite e*Way provides a method of exchanging data across an enterprise that incorporates the Commerce One MarketSite application and a variety of other applications. The e*Way provides both buy-side and sell-side solutions and utilizes the Commerce One Portal Connector (XPC). By leveraging MarketSite's automated procurement cycle and XML technology, the e*Way transfers information between MarketSite and e*Gate Integrator, enabling information to be disseminated throughout the enterprise.

Synchronous and Asynchronous document submission is performed by the e*Way using the Commerce One Application Programming Interface (API) referred to as the Transmitter API, as well as the XPC server.

1.1.1. Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have moderate to advanced-level knowledge of Windows operations and administration; and to be thoroughly familiar with the Commerce One MarketSite XPC application framework, and Windows-style GUI operations.

1.1.2. Components

The following components comprise the Commerce One MarketSite e*Way:

- Configuration files, which the e*Way Editor uses to define configuration parameters

A complete list of installed files appears in **Table 1 on page 20**.

1.2 Operational Overview

1.2.1. Basic Operations

Commerce One provides buyers and suppliers with the ability to come together in the electronic marketplace. Commerce One's solutions are based on a suite of products, services and standards that provide organizations with the vehicle to extend existing internal systems real-time access to the Internet trading communities.

The Commerce One MarketSite e*Way supports both MarketSite 3.2 and MarketSite 4.0. Both versions of MarketSite implement the XML Common Business Library (xCBL) messaging standard. xCBL has been specifically designed for e-commerce. (For more information on xCBL, see **http://www.xcbl.org/**.)

Figure 1 on page 10 portrays the relationship between the Commerce One MarketSite e*Way and MarketSite.

The Commerce One MarketSite e*Way provides the ability to exchange documents between a trading partner using e*Gate and the CommerceOne XPC platform with another trading partner registered with MarketSite. The trading partner may be acting as either a buyer sending documents, such as orders, to a supplier registered on MarketSite, or a supplier processing orders or other requests, and sending back order responses to the buyer via MarketSite. These documents are exchanged in xCBL format.

The e*Way utilizes two different interfaces provided by CommerceOne for XPC users communicating with MarketSite:

- Transmitter Application Programming Interface (API): allows users to create applications that send documents directly to MarketSite (asynchronous) or send and receive documents (synchronous) without the XPC server.
- XPC Server: provides an extendable environment for processing documents exchanged with MarketSite. The XPC server may be used with pre-configured services or with services configured with the user's custom XPC components. The XPC server's main interface for sending and receiving documents is the file system. Document files are polled from inbound and outbound directories. It has two types of configurable services; document services and timed services. *Document services* handle incoming documents from MarketSite, while *Timed services* handle sending documents to MarketSite.

Transmitting Documents Using the Commerce One e*Way Transmitter ETD

An e*Way Connection and the associated e*Way interface ETD may be used in a Java™ Collaboration to send documents synchronously or asynchronously directly to MarketSite via the Transmitter API. The Transmitter API also supports dynamic transmission to different suppliers. The XPC server interface supports asynchronous exchange only. Response to documents sent synchronously may be obtained via this Transmitter ETD. Responses for documents sent asynchronously must be received via the XPC server.

Note: *Among the advantages of using the Transmitter API is the ability to send documents that must be sent synchronously to MarketSite. It also allows you to send documents to multiple suppliers.*

The Transmitter ETD component is primarily used by buyers for sending orders asynchronously, or price checks and availability checks sent synchronously.

Receiving Documents Using XPC Server

Documents may be received from MarketSite via XPC Document Services. The Commerce One e*Way utilizes the pre-configured Trading Partner Configuration. When installing XPC, you must run the Configure GUI and select the Pre-configure Trading Partner Configuration button to ensure that the pre-configured services are installed. The pre-configured services provide a simple interface for obtaining and sending documents to MarketSite through inbound and outbound directories.

The Batch e*Way is used to exchange these documents in e*Gate. Collaborations in e*Gate, are used to process documents and generate the appropriate responses and document error-handling.

The XPC server may be used by a buyer to receive response documents or request documents sent by the supplier. It may also be used by a supplier receiving request documents from a buyer.

Sending Documents Using XPC Server

Documents may be generated in an e*Gate Java Collaboration and sent to MarketSite via the Batch e*Way to "drop" them into the configured XPC server outbound directory. The XPC server must simply have the appropriate Timed service for the documents enabled. As files appear in the outbound directory, the XPC server picks them up and sends them to MarketSite. The supplier information for these outbound documents are specified in a text file located in the XPC root directory ($XPCROOTDIR/tpid_map/map.txt).

This component may be used by a buyer to send request documents to a supplier. It may also be used by a supplier to send request documents to a buyer or response documents for request documents received from the buyer.

Help in Java Collaborations Through the xCBL ETD Library and Commerce One XPC Helper ETD

Java Collaboration used for processing xCBL documents must use the xCBL ETD library, which is installed as a separate add-on component. This library was generated from DTDs obtained from www.xcbl.org.

Note: *Commerce One expects the two lines prepended to these documents, specifying that they are SOX based documents. For example, for xCBL version 3:*

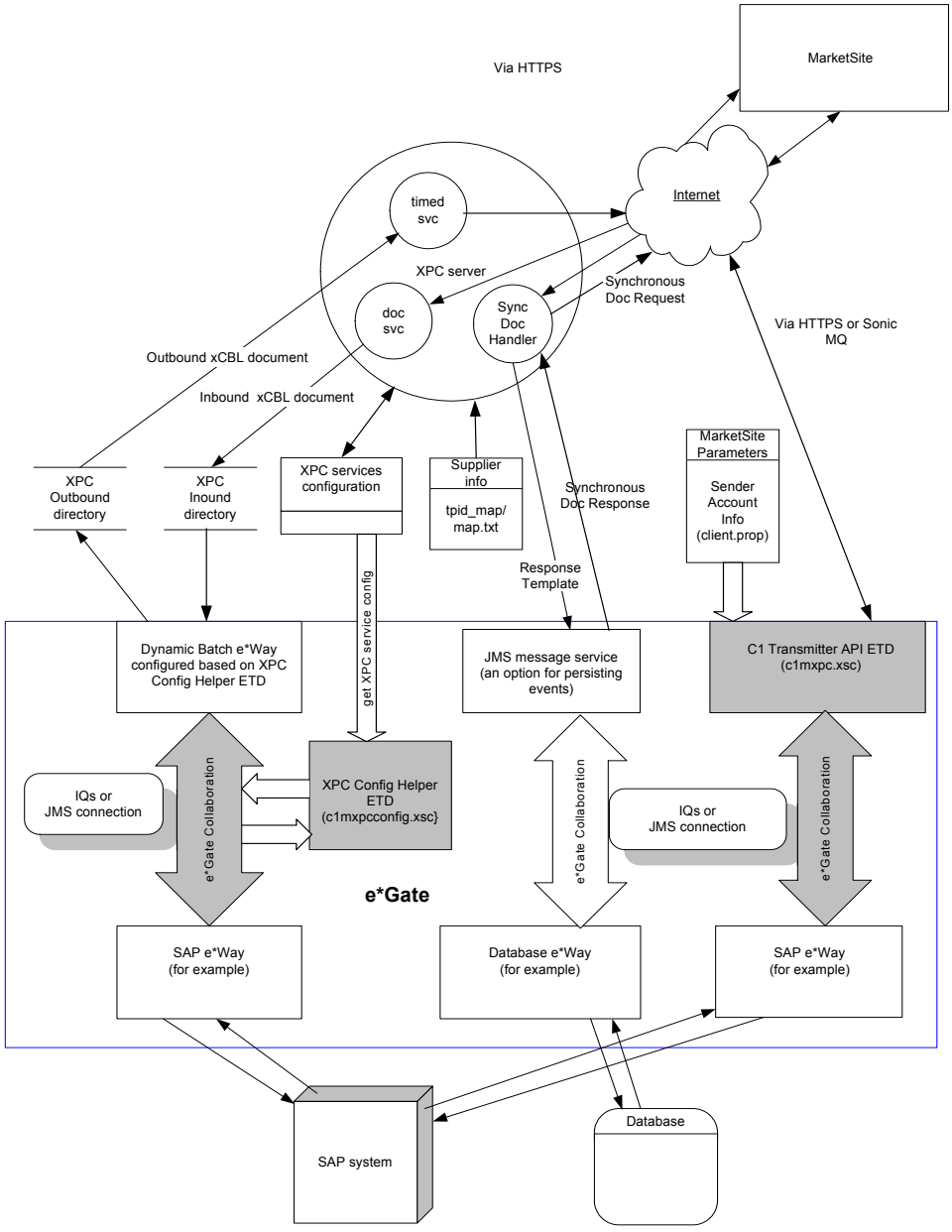
```
<?soxtype urn:x-
commerceone:document:com:commerone:XCBL30:XCBL30.sox$1.0?>
```

```
<?import urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30:sox$1.0?>
```

For Collaborations involving the XPC services, an ETD that can be loaded in the e*Gate Java Collaboration Editor called "c1mxpcconfig" is provided to help the user determine where inbound files must be obtained by the Batch e*Way, where outbound files must be stored by the Batch e*Way, what file name prefix is associated with specific document types, where the supplier map file is located. The class associated with this ETD is **com.stc.eways.c1mxpc.C1MXPCConfigHelper**. The associated XSC file for this interface ETD is c1mxpcconfig.xsc

The following diagram provides a more detailed view of the Commerce One MarketSite e*Way components.

Figure 1 Commerce One MarketSite Information Flow



1.2.2. Considerations

The classes are located in **stcc1mxpc.jar** installed in ..\eGate\client\classes and .. eGate\Server\registry\...\classes.

Use a Multimode e*Way (**stceway.exe**) to create the Collaboration Rule for data mapping and for sending documents to MarketSite.

xCBL documents sent to MarketSite must contain SOX headers.

1.2.3. Authentication with MarketSite and Security

The MarketSite Administrator defines the exact requirements for correctly configuring Authentication and Security.

Transmitter API

Authentication and security is based on configuration parameters specified by the user (usually in the client.prop file). The e*Way must be informed where this file is located to enable the successful retrieval of information, when communicating with MarketSite. For more information see Chapter 3, Configuring the Commerce One MarketSite e*Way, **client.prop File Path** on page 33

Note: *CommerceOne stores the encrypted password in the configuration file. The passwords are then decrypted by the appropriate CommerceOne API call.*

XPC Server

Authentication and security is based on configuration information specified in the Configure GUI. This information is stored in an XPC configuration file also. Since communication with MarketSite performed by the XPC server, no authentication configuration information needs to be specified on the e*Gate side. The XPC Security Manager is used.

Note: *XPC may use username/password authentication or certificates. If communicating with MarketSite 3.2, HTTP SSL is used with client authentication. You are then acting as a client to the MarketSite HTTP server. In MarketSite 4.0, support is provided for HTTPs or Sonic MQ for transport. Sonic MQ uses username/password for authentication.*

1.3 Supported Operating Systems

The Commerce One MarketSite e*Way is available on the following operating systems:

- Window 2000 and Windows Server 2003

Note: *Open and review the **Readme.txt** for the* Commerce One MarketSite e*Way *regarding any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM at setup\addons\ewc1.*

1.4 System Requirements

To use the Commerce One MarketSite e*Way, you need the following:

- 170 MB of disk space is required for installation. Additional disk space may be required for e*Way executable, configuration, library, and script files. This disk space is required on both the Participating and the Registry Host. Additional disk space is required to process and queue the data that this e*Way processes; the amount necessary varies based on the type and size of the data being processed, and any external applications performing the processing. XPC also requires additional disk space for document processing and envelope/message storage. 2 GB of available disk space is recommended.

Note: *The disk space noted above is dependent upon the number of processors used and the expected load (message size and frequency). See the* ***XML Portal Connector (XPC) Installation and Administration Guide*** *for* ***Windows*** *for more information on hardware and disk space requirements.*

- An e*Gate Participating Host.
- The Batch e*Way (installed with this e*Way).
- xCBL ETD Library 3.0 (installed separately).
- Open and review the Readme.txt for the Commerce One MarketSite e*Way for any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM at setup\addons\ewc1.
- The e*Gate API Kit (JMS).

1.5 External System Requirements

The Commerce One MarketSite e*Way requires the following external applications:

- XML Portal Connector (XPC) 4.0 and 4.1. This is included in the e*Way Intelligent Adapter for Commerce One MarketSite installation.
- Commerce One MarketSite XPC server.
- Java 2 Runtime Environment or JRE v 1.2.2. This is required for the installation of XML Portal Connector (XPC) 4.0.
- Java 2 Runtime Environment or JRE 1.3.0_002 with HotSpot is required for the installation of Sonic MQ Broker.
- Java 2 Runtime Environment or JRE v 1.3.1. This is required for the installation of XML Portal Connector (XPC) 4.1.

1.6 SeeBeyond Web Site

The SeeBeyond Web site is your best source for up-to-the-minute product news and technical support information. The site's URL is

http://www.SeeBeyond.com

Chapter 2

Installation

This chapter covers how to install the Commerce One MarketSite e*Way. It also includes a list of the files and directories the installation process creates.

2.0.1. Installing XPC

Documents are received from MarketSite via XPC Document Services. The Commerce One MarketSite e*Way utilizes the pre-configured Trading Partner Configuration. When installing XPC, you must run the Configure GUI and select the Pre-configure Trading Partner Configuration button to ensure that the pre-configured services are installed. Pre-configured services provide a simple interface for obtaining and sending documents to MarketSite through inbound and outbound directories.

Documents can be generated in an e*Gate Java Collaboration and sent to MarketSite, via the Batch e*Way, and "dropped" into the configured XPC server outbound directory. The XPC Server must simply have the appropriate Timed service for the documents enabled. As files appear in the outbound directory, the XPC server picks them up and sends them to MarketSite. Supplier information for these outbound documents is specified in a text file located in the XPC root directory ($XPCROOTDIR/tpid_map/map.txt).

XPC 4.0 and XPC 4.1 are installed during the Commerce One MarketSite e*Way installation for Windows. For Windows 2000 (XPC 4.1 must be installed over an existing XPC 4.0 installation).

2.1 Windows 2000

2.1.1. Pre-installation

1. Open and review the Readme.txt for the Commerce One MarketSite e*Way for any additional requirements prior to the installation. The Readme.txt is located on the Installation CD_ROM at setup\addons\ewc1.
2. XPC 4.0, 4.1 and MQ Broker require JRE 1.2.2, 1.3.0, and 1.3.1 installed on the system prior to prior to installation. JRE 1.2.2, 1.3.0 and 1.3.1 are available on the installation CD ROM in the ..\setup\addons\ewc1\jre\win32 or \sparc26 directories.

3 Exit all Windows programs before running the setup program, including any anti-virus applications.

4 It is recommended that write caching for arrayed drives where SonicMQ Broker is to be installed be turned off. Write caching increases the possibility for messages to be lost when a broker machine fails. For more information on SonicMQ Broker and write cashing see the *XPC Installation and Administration Guide*.

5 Administrator privileges are required to install this e*Way.

2.1.2. Installation Procedure

To install Commerce One MarketSite e*Way and XPC 4.1 on a Windows 2000 system

1 Log in as an Administrator to the workstation on which the e*Way is to be installed.

2 Insert the installation CD-ROM into the CD-ROM drive.

3 If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

4 The InstallShield setup application launches. Follow the on-screen instructions to begin installation of the e*Way.

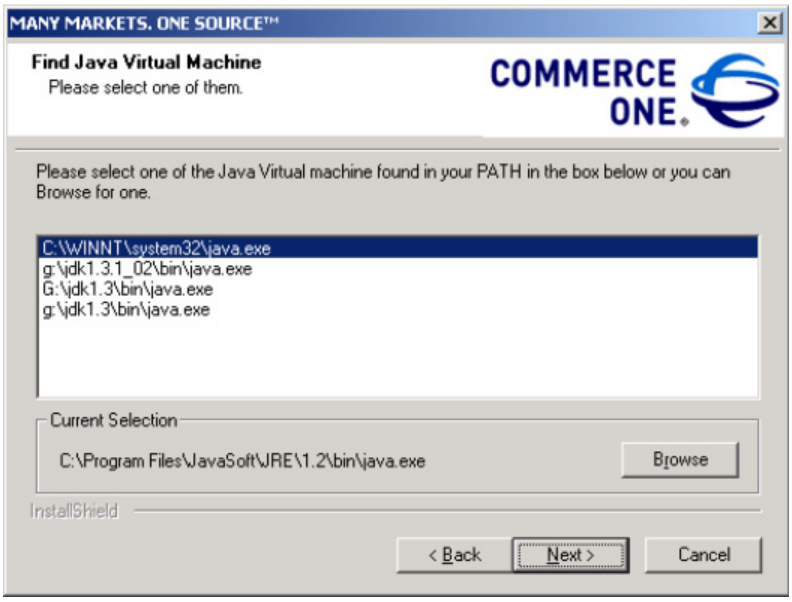
Note: *Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

5 Prior to installing XPC 4.0, installation searches for an installed version of Java Virtual Machine JRE 1.2.2. If it is not found, it is automatically installed at this point.

2.1.3. XPC 4.0 and 4.1 Installation

6 Install XPC 4.0. When asked, during installation, to select the required version of the Java Virtual Machine (see Figure 2) it is essential that even though the correct version may be listed in the selection window, the user must click on the Browse button and specify the absolute path of the correct JRE (JVM) java.exe. XPC 4.0 requires JRE 1.2.2 (See Figure 2).

Figure 2 XPC Installation, Select JRE (JVM)



7 Enter the XPC Server name. This can be left as **defaultserver**.

8 Select the install destination folder. The default can be used if appropriate.

9 At this point XPC installation is ready to start copying program files. Sonic MQ is also installed at this time. Click Next to proceed.

10 When the install wizard is finished installing XML Portal Connector 4.0, the user is prompted to restart the computer. Select **No**, and click **Finish** to proceed with the installation.

11 The XPC 4.1 installation wizard begins. Continue with the installation. When prompted for a password enter **Admin**.

12 Prior to installing XPC 4.1, installation searches for an installed version of (JVM) JRE 1.3.1. If JRE 1.3.1. is not found, it is automatically installed at this point.

13 As in step 1, When asked during installation to select the required version of the Java Virtual Machine it is essential that even though the correct version may be listed in the selection window, the user must click on the Browse button and specify the absolute path of the correct JRE (JVM) java.exe. XPC 4.1 requires JRE 1.3.1.

14 Again, when the installation wizard has finished installing XML Portal Connector 4.1 the user is prompted to restart the computer. Select **No**, and click **Finish** to proceed with the Commerce One MarketSite e*Way installation.

15 When the e*Way installation is complete, reboot the computer. The computer must be restarted before XPC can be configured.

Note: *The e*Way Configuration parameters are discussed in* **Chapter 3**. *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.*

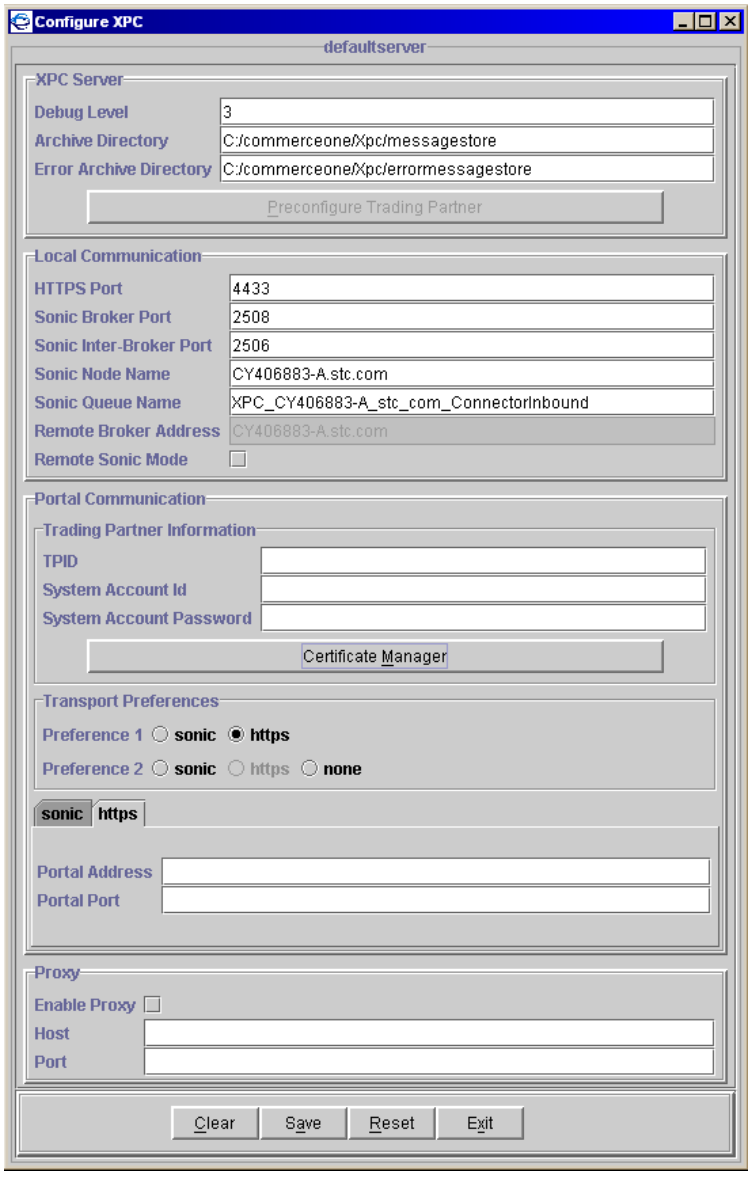
*For more information about configuring e*Ways or how to use the e*Way Editor, see the **e*Gate Integrator User's Guide**.*

2.1.4. Configuring XML Portal Connector 4.1

1. You must use **Start-> XML Portal Connector -> Configure** to specify the MarketSite configuration. This may be obtained during MarketSite registration. For MarketSite 3.2, you must specify the correct certificate and use HTTPS (see Figure 3)
2. Run **Start->XML Portal Connect-> Invoker**. Using Invoker, perform a ping test. A successful ping test receives a **got pong!** response.

Note: *Make sure the XPC Server is started before you begin these steps. If XPC Server is not running, the Configure XPC tool will not work.*

Figure 3 .Configure XPC Dialog Box



3 When the Configure XPC Dialog Box is launched, click on **Preconfigure Trading Partner**. Stop the XPC server, then restart it and wait for about 5 minutes until all of the Java process are done. This preconfigures the services that can be viewed using the XPC Manager

4 The Supplier information is specified in the file %xpcroot%\tpid_map\map.txt for the XPC server. For the transmitter ETD component of the e*Way, use the setSupplier method. This allows you to set multiple suppliers.

2.1.5. Configuring XPC Manager

The XPC configuration can be performed by accessing the XPC Manager via the browser, located on the XPC machine.

Loading XPC Manager

Before running the XPC Manager, first load XPC's self-signed certificate into the browser. This certificate is automatically created when you install XPC. To load the self-signed certificate on Internet Explorer 5, do the following:

1 Select **Tools**, **Internet Options**, **Content Tab**, **Certificates**, **Import**.

2 Select **<%XPCROOT%>/bin/client.p12**.

Load XPC Manager as follows:

1 Start the XPC server.

2 Browse to the following URL:

```
https://localhost:4433/servlet/XPCManager (case sensitive)
```

Note: *If Configure XPC is used to change the port that the XPC server is listening on from the default of 4433, the URL also needs to be suitably modified. If you cannot load XPC Manager, restart XPC from the command line and check the logs.*

The following services must be enabled:

- XPCAvailability CheckRequest30Inbound
- XPCOrder30Inbound
- XPCOrderResponseFromOrder30Outbound
- XPCOrderStatusRequest30Inbound
- XPCPriceCheckRequest30Inbound
- XPCOrder30Outbound
- XPCOrderResponse30Inbound

To support AdvancedShipNotice, Invoice, and Change Order XCBL docs, and other doc types, the following services should also be enabled (depending on the role of buyer or supplier):

- XPCAdvanceShipmentNotice30Inbound

- XPCAdvanceShipmentNotice30Outbound
- XPCChangeOrder30Inbound
- XPCChangeOrder30Outbound
- XPCInvoice30Inbound
- XPCInvoice30Outbound
- XPCOrderResponseFromChangeOrder30Outbound

More services can be enabled if necessary. The above list contains the minimum set recommended by the XPC 4.1 FAQ

Note: *Before updating your services in the XPC Manager, it is recommended to backup the following directory:*

<rootdir>:\commerceone\Xpc\runtime\servers\defaultserver\config\service

2.1.6. Configuring the Synchronous Document Support Samples for Commerce One XPC.

To configure the Document Support Samples for the sample schemas do the following:

1. Extract supplierxpcsync_MyIntegrator_Java.zip from the ewc1 sample directory to a temporary directory.
2. Back up all existing files in the following directories:

```
<%XPCROOT%>\sample\com\commerceone\sample\xpc\my_integrators
<%XPCROOT%>\lib\com\commerceone\xpc\my_integrators
<%XPCROOT%>\sample\classes\com\commerceone\xpc\my_integrators
<%XPCROOT%>\etc\classpath
```

3. Copy the following precompiled class files extracted from the .zip file:

```
myAvailabilityCheckIntegrator30.class
myOrderStatusIntegrator30.class
myPriceCheckIntegrator30.class
```

   copying these files to the following location:

```
<%XPCROOT%>\lib\com\commerceone\xpc\my_integrators
```

4. Copy the e*Gate property file **egateservice.properties** that is included in the supplierxpcsync_MyIntegrator_Java.zip file to the following location:

```
<%XPCROOT%>\runtime\servers\defaultserver\config\egateservice.
properties
```

   (For further information refer to the Readme.txt, step B, included with supplierxpcsync_MyIntegrator_Java.zip.)

5. Append the following two lines:

```
<%XPCROOT%>:/eGate/client/classes/stcc1mxpc.jar
<%XPCROOT%>:/eGate/client/classes/stcjms.jar
```

   to the following file on the XPC machine:

```
<%XPCROOT%>\etc\classpath\default
```

Note: *If e*Gate is installed on the machine then simply append the lines to the above file. If e*Gate is not installed on this machine, copy these from the e*Gate machine and add them to the respective path, then append the files to the above file. (For further information refer to the Readme.txt, step D, included with supplierxpcsync_MyIntegrator_Java.zip.)*

6 Restart XPC after completing the above in order for the changes to take place.

7 This is sufficient for the purposes of the sample schemas. Developers interested in further configuration of the samples may consult the Readme.txt file included in the supplierxpcsync_MyIntegrator_Java.zip file.

2.2 Files/Directories Created by the Installation

The Commerce One MarketSite e*Way installation process installs the following files within the e*Gate directory tree. Files are installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files created by the Participating Host installation

e*Gate Directory	File(s)
configs\c1mxpc	stcc1mxpc.def stcc1mxpcconfig.def
configs\stcewgenericjava\	stecewc1.def
classes\	stcc1mxpc.jar
etd\	c1mxpc.ctl c1mxpcconfig.ctl
etd\c1mxpc\	c1mxpc.xsc c1mxpcconfig.xsc
ThirdParty\jdom\jdom-b6\	ant.jar collections.jar jdom-jdk11.jar jdom.jar xerces.jar

Table 1 Files created by the Participating Host installation

e*Gate Directory	File(s)
ThirdParty\xpc41	activation.jar broker.jar bussdocs.jar ccs_all.jar ccs_dir.jar ccs_event.jar ccs_install.jar ccs_server.jar ccs_util.jar ccs_xdk.jar ccs_xdkdir.jar client.jar enhydra.jar fscontext.jar hotFS.jar iaik.jar jdbc2_0-stdext.jar jigsawlite.jar jmail.jar jms.jar jndi.jar jsdk.jar ldap.jar mail.jar mspconfig.class Opta2000.zip Oracle.zip providerutil.jar sax.jar servlet.jar siswrapper.jar stcc1mxpc.jar swingall.jar vgateway.jar xcbl30.jar xmlc.jar xpc.jar

Table 2 Files Created in Conjunction with the Batch e*Way

e*Gate Directory	File(s)
bin\	stcewgenericmonk.exe stc_ewftp.dll stc_monkfilesys.dll
configs\stcewgenericmonk\	batch.def

Table 2 Files Created in Conjunction with the Batch e*Way

e*Gate Directory	File(s)
monk_library\batch\	batch-dynamic-init-c1.monk batch-dynamic-proc-out-c1.monk batch-dynamic-send-to-egate-c1.monk batch-exchange-data-c1.monk
eGate\client\monk_scripts\common	batch_eway_data.jar batch_eway_error.jar batch_eway_order.jar batch_eway_data.xsc batch_eway_error.xsc batch_eway_order.xsc
\eGate\client\etd\batchclient\	FtpFileETD.xsc
	batch-ack.monk batch-dynamic-init.monk batch-dynamic-proc-out.monk batch-dynamic-send-to-egate.monk batch-exchange-data.monk batch-exchange-utils.monk batch-ext-connect.monk batch-ext-shutdown.monk batch-ext-verify.monk batch-fetch-files-from-remote.monk batch-fetch-named-files.monk batch-init.monk batch-nak.monk batch-persist.monk batch-post-transfer.monk batch-proc-out.monk batch-regular-init.monk batch-regular-proc-out batch-send-path-file.monk batch-shutdown-notify.monk batch-startup.monk batch-utils.monk batch-validate-params.monk file-ext-connect.monk file-ext-shutdown.monk file-ext-verify.monk file-fetch.monk file-fetch-path.monk file-init.monk file-remote-path-list.monk

Table 2 Files Created in Conjunction with the Batch e*Way

e*Gate Directory	File(s)
	file-remote-post-transfer.monk file-rmt-list.monk file-rmt-post-transfer.monk file-send.monk file-send-path-file.monk file-startup.monk file-vaildate-params.monk ftp-connect.monk ftp-disconnect.monk ftp-ext-connect.monk ftp-ext-shutdown.monk ftp-ext-verify.monk ftp-fetch.monk ftp-fetch-path.monk ftp-init.monk ftp-pre-post-commands.monk ftp-remote-path-list.monk ftp-remote-post-transfer.monk ftp-rmt-list.monk ftp-rmt-post-transfer.monk ftp-send.monk ftp-send-path-file.monk ftp-startup.monk ftp-validate-params.monk local-post-transfer.monk

2.2.1. Post Installation

After installing the Commerce One MarketSite e*Way, run the following command:

```
java com.stc.eways.clmxpc.InstallJCSRC
```

to update the .jcsrc file, which affects the behavior of the xCBL ETD and XML Builder.

stcc1mxpc.jar (the e*Way's jar file) must be in your classpath.

Chapter 3

Configuration

This chapter describes how to configure the Multi-Mode e*Way and the Commerce One MarketSite e*Way Connection.

3.1 Multi-Mode e*Way Configuration

A Multi-Mode e*Way is a multi-threaded component used to route and transform data within e*Gate. Unlike traditional e*Ways, Multi-Mode e*Ways can use multiple simultaneous e*Way Connections to communicate with several external systems, as well as IQs or JMS IQ Managers. Multi-Mode e*Way properties are set using the Schema Designer.

To create and configure a New Multi-Mode e*Way:

1. Select the Navigator's Components tab.
2. Open the host and control broker on which you want to create the e*Way.
3. On the Palette, click on the **Create a New e*Way** button.
4. The New e*Way Component window opens. Enter the name of the new e*Way, then click **OK**.
5. Right-click the new e*Way and select **Properties** edit its properties.
6. When the e*Way Properties window opens, click on the **Find** button beneath the **Executable File** field, and select an executable file. For the purposes of the sample select **stceway.exe** (**stceway.exe** is located in the "bin\" directory).
7. Under the **Configuration File** field, click on the **New** button. When the Settings page opens, set the configuration parameters for this configuration file.
8. After selecting the desired parameters, save the current configuration. Close the **.cfg** file and select **OK** to close the e*Way Properties Window.

For more information on using the Configuration Editor, see the Configuration Editor's online Help or see the *e*Gate Integrator User's Guide*.

Multi-Mode e*Way Configuration Parameters

3.1.1. JVM Settings

The JVM Settings control basic Java Virtual Machine settings.

JNI DLL Absolute Pathname

Description

Specifies the absolute pathname to where the JNI DLL installed by the *Java 2 SDK* 1.3.1_02 is located on the Participating Host.

Required Values

A valid pathname.

Additional Information

The JNI dll name varies on different O/S platforms:

OS	Java 2 JNI DLL Name
Windows 2000	jvm.dll
Solaris	libjvm.so

The value assigned may contain a reference to an environment variable. To do this, enclose the variable name within a pair of % symbols. For example:

```
%MY_JNIDLL%
```

Such variables are used when multiple Participating Hosts are used on different platforms.

Note: *To ensure that the JNI DLL loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory that contain shared libraries (UNIX) or DLLs.*

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

If left unset, no paths are prepended to the CLASSPATH environment variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the JVM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) is set.

Note: *All necessary JAR and ZIP files needed by both e*Gate and the JVM must be included. It is advised that the* ***CLASSPATH Prepend*** *parameter be used.*

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies whether the path is appended for the CLASSPATH environmental variable to jar and zip files needed by the JVM.

Required Values

YES or **NO**. The configured default is YES.

Initial Heap Size

Description

Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for Native Threads

Description

Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler is disabled.

Required Values

YES or **NO**.

Note: *This parameter is not supported for Java Release 1.*

Remote Debugging port number

Description

Specifies the port number by which the e*Gate Java Debugger can connect with the JVM to allow remote debugging.

Required Values

An unused port number in the range 2000 through 65535. If not specified, the e*Gate Java Debugger is not able to connect to this e*Way.

Suspend option for debugging

Description

Allows you to specify that the e*Way should do no processing until an e*Gate Java Debugger has successfully connected to it.

Required Values

YES or **No**. YES suspends e*Way processing until a Debugger connects to it. NO enables e*Way processing immediately upon startup.

Auxiliary JVM Configuration File

Description

Specifies an auxiliary JVM configuration file for additional parameters.

Required Values

The location of the auxiliary JVM configuration file.

3.2 e*Way Connection Configuration Parameters

e*Way configuration parameters are set using the Configuration Editor.

To change e*Way Connection configuration parameters

1. In the Schema Designer's Component editor, select the e*Way Connection you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of

the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

For more information about how to use the Configuration Editor, see the Configuration Editor's online Help or see the *e*Gate Integrator User's Guide*.

3.3 e*Way Connection for XPC Server Based Modules

When creating an XPC e*Way connection, the **c1mxpcconfig.def** file is used. The following parameters are used to configure the e*Way Connection's configuration parameters necessary to facilitate communication with the XPC Server and are organized into the following sections:

- **Connector** on page 31
- **XPC Settings** on page 32

3.3.1. Connector

The Connector Settings control basic operational parameters.

Type

Description

Specifies the type of connection.

Required Values

c1mxpcconfig is the default value for Commerce One MarketSite XPC configuration connection.

Class

Description

Specifies the class name of the Commerce One MarketSite XPC connector object.

Required Values

com.stc.eways.c1mxpc.C1MXPCConnector is the default value.

Property Tag

Description

Specifies data source identity value required by EBobConnectorFactory.

Required Values

A string.

3.3.2. XPC Config Settings

The XPC Config Settings parameters contain the information needed to access XPC.

XPC Config Root

Description

Specifies the root directory where XPC is accessible locally (file system) to e*Gate. For example, the network drive letter for a Windows 2000 system or mount point for NFS share. This XPC Root directory is prepended to the XPC "default property file path" to obtain the full path for the XPC default property file associated with the underlying inbound or outbound services. The path is normally modified/configured via the XPC user interface

Required Values

The valid XPC root directory.

Default Property File Path

Description

Specifies the location of the default property file which contains information about inbound and outbound services. This file path is concatenated with the "XPC Root directory" to obtain the full path for the XPC default property file.

Required Values

A string.

Default Property File Name

Description

Specifies the name of your default property file, which contains information about inbound and outbound services. The "XPC Root directory" is concatenated with the "XPC default property file path" to obtain the full path for this XPC default property file. Always ensue a current copy of this property file is available to e*Gate. After a configuration change using the XPC interface tool, this file must be copied to the e*Gate system.

Required Values

A string.

3.3.3. Additional XCBL Processing

The parameters in this section are used to specify additional XCBL processing information.

Soxtype Namespace Processing Instruction

Description

An optional processing instruction with a soxtype target, indicating the complete sox namespace, (CBL or XCBL30) with version information ($1.0). It may be used by the Java Collaboration Editor to prepend to the underlying xCBL data. This can also be accomplished by hardcoding this PI string to prepend to the XCBL data in the Java Collaboration Service.

Required Values

A string. For example,

```
"<?soxtype urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0?>"
```

Import Namespace Processing Instruction

Description

An optional processing instruction with a soxtype target, indicating the complete sox namespace, (CBL or XCBL30) with version information ($1.0). It may be used by the Java Collaboration Editor to prepend to the underlying xCBL data. This can also be accomplished by hardcoding this PI string to prepend to the XCBL data in the Java Collaboration Service.

Required Values

A string. For example,

```
"<?import urn:x-
commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0?>"
```

3.4 e*Way Connection for Transmitter API Based Modules

When creating an XPC e*Way connection, the **c1mxpc.def** file is used. The following parameters are used to configure the e*Way Connection's configuration parameters necessary to facilitate communication with the Transmitter API and are organized into the following sections:

- Connector
- XPC Settings

3.4.1. Connector

The Connector Settings control basic operational parameters.

Type

Description

Specifies the type of connection.

Required Values

c1mxpc is the default value for Commerce One MarketSite XPC configuration connection.

Class

Description

Specifies the class name of the Commerce One MarketSite XPC connector object.

Required Values

com.stc.eways.c1mxpc.C1MXPCConnector is the default value.

Property Tag

Description

Specifies data source identity value required by EBobConnectorFactory.

Required Values

A string.

3.4.2. XPC Settings

The XPC Settings parameters contain the information needed to access XPC.

Document Type

Description

Specifies the document type for the messages being sent to MarketSite for a particular Collaboration.

Required Values

PurchaseOrder, **OrderStatus**, **PriceCheck, Request** or **AvailabilityCheck**.

Sender

Description

Specifies the MPID (MarketSite Participant ID, also referred to as Trading Partner ID or TPID) for the MarketSite sender.

Required Values

A valid MPID. Enter your MarketSite MPID since you are the sender.

Recipient

Description

Specifies the MPID for the MarketSite recipient.

Required Values

A valid MPID. Enter the MarketSite MPID for your supplier when running as a buyer.

Destination

Description

Specifies the destination for documents sent to MarketSite.

Required Values

A valid MarketSite destination. See the XPC documentation for more information.

XPC Root

Description

Specifies the root directory where XPC is installed.

Required Values

A valid root directory.

client.prop File Path

Description

Specifies the explicit location of your client.prop file. This is normally located under $XPCRootDirectory/bin.

Authentication and security is based on configuration parameters specified by the user (usually in the client.prop file). The e*Way must be informed where this file is located to enable the successful retrieval of information, when communicating with MarketSite.

Required Values

A valid path location.

Debug Level

Description

Specifies the level for debug logging information.

Required Values

debug, **info**, **warning**, **error**, **critical**, or **fatal**, .

Timeout

Description

Specifies the default timeout in milliseconds when sending documents to MarketSite.

Required Values

A integer.

Schema Path

Description

Specifies the schema path location.

Required Values

A valid path location.

Chapter 4

Implementation

This chapter discusses how to implement the Commerce One MarketSite e*Way in a production environment.

4.1 Implementation Process: Overview

To implement the Commerce One MarketSite e*Way within an e*Gate system, do the following:

- Define Event Type Definitions (ETDs) to package the data being exchanged with the ERP system(s).

Note: *See the default.prop file and XPC documents to find ETD values.*

- In the e*Gate Schema Designer, do the following:
  - Define Event Types for the ERP system.
  - Define Collaboration Rules to process Event data.
  - Configure the IQ Manager to suit your needs.
  - Define any IQs to which Event data is published prior to sending it to the external system.
  - Create one or more new e*Way components and configure their properties.
  - Within the e*Way component, configure the Collaborations to apply the required Collaboration Rules.
  - Define the necessary e*Way Connections.
- Use the e*Way Editor to set the e*Way's configuration parameters.
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections.

See **"Creating the Sample Schema" on page 38** for examples of how the above steps are combined to create a working implementation.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the e*Gate Schema Designer's online Help system.*

4.1.1. Considerations

The classes are located in stcc1marketsie.jar installed in ..\eGate\client\classes and .. eGate\Server\registry\repository\default\classes.

For the Commerce One MarketSite XPC service samples, each C1Config e*Way Connection corresponds to a C1 XPC service (inbound or outbound).

For the File e*Way (used as a feeder), ensure that the "MultipleRecordsPerFile" field is set to **No**, ensuring that XML content with carriage returns are not misinterpreted as multiple records, limiting the file input to one-per-file. This applies to the following containers:

- "send_feeder" in the buyerorderxpc schema
- "ChangeOrderTemplateFeeder" in the buyerxpc schema
- "AdvShipNoticeTemplateFeeder", "InvoiceTemplateFeeder" in supplierxpc schema

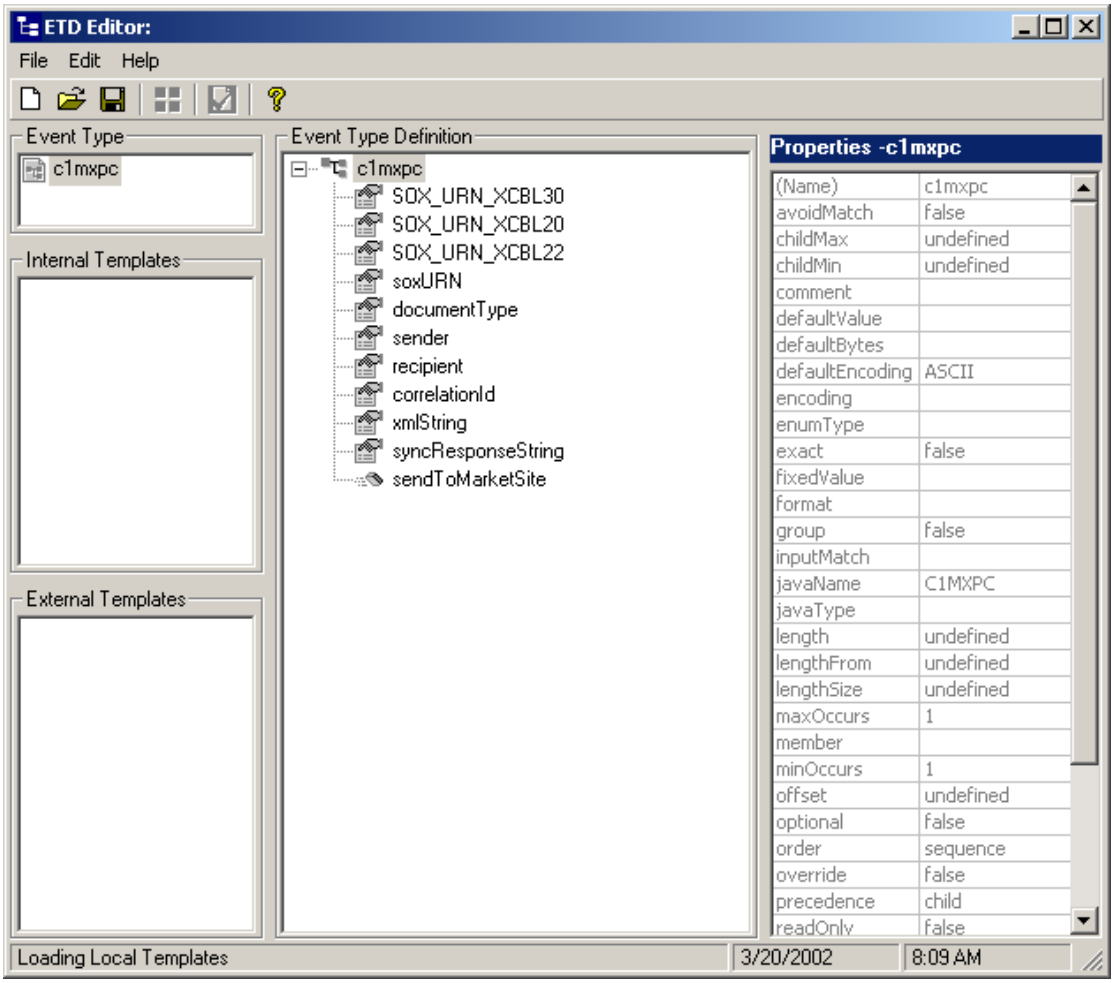
4.2 Event Types

The installation includes two Event Types created based on the xCBL libraries. Unless further customization is required, these Event Types should suffice.

4.2.1. TransmitterAPI : c1mxpc.xsc

The Event Type supplied for use with the Transmitter API is referred to as c1mxpc.xsc. It resides in the Default Schema, etd\c1mxpc\ (see Figure 4).

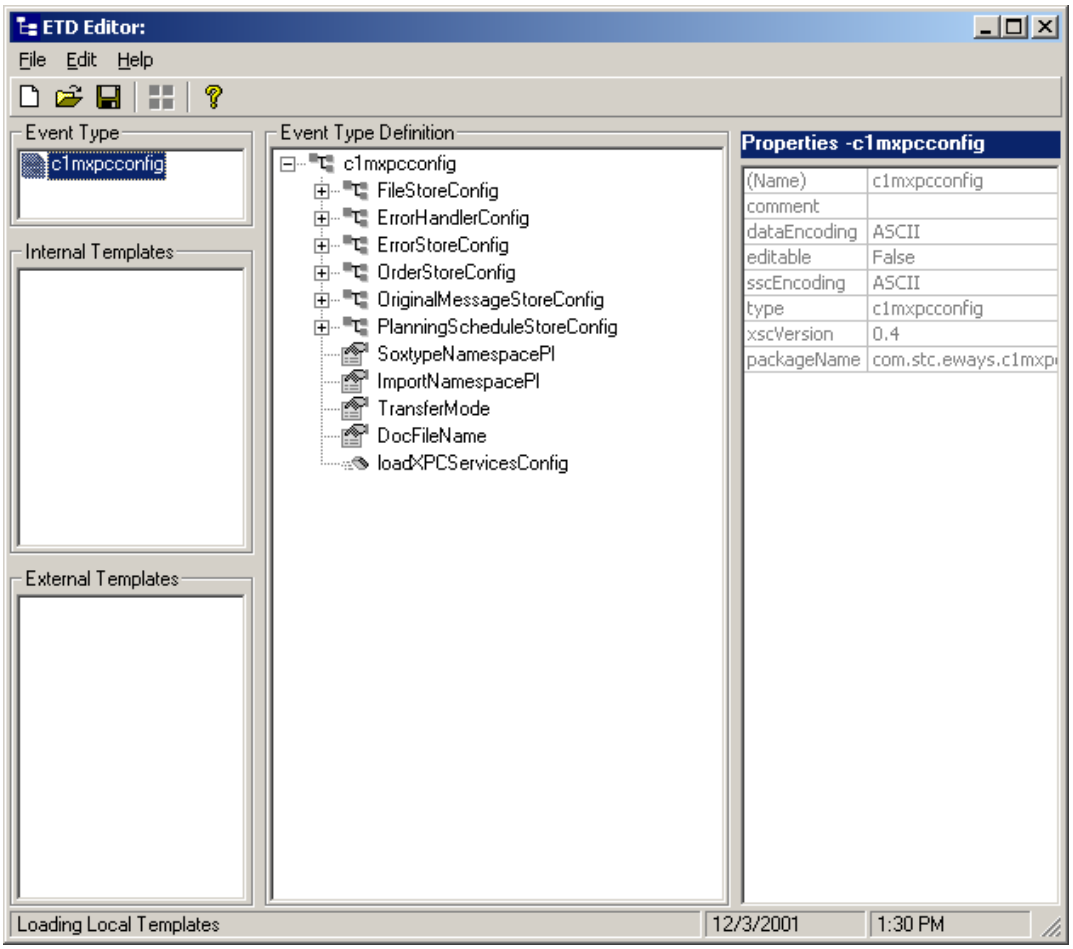
Figure 4 c1mxpc.xsc



4.2.2. XPC Server: c1mxpcconfig.xsc

The Event Type supplied for use with the XPC Server is referred to as c1mxpcconfig.xsc. It resides in the Default Schema, etd\c1mxpc\ (seeFigure 5).

Figure 5 c1mxpcconfig.xsc



4.3 Creating the Sample Schema

There are eight separate samples available on the installation CD.

- **The buyerorderXPC Sample Schema** on page 39, demonstrates the use of the Commerce One e*Way in implementing simple handling of outbound order XCBL documents and incoming order response documents. This schema relies on the Batch e*Way and File e*Way.
- **The supplierorderXPC Sample Schema** on page 56, demonstrates the use of the Commerce One e*Way in implementing handling of inbound order and outbound order response XCBL documents. This schema relies on the Batch e*Way and File e*Way
- **The TransmitterAsync Sample Schema** on page 62, demonstrates the use of the Commerce One e*Way in implementing the transmitter ETD component to send xCBL documents asynchronously to MarketSite.

- **The TransmitterSync Sample Schema** on page 66, demonstrates the use of the Commerce One e*Way in implementing the transmitter ETD component to send xCBL documents synchronously to MarketSite.
- **The buyerorderxpcftp Sample Schema** on page 70, demonstrates the use of the Commerce One e*Way in implementing FTP support for e*Gate to interface with the Commerce One XPC installed on another machine (as the counterpart for the buyerorderxpc sample schema for the simple buyer case). This schema also relies on Batch e*Way, File e*Way, and a running FTP server on the XPC machine.
- **The supplierxpc Sample Schema** on page 71, demonstrates the use of the Commerce One e*Way in implementing the handling of inbound order, change order, and outbound order response / invoice / advance shipment notice XCBL documents. This schema relies on the Batch e*Way and File e*Way.
- **The buyerxpc Sample Schema** on page 72, demonstrates the handling of not just outbound orders but also outbound change orders, as well as accepting and archiving inbound ASN (Advance Shipment Notice) and inbound invoice XCBL documents. This schema relies on the Batch e*Way and File e*Way.
- **The supplierxpcsync Sample Schema** on page 73, demonstrates the use of the Commerce One e*Way in implementing the simple handling of inbound and outbound XCBL synchronous documents (price check, order status, and availability check). This schema relies on e*Gate JMS and the File e*Way.

4.3.1. The buyerorderXPC Sample Schema

The buyerorderXPC sample schema demonstrates the use of the Commerce One e*Way in implementing simple handling of outbound order XCBL documents and incoming order response documents. This schema relies on the Batch e*Way and File e*Way.

After installing the sample schema, it must be configured before running. Each schema described in this document has a section of configuration instructions.

Table 3 Contents of the buyerorderXPC Sample Schema .zip

Directory	File(s)
	buyeroderxpc.ctl buyerorderxpc.exp

Directory	File(s)
buyerorderxpc\runtime\collaboration_rules\	dump_payload_cr.class dump_payload_cr.ctl dump_payload_cr.java dump_payload_cr.xpr dump_payload_cr.xts dump_payload_crBase.class dump_payload_eater_cr.class dump_payload_eater_cr.ctl dump_payload_eater_cr.java dump_payload_eater_cr.xpr dump_payload_eater_cr.xts dump_payload_eater_crBase.class ProcessC1In_java.class ProcessC1In_java.java ProcessC1In_java.ctl ProcessC1In_java.java ProcessC1In_java.xpr ProcessC1In_java.xts ProcessC1In_javaBase.class processC1out_java.class processC1out_java.ctl processC1out_java.java processC1out_java.xpr processC1out_java.xts processC1out_javaBase.class send_feeder_cr.class send_feeder_cr.ctl send_feeder_cr.java send_feeder_cr.xpr send_feeder_cr.xts send_feeder_crBase.class
buyerorderxpc\runtime\configs\c1mxpc\	C1ConfigInfo_order.cfg C1ConfigInfo_order.sc C1ConfigInfo_order_response.cfg C1ConfigInfo_order_response.sc
buyerorderxpc\runtime\configs\stewfile\	dump_payload_eater.cfg dump_payload_eater.sc feeder.cfg feeder.sc
buyerorderxpc\runtime\configs\stcewgeneric monk	dynamicBatchIn.cfg dynamicBatchIn.sc dynamicBatchOut.cfg dynamicBatchOut.sc

Directory	File(s)
buyerorderxpc\runtime\etd\	dynamicBatchReceiveData.jar dynamicBatchReceiveData.xsc dynamicBatchReceiveOrder.jar dynamicBatchReceiveOrder.jar dynamicBatchSendOrder.jar dynamicBatchSendOrder.xsc outputblob.jar outputblob.ssc outputblob.xsc
buyerorderxpc\runtime\etd\c1mxpc\	c1mxpcconfig.xsc
buyerorderxpc\runtime\etd\templates\xcbl\V30r2\lib\	Order.jar Order.xsc OrderResponse.jar OrderResponse.xsc

Configuring the buyerorderXPC Sample

Once the sample has been successfully imported into e*Gate, the user must configure it to correspond to the information as necessary. The following items should be examined:

- Each of the configuration files associated with the three e*Ways must be configured, as needed, saved, and promoted to runtime. Specifically, the following parameters must be addressed:
- The e*Way Connection configuration must be adjusted to suit the systems involved.
  - Root XPC, see **"XPC Config Root" on page 30**
  - Path for XPC Service use, see **"Default Property File Path" on page 30**
  - Additional XPC processing, xCBL 3.0 or xCBL 1.0, see **"Soxtype Namespace Processing Instruction" on page 31**
- Do not set Publish Status Record on Success, for the dynamic Batch based e*Way to Yes. If set to yes, the Batch-based e*Way publishes a "good error" record to e*Gate, with the format of batch_eway_error.dtd, when the payload has been successfully sent to the remote host. This can cause an exception to be thrown by the JCS, resulting from unexpected XML error message format. Sample error messages such as the following may be observed in the log file for the corresponding Batch e*Way:

  ```
  <batch_eWay_Data>, found '<batch_eWay_error>'
  ```

- Verify that the following is embedded in each new CommerceOne Java Collaboration that parses xCBL data types to suppress the inclusion of default namespaces (i.e., xmlns="...") as there is a #FIXED attribute for every element in the xCBL DTD as published:

  ```
  java.lang.System.setProperty("xml.marshal.noDefaultNamespace",
  "true");
  ```

  The XPC server deviates from this xCBL DTD convention.

- Set the **Process Outgoing Message Function** under **Monk configuration** for the Batch e*Way configuration to **batch-proc-out-c1**, not the **default batch-proc-out**.
- Set the **Exchange Data with External Function** under **Monk configuration** for the Batch e*Way configuration to **batch-exchange-data-c1**, not the default **batch-exchange-data**.
- Set the **File Transfer Method** under **External Host Setup** for the Batch e*Way configuration to FTP (even in the case that e*Gate and XPC are installed on the same machine, and no FTP is actually involved).
- Set **Enable Message Configuration** under **Dynamic Configuration** for the Batch e*Way configuration to **Yes** to enable dynamic Batch operation for the CommerceOne schema.
- Modify the "account code" information for the order_template file provided as part of the sample. For example:

```
<AccountCode>
  <Reference>
    <RefNum>Fill_in_your_account_code_here</RefNum>
      <RefDate>20001215T09:52:25</RefDate>
  </Reference>
</AccountCode>
```

  Alternately, the user can programmatically update the "account code" of the xCBL data within the processC1out_java Collaboration (after the order_template file is read as xCBL data).
- The archive directory for inbound xCBL files, after they are processed, is hardcoded in the Collaboration as:

```
"incoming_orderresponse_archived"
```

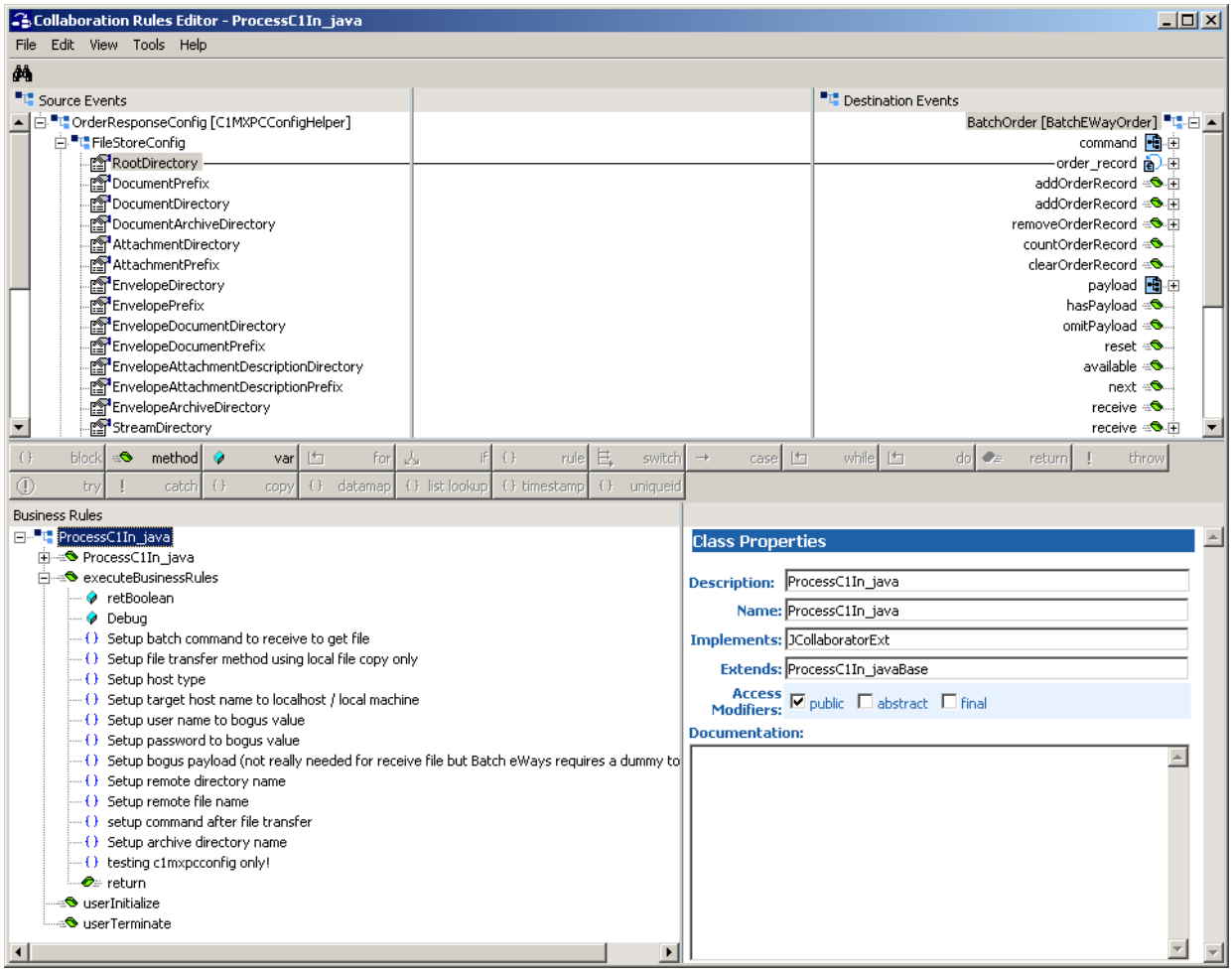
  This archive directory is placed in the user-created subdirectory:

```
<root>:\commerceone\Xpc\filestore\inbound
```

ProcessCIn_java Collaboration Rule

The ProcessCIn_java Collaboration Rule appears in Figure 6.

Figure 6 ProcessCIn_java Collaboration Rule



1. Each new rule is created by clicking the rule - as an expression button in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
2. "**Setup batch command to receive to get file**" is created by dragging the $text field, located under the Destination Event command node to the rule dialog box, selecting a set command, and entering "RECEIVE" as the parameter.
3. "**Setup file transfer method using local file copy only**" is created by dragging the $text field, located under BatchOrder\order_record\external_host_setup\file_transfer_method, selecting a set command, and entering "File Copy" as the parameter.
4. "**Setup host type**" is created by dragging the host_type field located under BatchOrder\external_host_setup\creating a set command, and entering the parameter.

5 "**Setup target host name to localhost / local machine**" is created by dragging the external_host_name field, located under BatchOrder\external_host_setup, creating a set command, and entering "localhost" as the parameter.

6 "**Setup user name to bogus value**" is created by dragging the user_name field, located under BatchOrder\external_host_setup, creating a set command, and entering "guest" as the parameter.

7 "**Setup password to bogus value**" is created by dragging the encrypted_password field, located under BatchOrder\external_host_setup, creating a set command, and entering "0123456789" as the parameter.

8 "**Setup bogus payload**" (not really needed for "receive files", but Batch e*Ways requires a value to get files) is created by dragging the $text field, located under the payload node, creating a set command, and entering "Place holder only!" as the parameter.

9 "**Setup remote directory name**" is created by dragging the remote_directory_name field, located under the BatchOrder, order_record\$1\subscribe_to_external\, creating a set command, and dragging the RootDirectory field, located under OrderResponseConfig\FileStoreConfig, creating a get function, and dropping it into the previously created setRemoteDirectoryName function as the parameter. In this case the sample then include + "/" + , drag the EnvelopeDocumentDirectory field also located under OrderResponseConfig\FileStoreConfig, creating a get function, and dropping it after the second plus sign (+).

10 "**Setup remote file name**" is created by dragging the remote_file_regexp field, located under the BatchOrder\order_record\$1\subscribe_to_external, creating a set command, and entering "Place holder only!" as the parameter and dragging the EnvelopeDocumentPrefix field, located under OrderResponseConfig\FileStoreConfig, creating a get function, and dropping it into the previously created setRemoteFileRegexp function. In this case the sample then include + "[a-zA-Z0-9-]*.xml".

Note: *Allow for hyphens and alphanumeric for file name suffix.*

11 "**Setup archive directory name**" is created by dragging the remote_rename_or_archive_name field, located under BatchOrder\order_record\$1\subscribe_to_external\remote_command_after_transfer, creating a set command, and dragging the RootDirectory field, located under OrderResponseConfig\FileStoreConfig, creating a get function, and dropping it into the previously created setRemoteRenameOrArchive function. In this case the sample then includes + "/" + "envelope_archive".

Note: *The sample hardcodes the archive directory for now (not really covered by XPC configuration)*

12 For debugging purposes, the "**testing c1mxpcconfig only!**" rule was created by including the following code in the Rule Dialog box:

```
/*
System.err.println("--------------FileStoreConfig-----------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getFileStoreConfig().getRootDirec
tory());
```

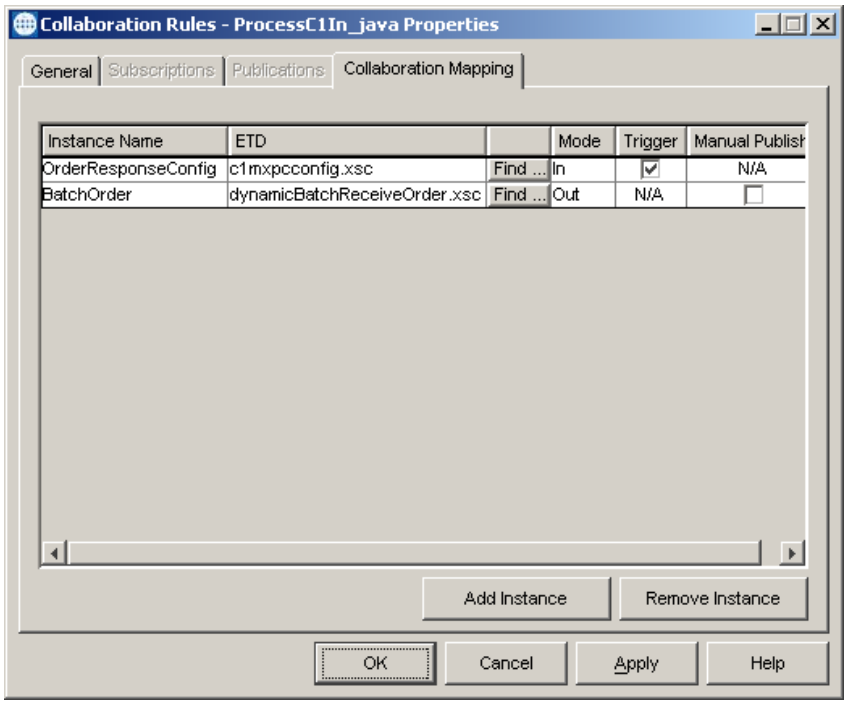
```
System.err.println("DocumentPrefix="+getOrderResponseConfig().getFileStoreConfig().getDocument
Prefix());
System.err.println("DocumentDirectory="
+getOrderResponseConfig().getFileStoreConfig().getDocumentDirectory());
System.err.println("DocumentArchiveDirectory=" +
getOrderResponseConfig().getFileStoreConfig().getDocumentArchiveDirectory());
System.err.println("AttachmentDirectory="+getOrderResponseConfig().getFileStoreConfig().getAtt
achmentDirectory());
System.err.println("AttachmentPrefix="+getOrderResponseConfig().getFileStoreConfig().getAttach
mentPrefix());
System.err.println("EnvelopeDirectory=" +
getOrderResponseConfig().getFileStoreConfig().getEnvelopeDirectory());
System.err.println("EnvelopePrefix=" +
getOrderResponseConfig().getFileStoreConfig().getEnvelopePrefix());
System.err.println("EnvelopeDocumentDirectory="+getOrderResponseConfig().getFileStoreConfig().
getEnvelopeDocumentDirectory());
System.err.println("EnvelopeDocumentPrefix="+getOrderResponseConfig().getFileStoreConfig().get
EnvelopeDocumentPrefix());
System.err.println("EnvelopeAttachmentDescriptionDirectory="+getOrderResponseConfig().getFileS
toreConfig().getEnvelopeAttachmentDescriptionDirectory());
System.err.println("EnvelopeAttachmentDescriptionPrefix="+getOrderResponseConfig().getFileStor
eConfig().getEnvelopeAttachmentDescriptionPrefix());
System.err.println("EnvelopeArchiveDirectory="+getOrderResponseConfig().getFileStoreConfig().g
etEnvelopeArchiveDirectory());
System.err.println("StreamDirectory="+getOrderResponseConfig().getFileStoreConfig().getStreamD
irectory());
System.err.println("StreamPrefix="+getOrderResponseConfig().getFileStoreConfig().getStreamPref
ix());
System.err.println("StreamExtension="+getOrderResponseConfig().getFileStoreConfig().getStreamE
xtension());
System.err.println("StreamArchiveDirectory="+getOrderResponseConfig().getFileStoreConfig().get
StreamArchiveDirectory());
System.err.println("----------------ErrorHandlerConfig----------------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getErrorHandlerConfig().getRootDi
rectory());
System.err.println("FileSourceDirectory="+getOrderResponseConfig().getErrorHandlerConfig().get
FileSourceDirectory());
System.err.println("FileTargetDirectory="+getOrderResponseConfig().getErrorHandlerConfig().get
FileTargetDirectory());
System.err.println("FilePrefix="+getOrderResponseConfig().getErrorHandlerConfig().getFilePrefi
x());
System.err.println("FileExtension="+getOrderResponseConfig().getErrorHandlerConfig().getFileEx
tension());
System.err.println("----------------ErrorStoreConfig----------------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getErrorStoreConfig().getRootDire
ctory());
System.err.println("DocumentDirectory="+getOrderResponseConfig().getErrorStoreConfig().getDocu
mentDirectory());
System.err.println("DocumentPrefix="+getOrderResponseConfig().getErrorStoreConfig().getDocumen
tPrefix());
System.err.println("----------------OrderStoreConfig----------------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getOrderStoreConfig().getRootDire
ctory());
System.err.println("DocumentDirectory="+getOrderResponseConfig().getOrderStoreConfig().getDocu
mentDirectory());
System.err.println("DocumentPrefix="+getOrderResponseConfig().getOrderStoreConfig().getDocumen
tPrefix());
System.err.println("DocumentExtension="+getOrderResponseConfig().getOrderStoreConfig().getDocu
mentExtension());
System.err.println("----------------OriginalMessageStoreConfig----------------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getOriginalMessageStoreConfig().g
etRootDirectory());
System.err.println("DocumentDirectory="+getOrderResponseConfig().getOriginalMessageStoreConfig
().getDocumentDirectory());
System.err.println("DocumentPrefix="+getOrderResponseConfig().getOriginalMessageStoreConfig().
getDocumentPrefix());
System.err.println("DocumentExtension="+getOrderResponseConfig().getOriginalMessageStoreConfig
().getDocumentExtension());
System.err.println("----------------PlanningScheduleStoreConfig----------------------");
System.err.println("RootDirectory="+getOrderResponseConfig().getPlanningScheduleStoreConfig().
getRootDirectory());
System.err.println("DocumentDirectory="+getOrderResponseConfig().getPlanningScheduleStoreConfi
g().getDocumentDirectory());
System.err.println("DocumentPrefix="+getOrderResponseConfig().getPlanningScheduleStoreConfig()
.getDocumentPrefix());
System.err.println("DocumentExtension="+getOrderResponseConfig().getPlanningScheduleStoreConfi
g().getDocumentExtension());
*/
retBoolean = true
```

Note: *Uncomment to turn on debugging!*

Collaboration Rule Mapping

The Collaboration Mapping associated with the ProcessCIn_java Collaboration Rule is set as displayed in Figure 7.

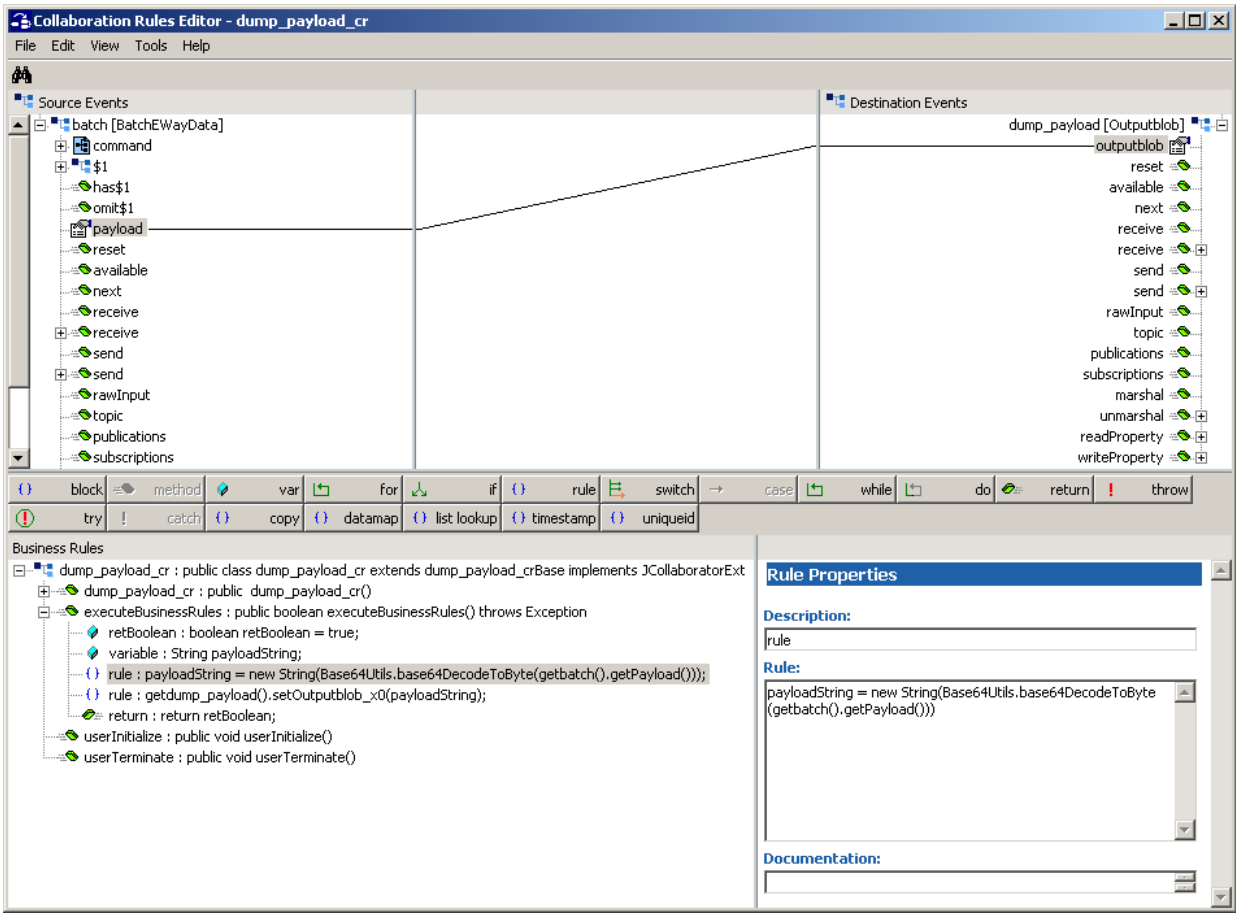
Figure 7 ProcessCIn_java Collaboration Mapping



dump_payload_cr Collaboration Rule

The dump_payload_cr Collaboration Rule appears in Figure 8.

Figure 8 dump_payload_cr Collaboration Rule



1. Each new rule is created by clicking the **rule** button on the toolbar in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
2. The first rule is created by using **Java Imports** under the Tools menu. It can also be created by entering the following in the Rule dialog box:

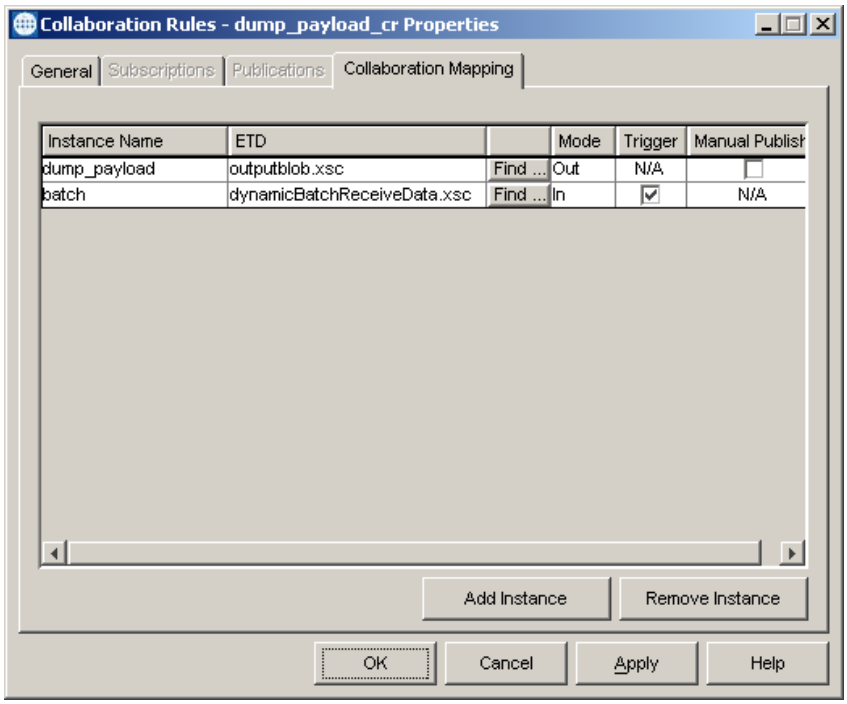
```
payloadString = new String(Base64Utils.base64DecodeToByte()
```

3. The second rule is created by dragging the setOutputBlob_x0 function to the Rule Dialog Box, entering the string "payloadString" to be passed in as the parameter value.

dump_payload_cr Collaboration Mapping

The Collaboration mapping associated with the dump_payload_cr Collaboration Rule is set as displayed in Figure 9.

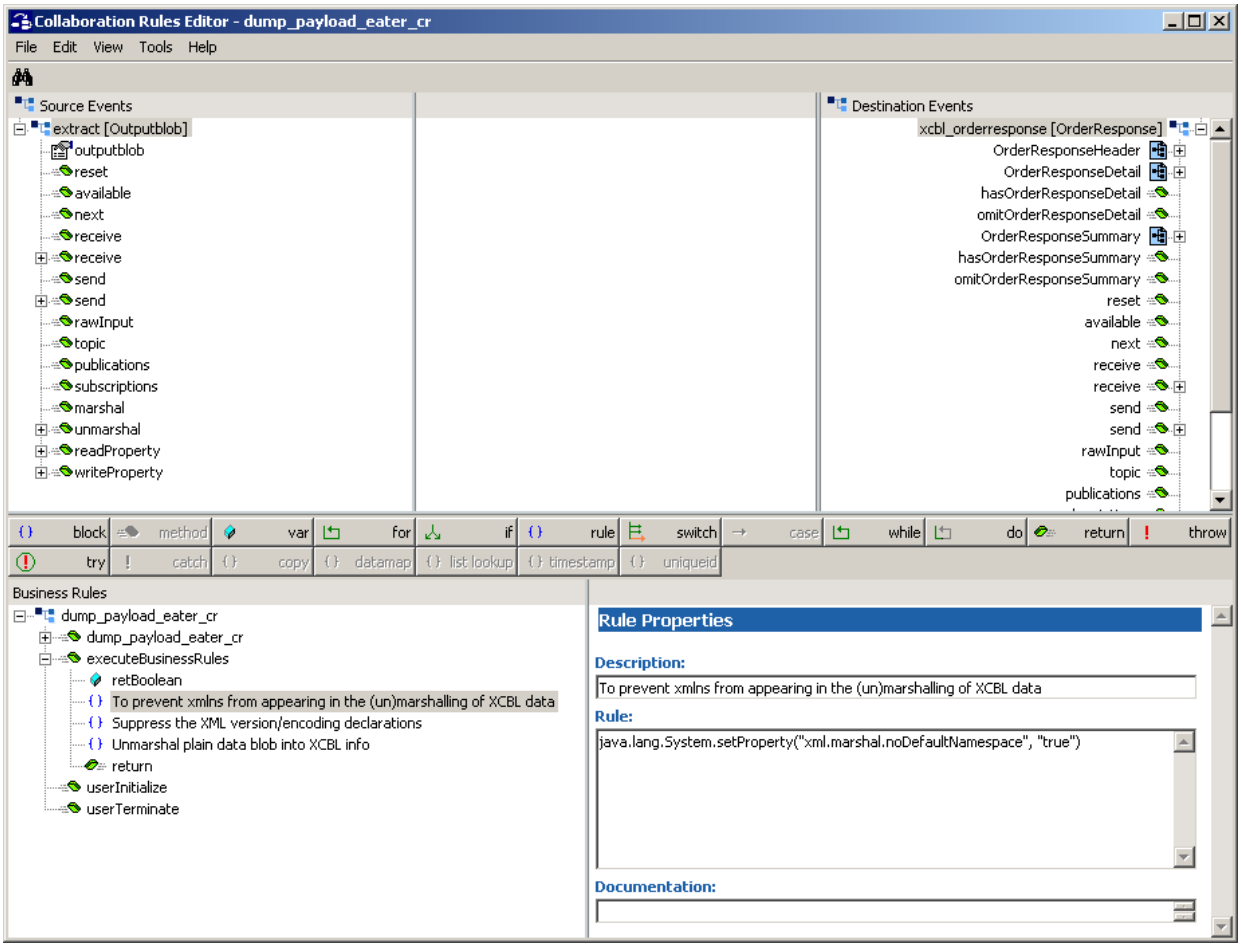
Figure 9 dump_payload_cr Properties



dump_payload_eater_cr Collaboration Rule

The dump_payload_eater_cr Collaboration Rule appears in Figure 10.

Figure 10 dump_payload_eater_cr



1 Each new rule is created by clicking the **rule** button on the toolbar in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.

2 "**To prevent xmlns from appearing in the (un)marshalling of XCBL data**" is created by entering the following in the Rule Dialog box:

```
java.lang.System.setProperty("xml.marshal.noDefaultNamespace",
"true")
```

3 The "**Suppress the XML version/encoding declarations**" rule is created by entering the following in the Rule Dialog box:

```
getxcbl_order().includeXmlDeclaration(false);
```

This is necessary because the XML declaration is not compatible with XPC server.

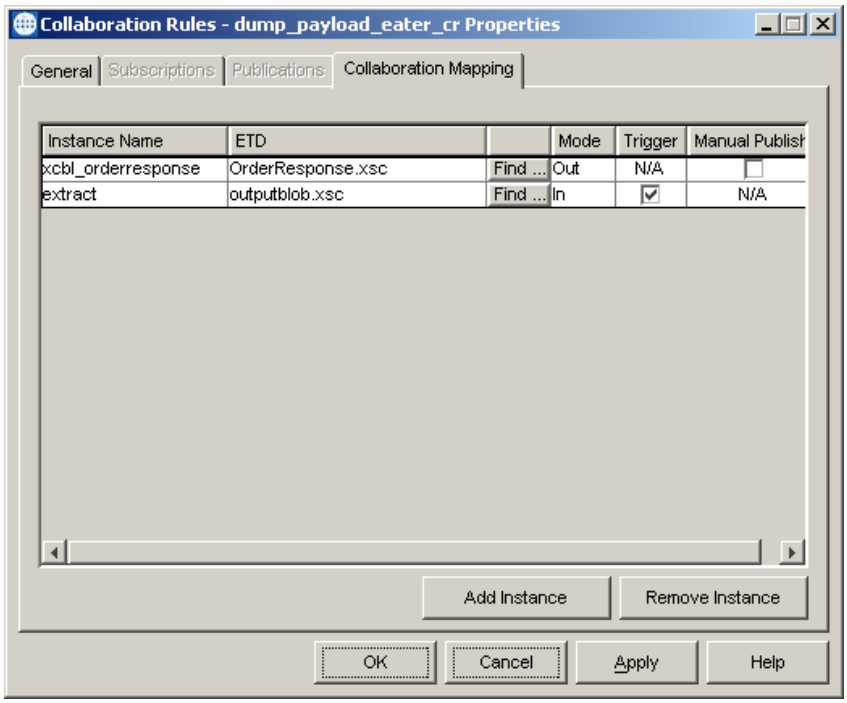
4 "**Unmarshal plain data blob into XCBL info**" is created by dragging the unmarshall function to the Rule Dialog box, and dragging the outputblob field,

located under the extract node, into the dialog box that appears. (Click ok to continue)

dump_payload_eater_cr Collaboration Mapping

The Collaboration mapping associated with the dump_payload_eater_cr Collaboration Rule is set as displayed in Figure 11.

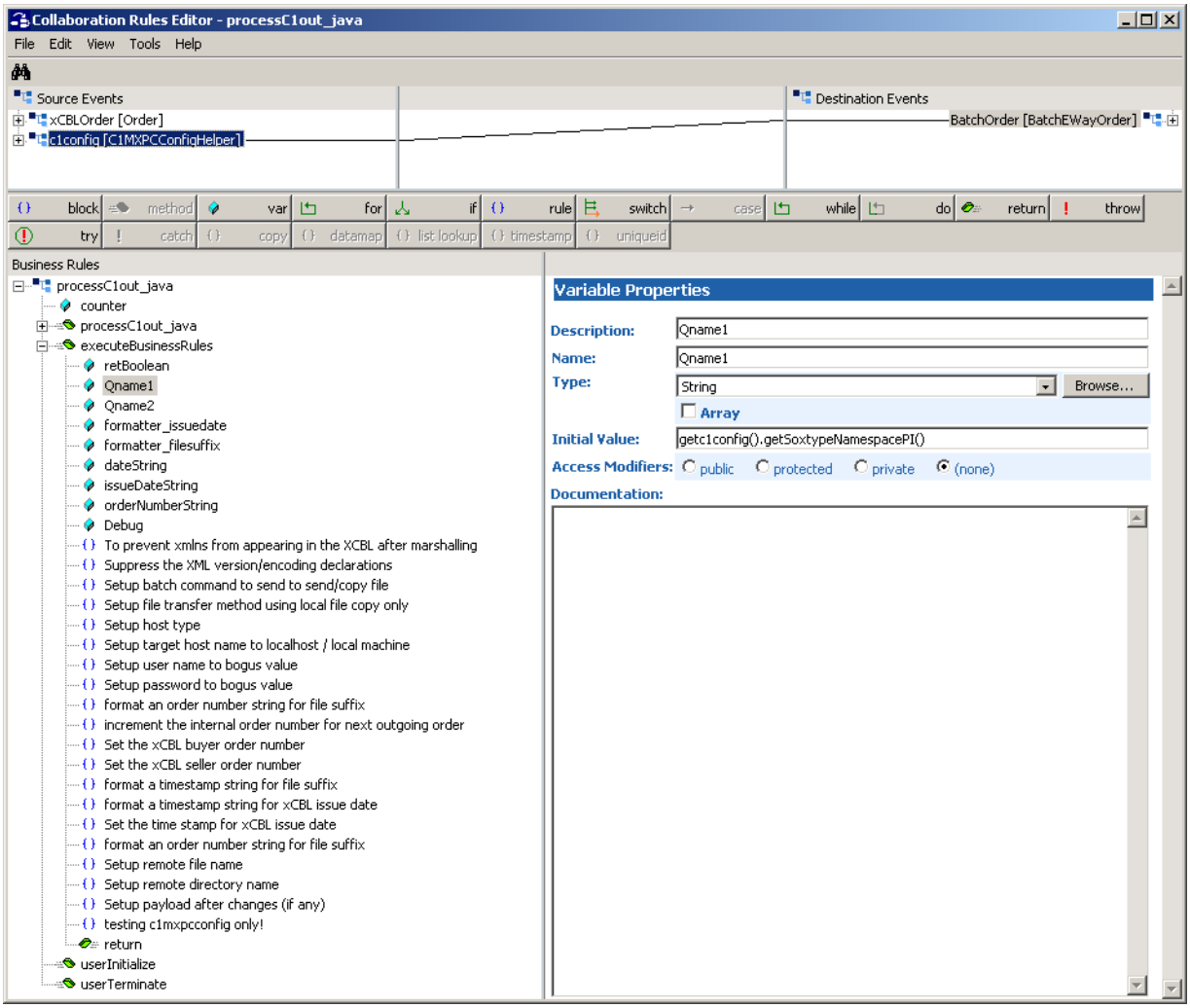
Figure 11 dump_payload_eater_cr



processC1out_java Collaboration Rule

The processC1out_java Collaboration Rule appears in Figure 11.

Figure 12 processC1out_java



1. Each new rule is created by clicking the **rule** button on the toolbar in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
2. Each variable is created in the same manner as the above mentioned rules. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
3. "**Qname1**" is created by dragging the SoxNamespacePI field, located below the PlanningScheduleStoreConfig node, selecting a get function, into the Initial Value Dialog box.
4. "**Qname2**" is created by dragging the ImportNamespacePI field, located below the PlanningScheduleStoreConfig node, selecting a get function, into the Initial Value Dialog box.

5 "**formatter_issuedate**" is created by clicking the **var** button on the toolbar and entering formatter_issuedate" in the Variable Properties Dialog box as the description and name. java.text.SimpleDateFormat is selected as the type. The following text is entered in the **Initial Value** field:

new java.text.SimpleDateFormat ("yyyyMMdd'T'HH:mm:ss")

The following documentation is entered into the Documentation field:

1) Notice that MM (month, 1 thru 12) is specified to 2 count so that only number is displayed.

2) Notice that HH is specified to display 24 hour clock (0-23) to match that of the "SDateTime" class in the "com.commerceone.xdk.maplibs.jbschema.jbdatatypes" package.

3) 'T' is the separator as specified for "SDateTime" class.

6 "**formatter_filesuffix**" is created by clicking the **var** button on the toolbar and entering formatter_filesuffix" in the Variable Properties Dialog box as the description and name. java.text.SimpleDateFormat is selected as the type. The following text is entered in the **Initial Value** field:

new java.text.SimpleDateFormat ("yyyyMMdd'T'HH:mm:ss")

The following documentation is entered into the Documentation field:

1) Notice that MM (month, 1 thru 12) is specified to 2 count so that only number is displayed.

2) Notice that HH is specified to display 24 hour clock (0-23) to match that of the "SDateTime" class in the 'com.commerceone.xdk.maplibs.jbschema.jbdatatypes" package.

3) 'T' is the separator as specified for "SDateTime" class.

4) Removed the colon separator (":") for time (hour/minute/second) because colon cannot be used as any part of file name in most O/S.

For Access Modifiers, **(none)** is selected.

7 "**dateString**" is included as a constant variable for the sample. It is created by clicking the **var** button on the toolbar and entering "dateString" in the Variable Properties Dialog box as the description and name. The type defaults to String.

8 "**issueDateString**" is included as a constant variable for the sample. It is created in the same manner as "dateString" variable.

9 "**orderNumberString**" is included as a constant variable for the sample. It is created in the same manner as "dateString" variable.

10 "**Debug**" It is created by clicking the **var** button on the toolbar and entering "Debug" in the Variable Properties Dialog box as the description and name. For Type, boolean is selected and "false" is entered in the Initial Value field.

11 "**To prevent xmlns from appearing in the XCBL after marshalling**" is created by entering the following in the Rule Dialog box:

```
java.lang.System.setProperty("xml.marshal.noDefaultNamespace",
"true")
```

12 The "**Suppress the XML version/encoding declarations**" rule is created by entering the following in the Rule Dialog box:

```
getxcbl_order().includeXmlDeclaration(false);
```

This is necessary because the XML declaration is not compatible with XPC server.

13 "**Setup batch command to send to send/copy file**" is created by dragging the $text field, located under BatchOrder\command, creating a set function, to the Rule Dialog box, and entering "SEND" as the parameter.

14 "**Setup file transfer method using local file copy only**" is created by dragging the $text field, located under BatchOrder\order_record\external_host_setup\file_transfer_method, creating a set function, to the Rule Dialog box, and entering "File Copy" as the parameter.

15 "**Setup host type**" is created by dragging the host_name field, located under BatchOrder\order_record\external_host_setup, to the Rule Dialog box, and entering the operating system type.

16 "**Setup target host name to localhost / local machine**" is created by dragging the external_host_name field, located under BatchOrder\order_record\external_host_setup, to the Rule Dialog box, and entering "localhost".

17 "**Setup user name to bogus value**" is created by dragging the user_name field, located under BatchOrder\order_record\external_host_setup, to the Rule Dialog box, and entering "guest".

18 "**Setup password to bogus value**" is created by dragging the encrypted_password field, located under BatchOrder\order_record\external_host_setup, to the Rule Dialog box, and entering "0123456789".

19 "**format an order number string for file suffix**" is created by entering the following:

```
orderNumberString = new Integer(counter).toString()
```

20 "**increment the internal order number for next outgoing order**" is created by entering the following:

```
counter = counter + 1
```

21 "**Set the xCBL buyer order number**" is created by dragging the OrderIssueDate field, located under xCBLOrder\OrderHeader, creating a set function, and entering issueDateString as the parameter.

22 "**format an order number string for file suffix**" is created by defining orderNumberString = by dragging the BuyerOrderNumber field, located under xCBLOrder\OrderHeader\$1\OrderNumber, and

23 "**increment the internal order number for next outgoing order**" is created by entering:

```
counter = counter + 1
```

24 "**Set the xCBL buyer order number**" is created by dragging the BuyerOrderNumber field, located under xCBLOrder\OrderHeader\$1\OrderNumber, creating a set function, and entering orderNumberString as the parameter.

25 "**Set the xCBL seller order number**" is created by dragging the SellerOrderNumber field, located under xCBLOrder\OrderHeader\$1\OrderNumber, creating a set function, and entering orderNumberString + "s" as the parameter.

26 "**format a timestamp string for file suffix**" is created by entering:

```
dateString = (String) formatter_filesuffix.format(new
java.util.Date())
```

27 "**format a timestamp string for xCBL issue date**" is created by entering:

```
issueDateString = (String) formatter_issuedate.format(new
java.util.Date())
```

28 "**Set the time stamp for xCBL issue date**" is created by dragging the OrderIssueDate field, located under xCBLOrder\OrderHeader, creating a set function, and entering issueDateString as the parameter.

29 "**format an order number string for file suffix**" is created by defining orderNumberString =, and dragging the BuyerOrderNumber field, located under xCBLOrder\OrderHeader\$1\OrderNumber, creating a get function, and dropping it following the orderNumberString =.

30 "**Setup remote file name**" is created by dragging the remote_file_name field, located under BatchOrder\order_record\$1\publish_to_external, creating a set function, and dragging DocumentPrefix, located under c1config\FileStoreConfig, dropping it as the parameter into the setSetupRemoteFileName function, creating a get function. In this case, + "-" + dateString + ".xml was also added.

31 "**Setup remote directory name**" is created by dragging the remote_directory_name field, located under BatchOrder\order_record\$1\publish_to_external, creating a set function, and dragging RootDirectory, located under c1config\FileStoreConfig, dropping it as the parameter into the setSetupRemoteDirectoryName function, creating a get function. In this case, + "-" + was entered, dragging DocumentDirectory, located under BatchOrder\order_record\$1\publish_to_external, creating a get function, directly after the + "/" +.

32 "**Setup payload after changes (if any)**" is created by dragging the $text field, located under BatchOrder\Payload, creating a set function, followed by:

```
Base64Utils.byteToBase64String((Qname1 + Qname2 +
getxCBLOrder().toString()).getBytes())
```

33 "**testing c1mxpcconfig only!**" is created by adding the following code:

```
/*
System.err.println("----------------FileStoreConfig---------------
-------");
System.err.println("RootDirectory="+getc1config().getFileStoreConf
ig().getRootDirectory());
System.err.println("DocumentPrefix="+getc1config().getFileStoreCon
fig().getDocumentPrefix());
System.err.println("DocumentDirectory="
+getc1config().getFileStoreConfig().getDocumentDirectory());
System.err.println("DocumentArchiveDirectory=" +
getc1config().getFileStoreConfig().getDocumentArchiveDirectory());
System.err.println("AttachmentDirectory="+getc1config().getFileSto
reConfig().getAttachmentDirectory());
System.err.println("AttachmentPrefix="+getc1config().getFileStoreC
onfig().getAttachmentPrefix());
System.err.println("EnvelopeDirectory=" +
getc1config().getFileStoreConfig().getEnvelopeDirectory());
System.err.println("EnvelopePrefix=" +
getc1config().getFileStoreConfig().getEnvelopePrefix());
System.err.println("EnvelopeDocumentDirectory="+getc1config().getF
ileStoreConfig().getEnvelopeDocumentDirectory());
```

```
System.err.println("EnvelopeDocumentPrefix="+getc1config().getFile
StoreConfig().getEnvelopeDocumentPrefix());
System.err.println("EnvelopeAttachmentDescriptionDirectory="+getc1
config().getFileStoreConfig().getEnvelopeAttachmentDescriptionDire
ctory());
System.err.println("EnvelopeAttachmentDescriptionPrefix="+getc1con
fig().getFileStoreConfig().getEnvelopeAttachmentDescriptionPrefix(
));
System.err.println("EnvelopeArchiveDirectory="+getc1config().getFi
leStoreConfig().getEnvelopeArchiveDirectory());
System.err.println("StreamDirectory="+getc1config().getFileStoreCo
nfig().getStreamDirectory());
System.err.println("StreamPrefix="+getc1config().getFileStoreConfi
g().getStreamPrefix());
System.err.println("StreamExtension="+getc1config().getFileStoreCo
nfig().getStreamExtension());
System.err.println("StreamArchiveDirectory="+getc1config().getFile
StoreConfig().getStreamArchiveDirectory());
System.err.println("----------------ErrorHandlerConfig----------
----------");
System.err.println("RootDirectory="+getc1config().getErrorHandlerC
onfig().getRootDirectory());
System.err.println("FileSourceDirectory="+getc1config().getErrorHa
ndlerConfig().getFileSourceDirectory());
System.err.println("FileTargetDirectory="+getc1config().getErrorHa
ndlerConfig().getFileTargetDirectory());
System.err.println("FilePrefix="+getc1config().getErrorHandlerConf
ig().getFilePrefix());
System.err.println("FileExtension="+getc1config().getErrorHandlerC
onfig().getFileExtension());
System.err.println("----------------ErrorStoreConfig------------
--------");
System.err.println("RootDirectory="+getc1config().getErrorStoreCon
fig().getRootDirectory());
System.err.println("DocumentDirectory="+getc1config().getErrorStor
eConfig().getDocumentDirectory());
System.err.println("DocumentPrefix="+getc1config().getErrorStoreCo
nfig().getDocumentPrefix());
System.err.println("----------------OrderStoreConfig------------
--------");
System.err.println("RootDirectory="+getc1config().getOrderStoreCon
fig().getRootDirectory());
System.err.println("DocumentDirectory="+getc1config().getOrderStor
eConfig().getDocumentDirectory());
System.err.println("DocumentPrefix="+getc1config().getOrderStoreCo
nfig().getDocumentPrefix());
System.err.println("DocumentExtension="+getc1config().getOrderStor
eConfig().getDocumentExtension());
System.err.println("----------------OriginalMessageStoreConfig----
------------------");
System.err.println("RootDirectory="+getc1config().getOriginalMessa
geStoreConfig().getRootDirectory());
System.err.println("DocumentDirectory="+getc1config().getOriginalM
essageStoreConfig().getDocumentDirectory());
System.err.println("DocumentPrefix="+getc1config().getOriginalMess
ageStoreConfig().getDocumentPrefix());
System.err.println("DocumentExtension="+getc1config().getOriginalM
essageStoreConfig().getDocumentExtension());
System.err.println("----------------PlanningScheduleStoreConfig---
-------------------");
System.err.println("RootDirectory="+getc1config().getPlanningSched
uleStoreConfig().getRootDirectory());
System.err.println("DocumentDirectory="+getc1config().getPlanningS
cheduleStoreConfig().getDocumentDirectory());
```

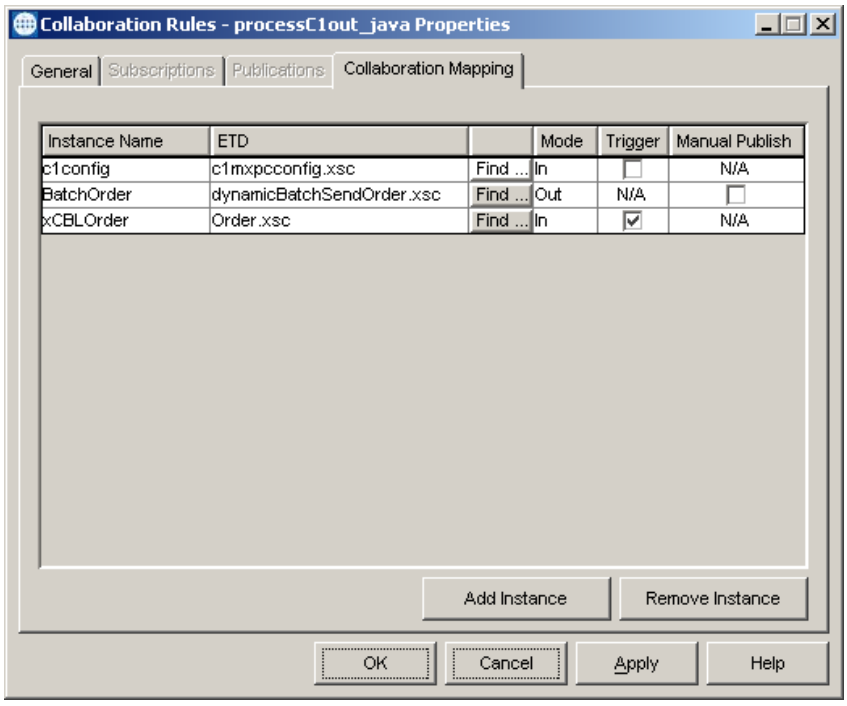
```
System.err.println("DocumentPrefix="+getc1config().getPlanningSche
duleStoreConfig().getDocumentPrefix());
System.err.println("DocumentExtension="+getc1config().getPlanningS
cheduleStoreConfig().getDocumentExtension());
*/
retBoolean = true
```

Note: *Uncomment to turn on debugging!*

processC1out_java Collaboration Mapping

The Collaboration Mapping associated with the processC1out_java Collaboration Rule set as displayed in Figure 13.

Figure 13 processC1out_java



send_feeder_cr Collaboration Rule

The send_feed_cr Collaboration Rule is created as basic pass through Collaboration Rule, which both publishes to and subscribes to an xCBL_event (configured using the Order.xsc definition). For more information on default Collaboration Rules, see the *e*Gate Integrator Collaboration Services Reference Guide*.

4.3.2. The supplierorderXPC Sample Schema

The supplierorderXPC sample schema demonstrates the use of the Commerce One e*Way in implementing handling of inbound order and outbound order response XCBL documents. This schema relies on the Batch e*Way and File e*Way

After installing the sample schema, it must be configured before running.

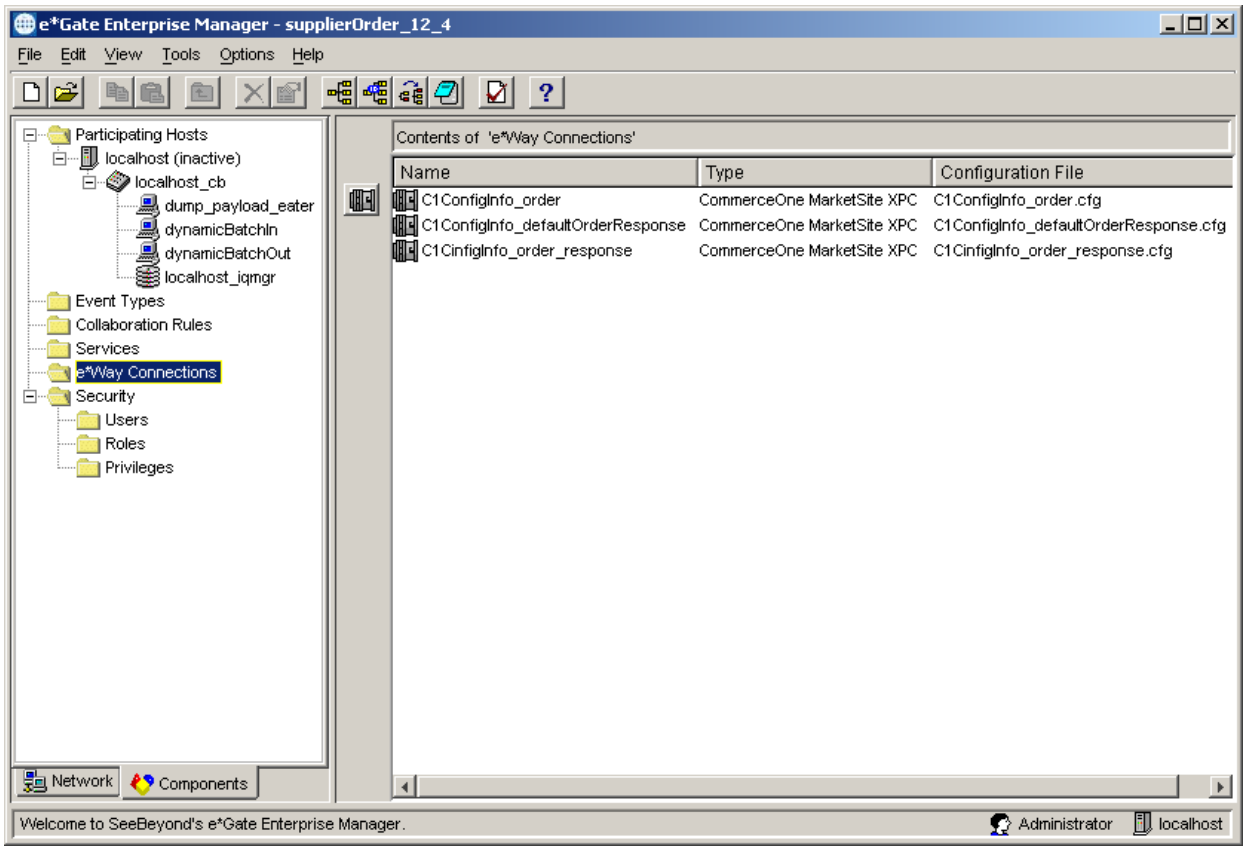
Table 4 Contents of the supplierorderXPC.zip file

Directory	File(s)
	supplierorderxpc.ctl supplierorderxpc.exp
supplierorderxpc\runtime\configs\c1mxpc\	C1ConfigInfo_defaultOrderResponse.cfg C1ConfigInfo_defaultOrderResponse.sc C1ConfigInfo_order.cfg C1ConfigInfo_order.sc C1ConfigInfo_order_response.cfg C1ConfigInfo_order_response.sc
supplierorderxpc\runtime\configs\stcewfile\	dump_payload_eater.cfg dump_payload_eater.sc feeder.cfg feeder.sc
supppliorderxpc\runtime\configs\stcewgenericmonk\	dynamicBatchIn.cfg dynamicBatchIn.sc dynamicBatchOut.cfg dynamicBatchOut.sc
supplierorderxpc\runtime\etd\	dynamicBatchReceiveData.jar dynamicBatchReceiveData.xsc dynamicBatchReceiveOrder.jar dynamicBatchReceiveOrder.xsc dynamicBatchSendOrder.jar dynamicBatchSendOrder.xsc outputblob.jar outputblob.xsc outputblob.ssc RxcFileName.jar RxcFileName.ssc RxcFileName.xsc
supplierorderxpc\runtime\etd\c1mxpc\	c1mxpcconfig.xsc
supplierorderxpc\runtime\etd\templates\xcbl\V30r2\lib\	Order.jar Order.xsc OrderResponse.jar OrderResponse.xsc

Directory	File(s)
supplierorderxpc\runtime\collaboration_rules	dump_payload_cr.class dump_payload_cr.ctl dump_payload_cr.java dump_payload_cr.xpr dump_payload_cr.xts dump_payload_crBase.class dump_payload_eater_cr.class dump_payload_eater_cr.ctl dump_payload_eater_cr.java dump_payload_eater_cr.xpr dump_payload_eater_cr.xts dump_payload_eater_crBase.class ProcessC1DefaultOrderResponse.class ProcessC1DefaultOrderResponse.ctl ProcessC1DefaultOrderResponse.java ProcessC1DefaultOrderResponse.xpr ProcessC1DefaultOrderResponse.xts ProcessC1DefaultOrderResponseBase.class ProcessC1In_java.class ProcessC1In_java.ctl ProcessC1In_java.java ProcessC1In_java.xpr ProcessC1In_java.xts ProcessC1In_javaBase.class processC1out_java.class processC1out_java.ctl processC1out_java.java processC1out_java.xpr processC1out_java.xts processC1out_javaBase.class ProcessC1PayloadDefaultOrderResponse_Java.class ProcessC1PayloadDefaultOrderResponse_Java.ctl ProcessC1PayloadDefaultOrderResponse_Java.java ProcessC1PayloadDefaultOrderResponse_Java.xpr ProcessC1PayloadDefaultOrderResponse_Java.xts ProcessC1PayloadDefaultOrderResponse_JavaBase.class send_feeder_cr.class send_feeder_cr.ctl send_feeder_cr.java send_feeder_cr.xpr send_feeder_cr.xts send_feeder_crBase.class

Figure 14 shows the Components view of the SupplierOrder Sample schema.

Figure 14 SupplierOrder Sample Schema



Configuring the SupplierOrder Sample

Once the sample has been successfully imported into e*Gate, the user must configure it to correspond to the information as necessary. The following items should be examined:

- Each of the configuration files associated with the three e*Ways must be configured, as needed, saved, and promoted to runtime. Specifically, the following parameters must be addressed:
- The e*Way Connection configuration must be adjusted to suit the systems involved.
  - Drive letter/prefix, see **"XPC Config Root" on page 30**
  - Path for XPC Service use, see **"Default Property File Path" on page 30**
  - Additional XPC processing, xCBL 3.0 or xCBL 1.0, see **"Soxtype Namespace Processing Instruction" on page 31**
- Do not set Publish Status Record on Success, for the dynamic Batch based e*Way to Yes. If set to yes, the Batch-based e*Way publishes a "good error" record to e*Gate, with the format of batch_eway_error.dtd, when the payload has been successfully sent to the remote host. This can cause an exception to be thrown by the JCS, resulting from unexpected XML error message format. Sample error messages such as the following may be observed in the log file for the corresponding Batch e*Way:

```
<batch_eWay_Data>, found '<batch_eWay_error>'
```

- Verify that the following is embedded in each new CommerceOne Java Collaboration that parses xCBL data types to suppress the inclusion of default namespaces (i.e., xmlns="...") as there is a #FIXED attribute for every element in the xCBL DTD as published:

  ```
  java.lang.System.setProperty("xml.marshal.noDefaultNamespace",
  "true");
  ```

  The XPC server deviates from this xCBL DTD convention.

- Set the **Process Outgoing Message Function** under **Monk configuration** for the Batch e*Way configuration to **batch-proc-out-c1**, not the **default batch-proc-out**.
- Set the **Exchange Data with External Function** under **Monk configuration** for the Batch e*Way configuration to **batch-exchange-data-c1**, not the default **batch-exchange-data**.
- Set the **File Transfer Method** under **External Host Setup** for the Batch e*Way configuration to FTP (even in the case that e*Gate and XPC are installed on the same machine, and no FTP is actually involved).
- Set **Enable Message Configuration** under **Dynamic Configuration** for the Batch e*Way configuration to **Yes** to enable dynamic Batch operation for the CommerceOne schema.
- The archive directory for inbound xCBL files after they are processed within the JCS as:

  ```
  "incoming_order_archived"
  ```

  It must be created manually in the:

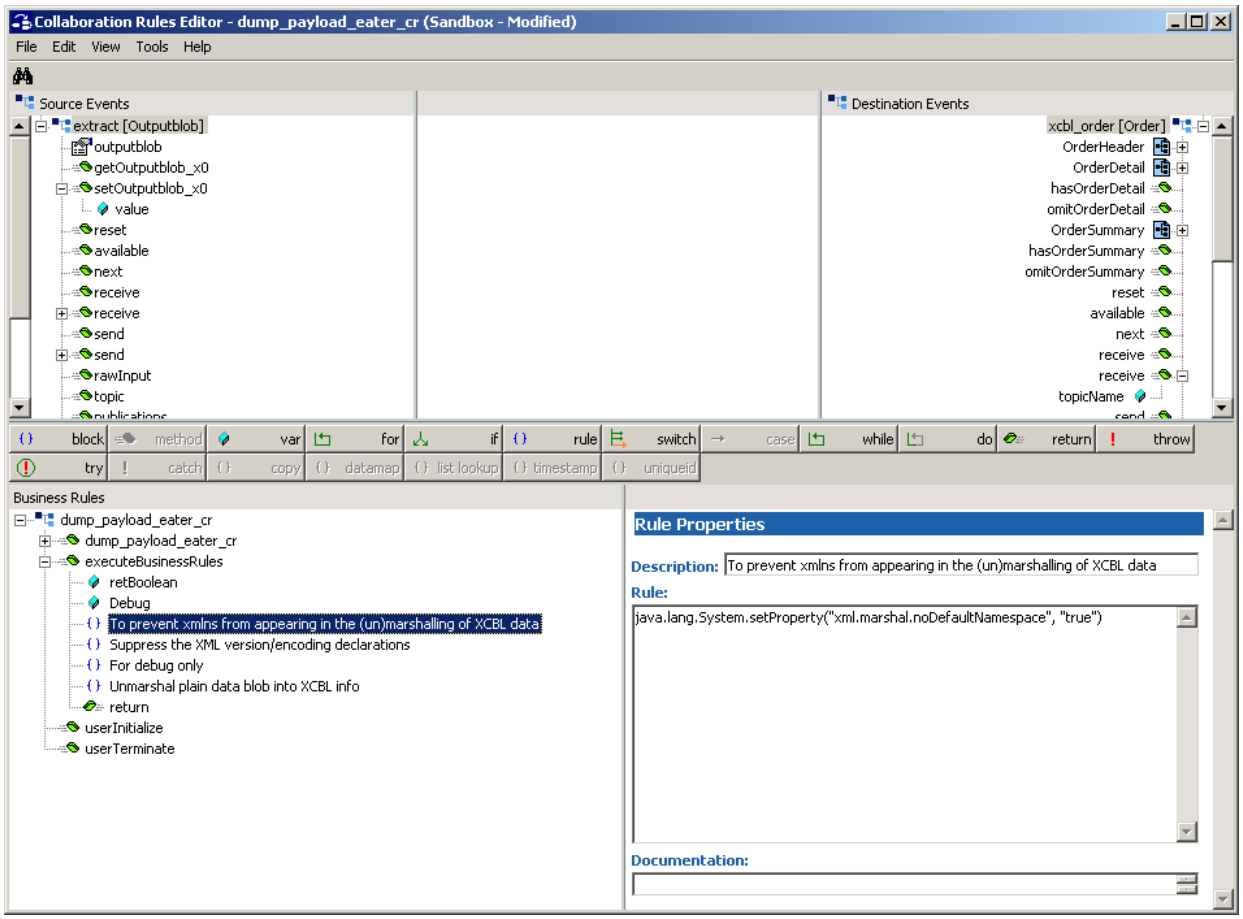
  ```
  <root directory>:\commerceone\Xpc\filestore\inbound subdirectory
  ```

- For the supplier application, there is a default order response xCBL file created by the XPC for every incoming order. This schema picks up this file and places it in the appropriate outbound directory, without archiving the default order response file. There is no processing of the corresponding xCBL (such as changing the date/time).

dump_payload_eater_cr Collaboration Rule

The dump_payload_eater_cr Collaboration Rule appears in the Figure 15:

Figure 15 dump_payload_eater_cr Collaboration Rule



1. Each new rule is created by clicking the rule - as an expression button in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
2. Each variable is created in the same manner as the above mentioned rules. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide.*
3. "**Debug**" is created by clicking on the var button on the center toolbar, and selecting boolean as the type value.
4. "**To prevent xmlns from appearing in the (un)marshalling of XCBL data**" is created by entering the following:

```
java.lang.System.setProperty("xml.marshal.noDefaultNamespace",
"true")
```

5. The "**Suppress the XML version/encoding declarations**" rule is created by entering the following in the Rule Dialog box:

```
getxcbl_order().includeXmlDeclaration(false);
```

6 The next rule, which creates a debug if statement, is created by entering the following:

```
if (Debug)
{
System.err.println("------------dump_payload_eater_cr.java--------
------------");
System.err.println("getextract().getOutputblob_x0()=" +
getextract().getOutputblob_x0());
}
```

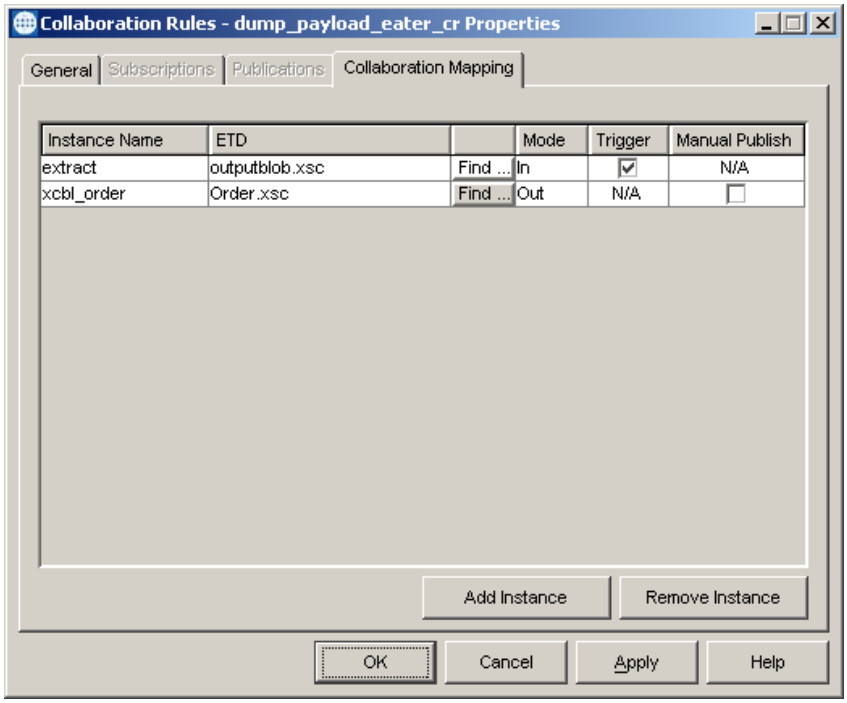
In the sample, the if statement is self contained, although the if - conditional statement button could have been used as well.

7 "**Unmarshal plain data blob into XCBL info**" is created by dragging the unmarshall method, located under xcbl_Order, dragging the outputblob field, located under extract into the Parameters dialog box that opens.

dump_payload_eater_cr Collaboration Rule Mapping

The Collaboration Mapping associated with the dump_payload_eater_cr Collaboration Rule is set as displayed in Figure 16.

Figure 16 dump_payload_eater_cr Mapping



4.3.3. The TransmitterAsync Sample Schema

The TransmitterAsync sample schema demonstrates the use of the Commerce One e*Way in implementing the transmitter ETD component to send xCBL documents asynchronously to MarketSite.

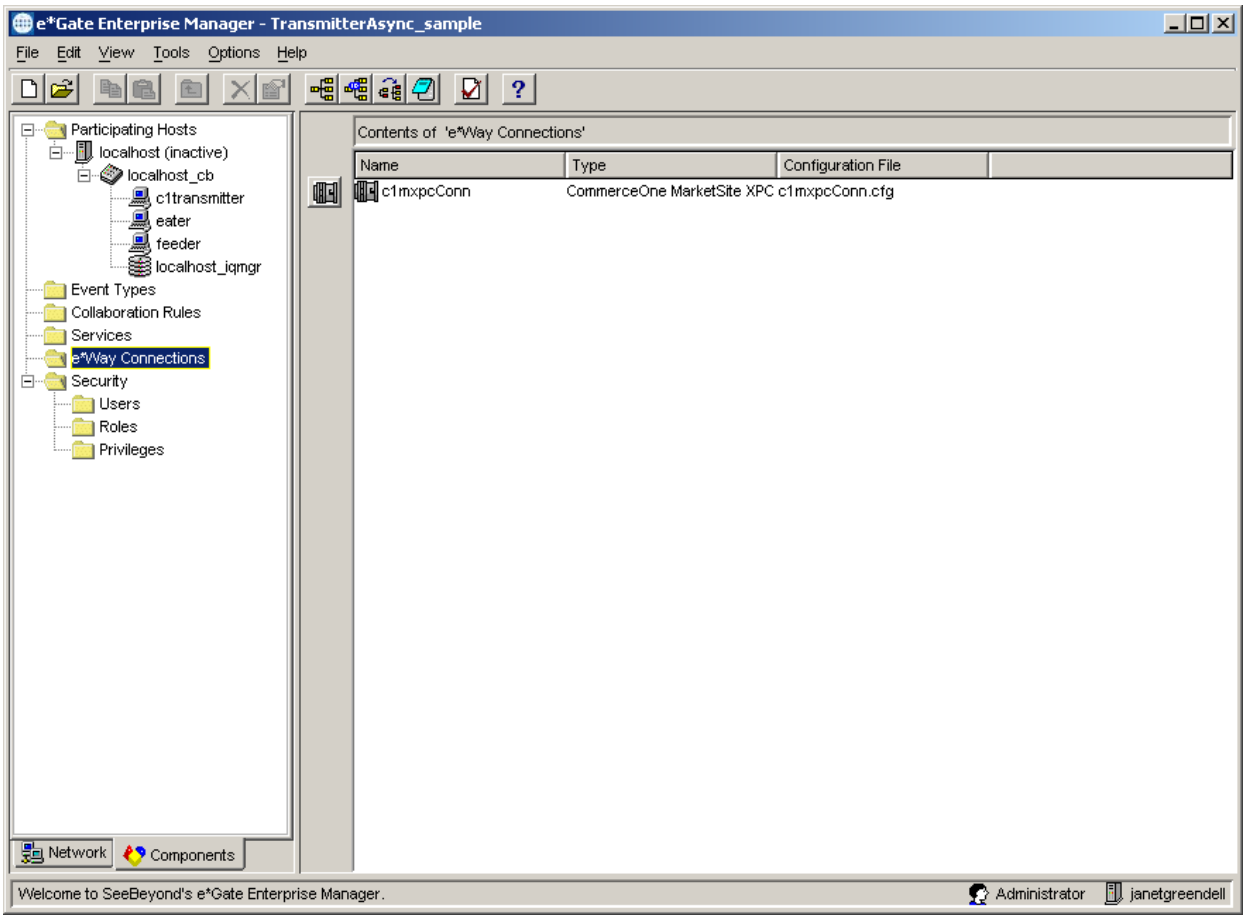
After installing the sample schema, it must be configured before running. The TransmitterAsync sample sends an Order to MarketSite. The response to that order can be obtained at a later time using XPC , rather than the Transmitter API.

Table 5 Contents of the TransmitterAsync.zip file

Directory	File(s)
	AsyncTransmitter.ctl AsyncTransmitter.exp
AsyncTransmitter\runtime\collaboration_rules\	c1collabrule.class c1collabrule.ctl c1collabrule.java c1collabrule.xpr c1collabrule.xts c1collabruleBase.class
AsyncTransmitter\runtime\configs\c1mxpc\	c1mxpcConn.sc c1mxpcConn.cfg c1mxpcConn1.sc c1mxpcConn1.cfg
AsyncTransmitter\runtime\configs\steway\	c1transmitter.cfg c1transmitter.sc
AsyncTransmitter\runtime\configs\stcewfile\	eater.cfg eater.sc feeder.cfg feeder.sc
AsyncTranmsitter\runtime\etd\	testblob.jar testblob.ssc testblob.xsc

Figure 17 shows the Components view of the TransmitterAsync Sample schema.

Figure 17 TransmitterAsync Sample Schema



Configuring the AsyncTransmitter Sample

Once the sample has been successfully imported into e*Gate, the user must configure it to correspond to the information as necessary. The following items should be examined:

- Each of the configuration files associated with the three e*Ways must be configured, as needed, saved, and promoted to runtime.
- The e*Way Connection configuration must be adjusted to suit the systems involved.
- Before executing the Java Collaborations that use the Transmitter ETD, the .ctl file for their Collaboration Rule must be modified as follows:

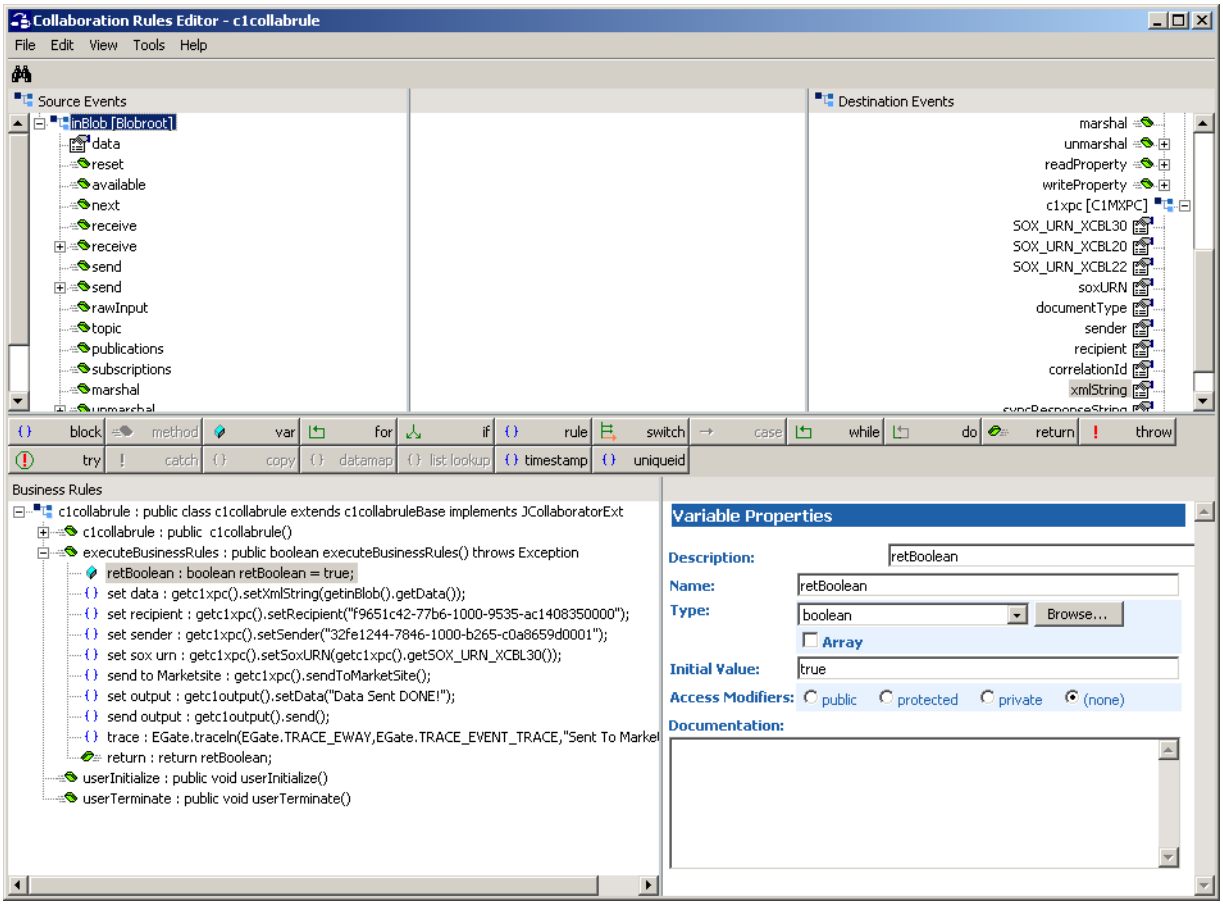
  Move the line containing the stcjcs.jar to the bottom of the import lines, but above the Collaboration Rules Java files. Any CommerceOne .jar file must be listed before stcjcs.jar.

  Delete the file from the client directory, and place a copy of the modified .ctl in the server directory.

c1collabrule

The c1collabrule appears in Figure 18.

Figure 18 c1collabrule Collaboration Rule



1. Each new rule is created by clicking the **rule** button on the toolbar in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.
2. "**set data**" is created by dragging the data field, located under inBlob under the Source Events, and dropping it on to the xmlString field, located under c1xpc under Destination Events.
3. "**set recipient**" is created by dragging the recipient field, located beneath c1mxpc, to the Rule dialog box, selecting set as the function-type. The Recipient TPID is then entered. If left blank, the value entered in the configuration file defaults in.
4. "**set sender**" is created by dragging the sender field, located beneath c1mxpc, to the Rule dialog box, selecting set as the function-type. The Recipient TPID is then entered. If left blank, the value entered in the configuration file defaults in.
5. "**set sox urn**" information is created by dragging and dropping the soxURN from the Destination Events, selecting set as the function-type. The correct

"SOX_URN_xCBL" is then dropped in as the parameter for the setSoxURN method.

6 "**send to Marketsite**" is created by dragging and dropping the sendToMarketSite method into the Rule dialog box.

7 "**set output**" is created by dragging the data field, located under the c1output node, to the Rule dialog box, then entering the desired expression, for example "Data Sent DONE!"

8 "**send output**" is created by dragging and dropping the send method into the Rule dialog box.

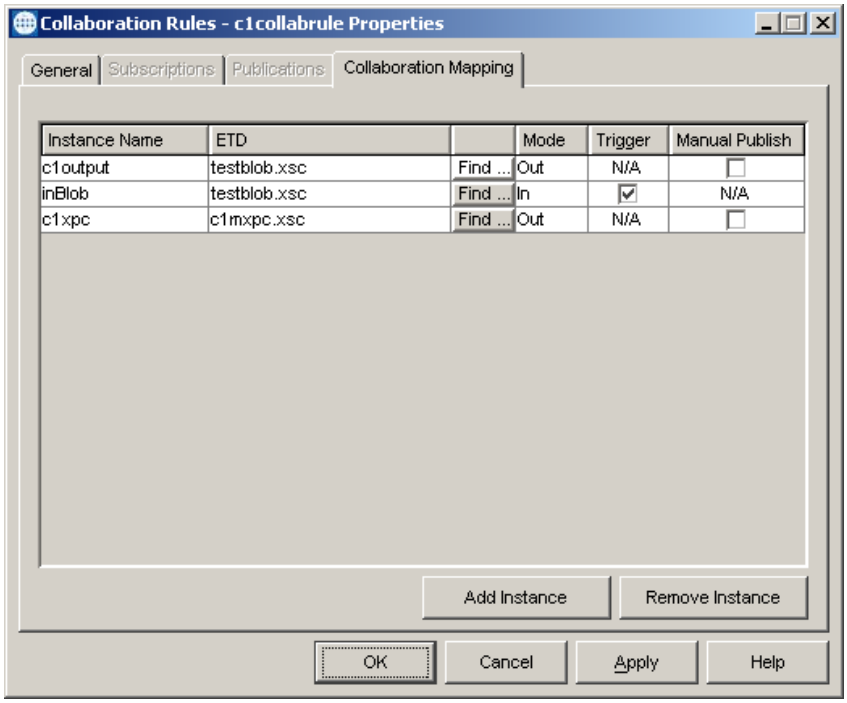
9 "**trace**" is created by entering the following into the Rule dialog box:

```
EGate.traceln(EGate.TRACE_EWAY,EGate.TRACE_EVENT_TRACE,"Sent To
MarketSite")
```

c1collabrule Collaboration Rule Mapping

The Collaboration Mapping associated with the c1collabrule Collaboration Rules is set as displayed in Figure 19.

Figure 19 c1collabrule Mapping



4.3.4. The TransmitterSync Sample Schema

The TransmitterSync sample schema demonstrates the use of the Commerce One e*Way in implementing the transmitter ETD component to send xCBL documents synchronously to MarketSite.

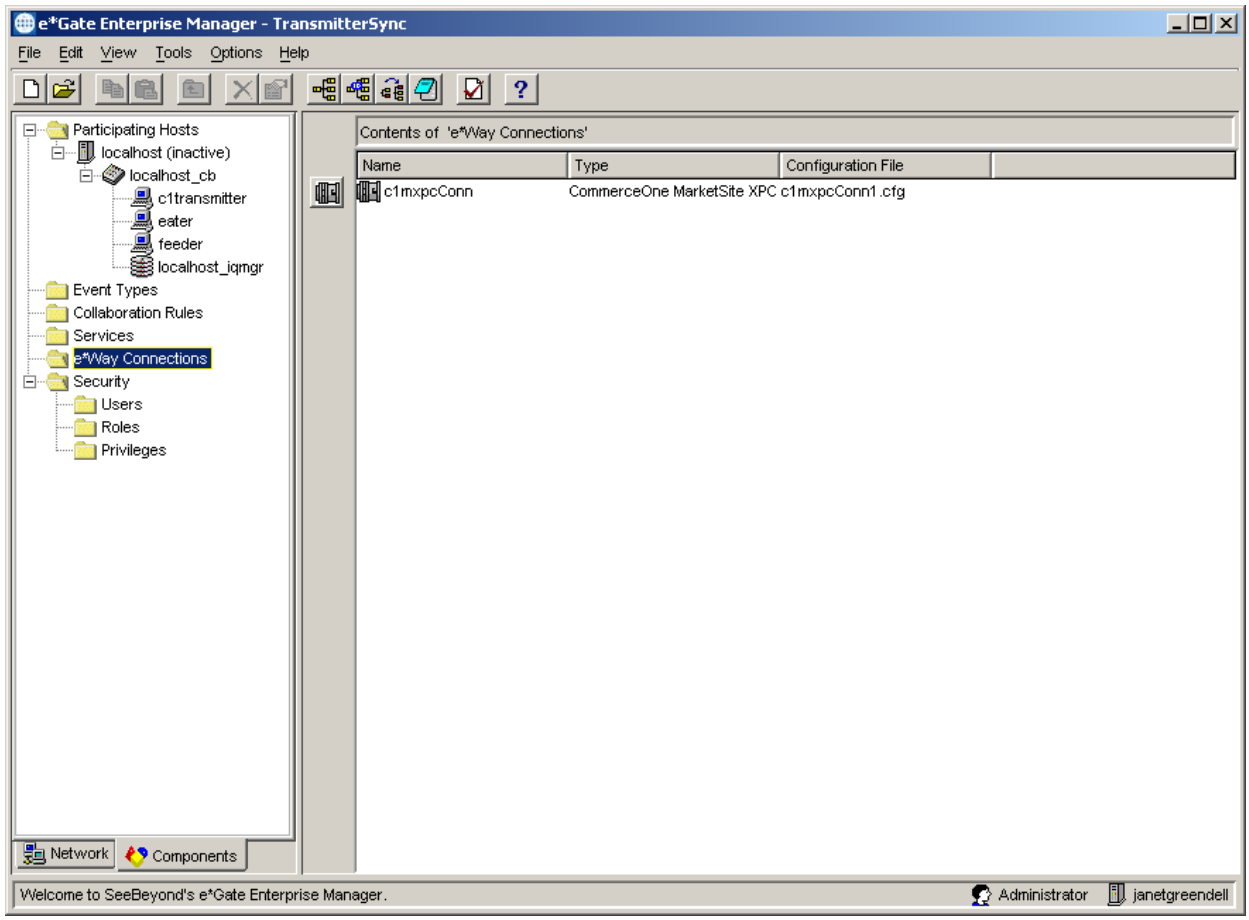
After installing the sample schema, it must be configured before running.

Table 6 Contents of the TransmitterSync.zip file

Directory	File(s)
	TransSync.ctl TransSync.exp
TransSync\runtime\collaboration_rules\	cr_MarketsiteBase.class cr_Marketsite.xts cr_Marketsite.xpr cr_Marketsite.java cr_Marketsite.ctl cr_Marketsite.class c1collabrule.class c1collabrule.ctl c1collabrule.java c1collabrule.xpr c1collabrule.xts c1collabruleBase.class
TransSync\runtime\configs\c1mxpc	c1mxpcConn.sc c1mxpcConn1.cfg c1mxpcConn1.sc
TransSync\runtime\configs\stceway\	c1transmitter.cfg c1transmitter.sc
TransSync\runtime\configs\stcewfile\	eater.cfg eater.sc feeder.cfg feeder.sc
TransSync\runtime\etd\	teestblob.jar testblob.ssc testblob.xsc
TransSync\runtime\etd\c1mxpc\	c1mxpc.xsc
TransSync\sandbox\collaboration_rules\	c1collabrule.xpr c1collabrule.xts
TransSync\sandbox\etd\	rtjar.ctl common.ctl c1mxpc.xsc

Figure 20 shows the Components view of the TransmitterSync Sample schema.

Figure 20 TransmitterSync Sample Schema



Configuring the TransmitterSync Sample

Once the sample has been successfully imported into e*Gate, the user must configure it to correspond to the information as necessary. The following items should be examined:

- Each of the configuration files associated with the three e*Ways must be configured, as needed, saved, and promoted to runtime.
- The e*Way Connection configuration must be adjusted to suit the systems involved.
- Before executing the Java Collaborations that use the Transmitter ETD, the .ctl file for their Collaboration Rule must be modified as follows:

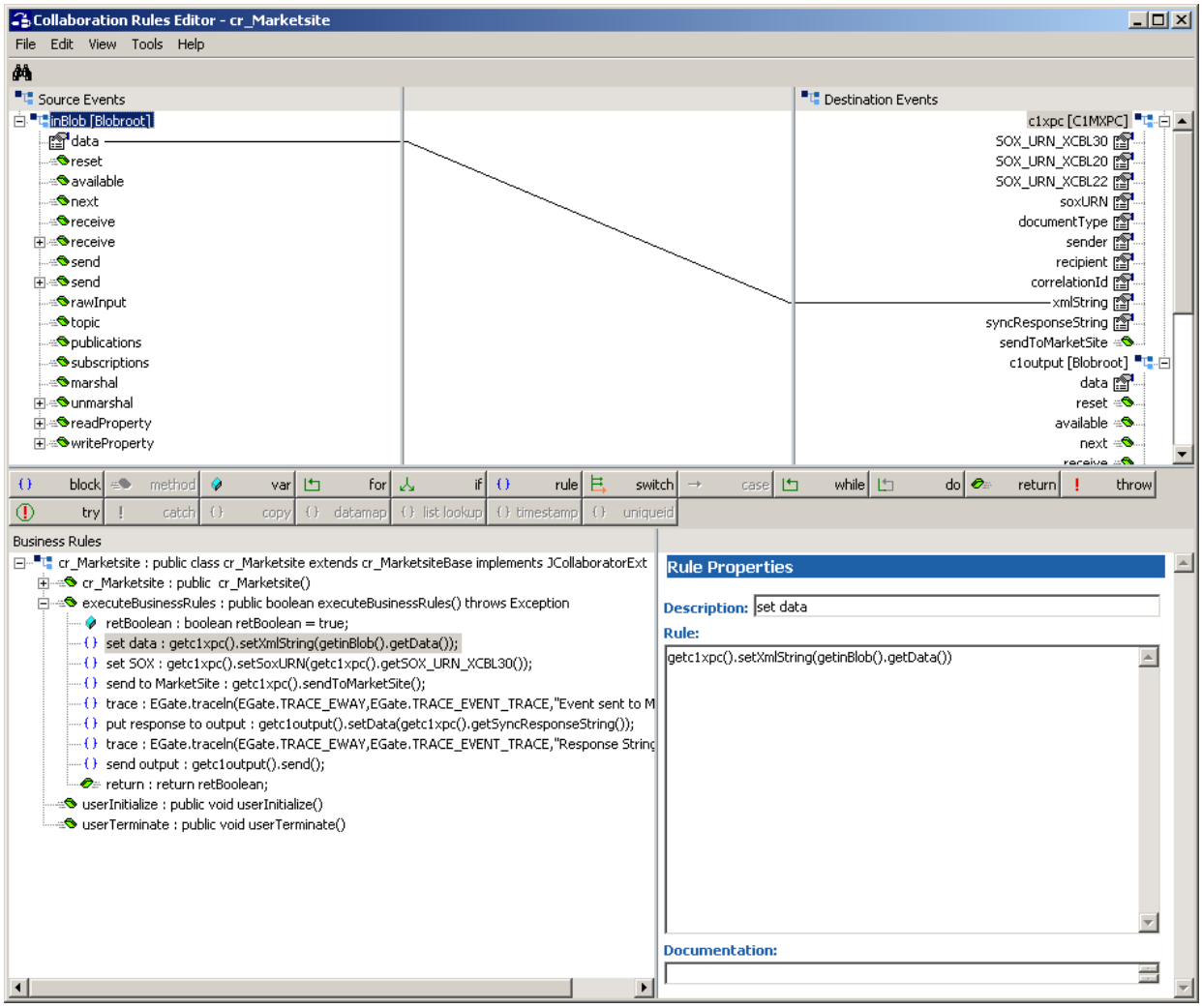
  Move the line containing the stcjcs.jar to the bottom of the import lines, but above the Collaboration Rules Java files. Any CommerceOne .jar file must be listed before stcjcs.jar.

  Delete the file from the client directory, and place a copy of the modified .ctl in the server directory.

cr_Marketsite Collaboration Rule

The cr_Marketsite Collaboration Rule appears in Figure 21.

Figure 21 cr_Marketsite Collaboration Rule



1 Each new rule is created by clicking the **rule** button on the toolbar in the center of the Collaboration Rules Editor. For more information about using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.

2 "**set data**" is created by dragging the data field, located under inBlob under the Source Events, and dropping it on to the xmlString field, located under c1xpc under Destination Events.

3 "**set Sox**" is created by dragging the soxURN field, located under c1xpc to the Rule dialog box, creating a set function, and dragging the SOX_URN_XCBL30 field, located under c1xpc, creating a get function, and dropping it into setSoxURN as the parameter.

4 "**send to Marketsite**" is created by dragging the sendtoMarketsite method under c1xpc to the Rule dialog box.

5 "**trace**" is created by entering the following into the Rule dialog box:

```
EGate.traceln(EGate.TRACE_EWAY,EGate.TRACE_EVENT_TRACE,"Event sent
to MarketSite
```

6 "**put response to output**" is created by dragging the data field, located under c1output to the Rule dialog box, creating a set function, and dragging the syncResponseString field, located under c1xpc, and dropping it into setData as the parameter. The setSyncResponseString() is then changed to getSyncResponseString().

7 "**trace**" is created by entering the following into the Rule dialog box:

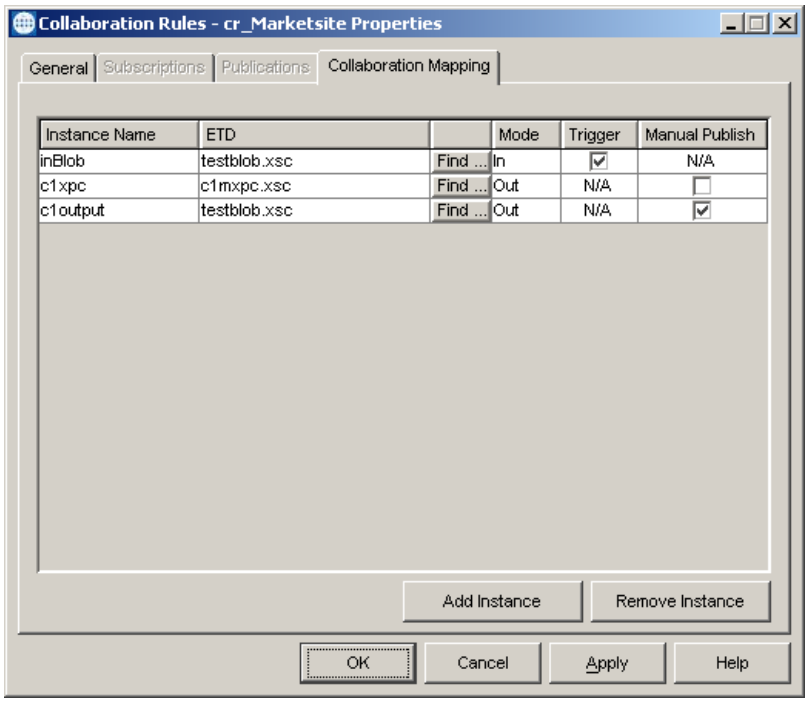
```
EGate.traceln(EGate.TRACE_EWAY,EGate.TRACE_EVENT_TRACE,"Response
String Set")
```

8 "**send output**" is created by dragging the send, located under c1output, to the Rule dialog box.

cr_Marketsite Collaboration Rule Mapping

The Collaboration Mapping associated with the cr_Marketsite Collaboration Rules is as displayed in Figure 22.

Figure 22 cr_Marketsite Collaboration Mapping



4.3.5. The buyerorderxpcftp Sample Schema

The buyerorderxpcftp sample schema demonstrates the use of the Commerce One e*Way in implementing FTP support for e*Gate to interface with the Commerce One XPC installed on another machine (as the counterpart for the buyerorderxpc sample schema for the simple buyer case). This schema also relies on Batch e*Way, File e*Way, and a running FTP server on the XPC machine.

As the counterpart for the buyerorderxpc sample schema for the simple buyer, this sample demonstrates FTP support for e*Gate to interface with Commerce One XPC installed on another machine.

There are two scenarios to consider:

- The user must copy all of the configuration default.props files in all of the subdirectories, corresponding to all of the XPC inbound Document Service and outbound Timed Services for:

  ```
  <rootdir>:\commerceone\Xpc\runtime\servers\defaultserver\config\service
  ```

  It is preferable to copy to a corresponding local directory in the machine for which e*Gate is installed.

- If possible, the user can make use of a Win2K network-mounted network drive (or NFS mount point in the case of Unix) capability to map the service configuration default.prop files on the XPC machine for e*Gate to access these files. The "XPC Config Root" entry for the associated e*Way Connection could be useful for the mapping.

  In the sample JCS, it is also assumed that a FTP server is running on the XPC machine and the FTP root path for the XPC machine is pointed to:

  ```
  <drive:>/commerceone/xpc/filestore
  ```

4.3.6. The supplierxpc Sample Schema

The supplierxpc sample schema demonstrates the use of the Commerce One e*Way in implementing the handling of inbound order, change order, and outbound order response / invoice / advance shipment notice XCBL documents. This schema relies on the Batch e*Way and File e*Way.

As the counterpart to the sample schema supplierorderxpc, which only handles order and order response XCBL documents, this schema demonstrates the handling of incoming order, outgoing invoice, outgoing advanced shipment notice (ASN), and outgoing order response.

Follow the directions for preparing the template file for outbound documents (see Table 7).

Table 7

Sample XCBL doc directories	File Name	Copy and Renamed
<rootdir>:\commerceone\Xpc\sample\xpc\instances\AdvanceShipNotice	AdvanceShipmentNotice_Sample.xml	AdvanceShipmentNotice_xxx.asn
<rootdir>:\commerceone\Xpc\sample\xpc\instances\Invoice	Invoice_Sample.xml	Invoice_xxx.invoice

The name-TPID lookup table in the map.txt file (in the <rootdir>\commerceone\xpc\tpid_map, e.g.) should be updated to provide a new mapping entry for the recipient (i.e. buyer) ID to the recipient TPID. As mentioned in

the XPC documentation, be sure to eliminate unnecessary blank spaces following the TPID.

Edit according to the following entry:

```
GetStringFromDocument.config=xPath=</InvoiceHeader/InvoiceParty/
BuyerParty/Party/PartyID/Identifier/Ident>
```

in the default.prop file for the invoice outbound service as in:

```
<rootdir>\commerceone\Xpc\runtime\servers\defaultserver\config\servic
e\TimedService.XPCTimedService.XPCInvoice30Outbound.1_0
```

Be sure to update (either the invoice_template file or accomplish in JCS to modify the XCBL data structure / ETD) the <Ident> field as follows:

```
...
 <InvoiceParty>
    <BuyerParty>
       <Party>
          <PartyID>
             <Identifier>
          ...
                <Ident>the_recipient_ID_as_in_the_map_file</Ident>
             </Identifier>
          </PartyID>
 ...
```

Similar measure shall be taken for the AdvanceShipmentNotice XCBL document.The ASN outbound service default.prop file is located in:

```
<rootdir>:\commerceone\Xpc\runtime\servers\defaultserver\config\servi
ce\TimedService.XPCTimedService.XPCAdvanceShipmentNotice30Outbound.1_
0
```

4.3.7. The buyerxpc Sample Schema

The buyerxpc sample schema demonstrates the handling of outbound orders, and accepting and archiving inbound ASN (Advance Shipment Notice) and inbound invoice XCBL documents. This schema relies on the Batch e*Way and File e*Way.

As the counterpart to the sample schema buyerorderxpc, which only handles order and order response XCBL documents, this schema demonstrates the handling of not just outbound orders, but also outbound change orders. The schema also accepts and archives inbound ASN and inbound invoice XCBL documents.

In order to send an order, change the order XCBL doc to a specific supplier.

The name-TPID lookup table in the map.txt file (in the < rootdir>\commerceone\xpc \tpid_map, e.g.) should be updated to provide a new mapping entry for the recipient (i.e. supplier) ID to the recipient TPID. As mentioned in the XPC documentation, be sure to eliminate unnecessary blank spaces following the TPID.

Be sure to also update (either the Order_xxx.order file or accomplish in JCS to modify the XCBL data structure / ETD) the <Ident> field as follows:

```
...
  <SellerParty>
    <Party>
      <PartyID>
        <Identifier>
```

```
        ...
          <Ident>the_recipient_ID_as_in_the_map_file</Ident>
        ...
      </Identifier>
    </PartyID>
```

The user should copy the sample order temple file order_template.xml (unzipped from order_template.zip) to the following directory:

```
<rootdir>:\commerceone\Xpc\sample\xpc\instance\Order
```

and rename the file extension to (.order).

Follow the directions for preparing all the template files for outbound documents (see Table 8).

Table 8

Sample XCBL doc directories	File Name	Copy and Renamed
<rootdir>:\commerceone\Xpc\sample\xpc\instances\Order	order_template.xml (provided sample), different from the default Order_Sample.xml	Order_xxx.order

4.3.8. The supplierxpcsync Sample Schema

The supplierxpcsync sample schema demonstrates the use of the Commerce One e*Way in implementing the simple handling of inbound and outbound XCBL synchronous documents (price check, order status, and availability check). This schema relies on e*Gate JMS and the File e*Way.

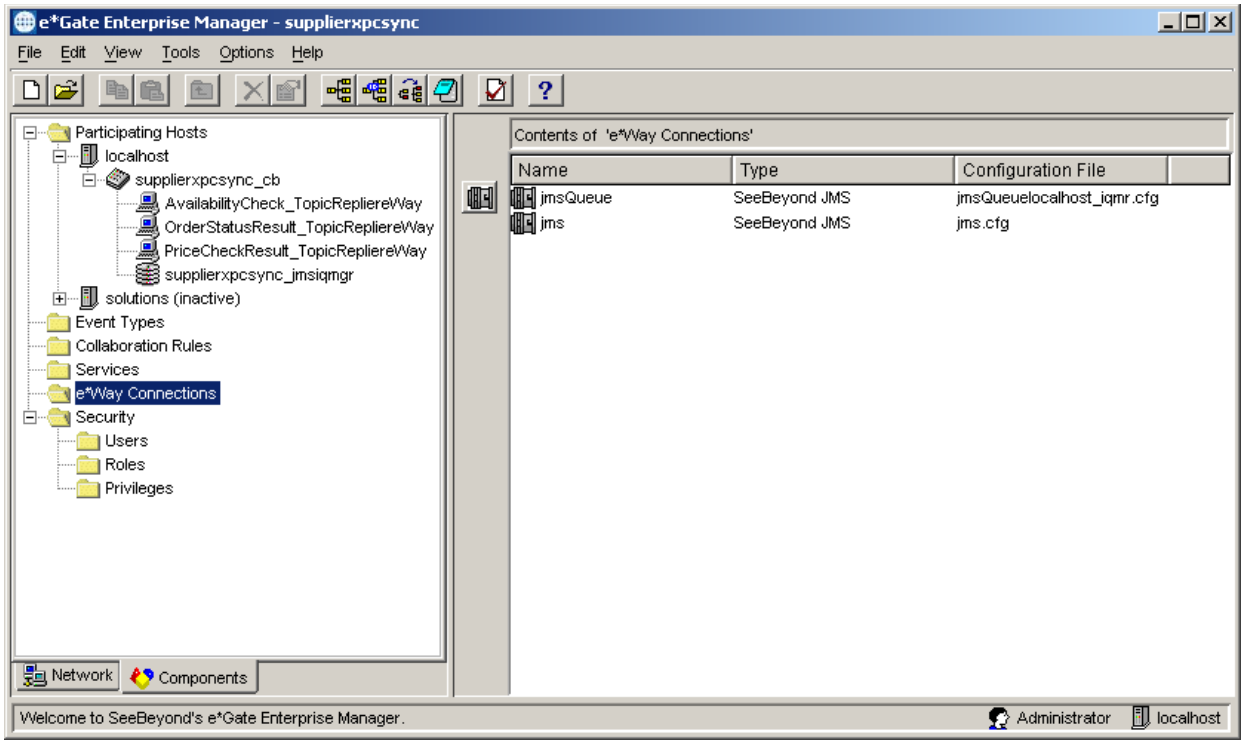
After installing the sample schema, it must be configured before running.

Table 9 Contents of the supplierxpcsync.zip file

Directory	File(s)
	supplierxpcsync.ctl supplierxpcsync.exp
supplierxpcsync\runtime\collaboration_rules	supplierxpcsync AvailabilityCheck_ReplyCollab.class AvailabilityCheck_ReplyCollab.ctl AvailabilityCheck_ReplyCollab.java AvailabilityCheck_ReplyCollab.xpr AvailabilityCheck_ReplyCollab.xts AvailabilityCheck_ReplyCollabBase.class OrderStatusResult_ReplyCollab.class OrderStatusResult_ReplyCollab.ctl OrderStatusResult_ReplyCollab.java OrderStatusResult_ReplyCollab.xpr OrderStatusResult_ReplyCollab.xts OrderStatusResult_ReplyCollabBase.class PriceCheckResult_ReplyCollab.class PriceCheckResult_ReplyCollab.ctl PriceCheckResult_ReplyCollab.java PriceCheckResult_ReplyCollab.xpr PriceCheckResult_ReplyCollab.xts PriceCheckResult_ReplyCollabBase.class
supplierxpcsync\runtime\configs\messageservice\	jms.cfg jms.sc jmsQueuelocalhost_iqmr.cfg jmsQueuelocalhost_iqmr.sc
supplierxpcsync\runtime\configs\stcewfile\	AvailabilityCheck_TopicRepliereWay.cfg AvailabilityCheck_TopicRepliereWay.sc PriceCheckResult_TopicRepliereWay.cfg PriceCheckResult_TopicRepliereWay.sc TopicReplieреWay.cfg TopicReplieреWay.sc
supplierxpcsync\runtime\configs\stcmsagent	localhost_iqmgr.cfg localhost_iqmgr.sc
supplierxpcsync\runtime\etd\	root.jar root.ssc root.xsc
supplierxpcsync\sandbox\collaboration_rules\	OrderStatusResult_ReplyCollab.xpr OrderStatusResult_ReplyCollab.xts

Figure 23 shows the Components view of the SupplierOrder Sample schema.

Figure 23 supplierxpcsync Sample Schema



Configuring the supplierxpcsync Sample

Once the sample has been successfully imported into e*Gate, the user must configure it to correspond to the specific systems. The following items should be examined:

- Each of the configuration files associated with the three e*Ways must be configured, as needed, saved, and promoted to runtime. Specifically, the following parameters must be addressed:
  - The e*Way Connection configuration must be adjusted to suit the systems involved.
  - Drive letter/prefix, see **"XPC Config Root" on page 30**
  - Path for XPC Service use, see **"Default Property File Path" on page 30**
  - Additional XPC processing, xCBL 3.0 or xCBL 1.0, see **"Soxtype Namespace Processing Instruction" on page 31**
- The File e*Way eater (simulating backend data sink) directory and file name for the incoming XCBL synchronous document (price check, order status, and availability check) should be tailored accordingly.
- Document support samples must be configured for the supplierxpcsync schema. See **Configuring the Synchronous Document Support Samples for Commerce One XPC.** on page 19 for directions.

JMS Considerations

The supplierxpcsync Sample utilizes the e*Gate Java Message Service (JMS). To enable the client system to communicate with the e*Gate API Kit, you must do the following:

- The JMS Topic/Queue names and the e*Gate Event Types names must coincide.
- The individual writing any external JMS code must know the expected data format, the name of the Topic/Queue, and the name of host and port number of the JMS server.
- The methods used must correspond to the expected data format. For a list of e*Gate supported Java classes, interfaces and methods, please see e*Gate API Kit User's Guide, supported libraries for the e*Gate Message Service.
- The client code samples provided are intended to work directly with the sample schema provided. These are only samples created as a demonstration of possible behavior.

4.3.9. Order_Template

The Order_Template.zip file contains Order_Template.xml file. The first few lines of which appear below:

```
<?soxtype urn:x-commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0?>
<?import urn:x-commerceone:document:com:commerceone:XCBL30:XCBL30.sox$1.0?>
<Order>
   <OrderHeader>
      <OrderNumber>
         <BuyerOrderNumber>2001-1116-3</BuyerOrderNumber>
         <SellerOrderNumber>2001-1116F-3a</SellerOrderNumber>
         <ListOfMessageID>
            <MessageID>
               <IDNumber>kbrcymciuk</IDNumber>
               <IDAssignedBy>
                  <IDAssignedByCoded>Non-ResidentBeneficiary</IDAssignedByCoded>
                  <IDAssignedByCodedOther>itzihljne</IDAssignedByCodedOther>
               </IDAssignedBy>
               <IDAssignedDate>20001215T09:52:25</IDAssignedDate>
            </MessageID>
            <MessageID>
               <IDNumber>xtmxc</IDNumber>
               <IDAssignedBy>
                  <IDAssignedByCoded>SubsidiaryDivision</IDAssignedByCoded>
                  <IDAssignedByCodedOther>zxsuvby</IDAssignedByCodedOther>
               </IDAssignedBy>
               <IDAssignedDate>20001215T09:52:25</IDAssignedDate>
            </MessageID>
         </ListOfMessageID>
      </OrderNumber>
```

4.3.10. Supporting Documents

The following documents are designed to work in conjunction with the *e*Way Intelligent Adapter for Commerce One MarketSite User's Guide* and to provide additional information that may prove useful.

SeeBeyond ICAN Suite Installation Guide.

*e*Gate Integrator System Administration and Operations Guide.*

*e*Gate Integrator User's Guide.*

*e*Gate API Kit User's Guide.*

SeeBeyond JMS Intelligent Queue User's Guide

*SeeBeyond Master Index (SeeBeyond_Index.pdx; refer to e*Gate Integrator User's Guide for instructions on how to access).*

*README.txt files on the e*Gate installation CD ROM.*

Chapter 5

Commerce One MarketSite e*Way Methods

The Commerce One MarketSite e*Way methods fall into the following categories:

5.1 com.stc.eways.c1mxpc.C1MXPC

The hierarchy for all packages pertaining to the Commerce One MarketSite e*Way are as follows:

```
class com.stc.eways.c1mxpc.C1MXPC
class java.lang.Object
    class com.stc.eways.c1mxpc.C1MXPCConfigHelper.FileStoreConfigClass
    class com.stc.eways.c1mxpc.C1MXPCConnector
    class com.stc.eways.c1mxpc.C1MXPCContext
    class com.stc.eways.c1mxpc.C1MXPCDefs
    class com.stc.eways.c1mxpc.C1MXPCDocTransmitter
    class com.stc.eways.c1mxpc.C1MXPCTester
    class java.util.Dictionary
        class java.util.Hashatable (implements java.lang.Cloneable, java.util.Map,
java.ioializable)
            class java.util.Properties
                class com.stc.eways.c1mxpc.FileProperties
    class com.stc.jcsre.SimpleETDImpl (implements com.stc.jcsre.ETD)
        class com.stc.eways.c1mxpc.C1MXPCConfigHelper
        class com.stc.eways.c1mxpc.eGateRequestor
                class com.stc.eways.c1mxpc.eGateRequestor.eGateRequestorException
```

The following classes are included in this document:

5.1.1. Class C1MXP

The C1MXP class contains the following methods:

C1MXPC

Syntax

```
public C1MXPC()
```

Description

Constructor.

getDestination

Syntax

```
public java.lang.String getDestination()
```

Description

getDestination obtains the current value for MarketSite destination.

Parameters

None.

Return Values

java.lang.String
Returns a string containing the MarketSite destination value.

Additional information

This function can be accessed via the Collaboration. Refer to the CommerceOne XPC documentation for details on valid values for destinations.

getDocumentType

Syntax

```
public java.lang.String getDocumentType()
```

Description

getDocumentType returns the document type of the document being sent to MarketSite. The document is passed as an XML string.

Parameters

None.

Return Values

java.lang.String
Returns the document type.

getPassword

Syntax

```
public java.lang.String getPassword()
```

Description

getPassword obtains the MarketSite password used for authentication.

Parameters

None.

Return Values

java.lang.String
Returns the MarketSite password (unencrypted).

getRecipient

Syntax

```
public java.lang.String getRecipient()
```

Description

getRecipient obtains the current value for the MarketSite recipient. Refer to the CommerceOne XPC documentation for details on valid values for recipients.

Parameters

None.

Return Values

java.lang.String
Returns the current value for the MarketSite recipient.

getSender

Syntax

```
public java.lang.String getSender()
```

Description

getSender obtains the current value for the MarketSite sender. Refer to the CommerceOne XPC documentation for details on valid values for senders.

Parameters

None.

Return Values

java.lang.String
Returns the current value for the MarketSite sender.

getSyncResponseString

Syntax

```
public java.lang.String getSyncResponseString()
```

Description

getSyncResponseString returns the last response string received from MarketSite from a synchronous transmission of documents, usually the originated by the sendToMarketSite method.

Parameters

None.

Return Values

java.lang.String
Returns the response string from synchronous transmissions (syncResponseString).

getUserName

Syntax

```
public java.lang.String getUserName()
```

Description

getUserName returns the MarketSite user name used for authentication with MarketSite.

Parameters

None.

Return Values

java.lang.String
Returns the MarketSite user name (username).

getXmlString

Syntax

```
public java.lang.String getXmlString()
```

Description

getXmlString returns the current string value assigned to be sent to MarketSite using the **sendToMarketSite** method.

Parameters

None.

Return Values

java.lang.String
Returns the string to be transmitted to MarketSite (xmlString).

initialize

Syntax

```
public void initialize(com.stc.common.collabService.JCollabController
cntrCollab, java.lang.String key, int mode)
```

Description

initialize reads the configuration information from the file, initializing the transmitter context (called by the Collaboration service).

Parameters

Name	Type	Description
cntrCollab	com.stc.common.collabService.JCollabcontroller	The Java Collaboration Controller object.
key	String	
mode	int	

Return Values

void

Throws

com.stc.common.collabService.CollabConnException
com.stc.common.collabService.CollabDataException

reset

Syntax

```
public boolean reset()
```

Description

reset clears the settings for document type, recipient, sender, destination, xmlString, and syncResponseString.

Parameters

None.

Return Values

boolean
Returns true if successful; otherwise, returns false.

sendToMarketSite

Syntax

```
public void sendToMarketSite()
public void sendToMarketSite(byte[] outEvent)
public void sendToMarketSite(java.lang.String inXmlString)
```

Description

sendToMarketSite is called from the Collaboration to send the current xmlString value (first instance) or the passed byte array (second instance), or the passed xmlString value, to MarketSite. The xmlString must be a valid xCBL document.

Parameters

Name	Type	Description
outEvent	byte[ ]	The xCBL document as a byte array.
inXmlString	java.lang.String	The xCBL document string to be passed to MarketSite.

Return Values

void

Throws

com.stc.common.collabService.CollabConnException
com.stc.common.collabService.CollabDataException
com.stc.common.collabService.CollabResendException

setDestination

Syntax

```
public void setDestination(java.lang.String destination)
```

Description

setDestination designates the MarketSite destination TPID (Trading Partner ID). Refer to the CommerceOne XPC documentation for details on valid destination values.

Parameters

Name	Type	Description
destination	java.lang.String	A valid destination string (MPID).

Return Values

void

setDocumentType

Syntax

```
public void setDocumentType(java.lang.String documentType)
```

Description

setDocumentType specifies the document type being sent to MarketSite, passed as an xmlString.

Parameters

Name	Type	Description
documentType	java.lang.String	A valid document type string.

Return Values

void

setPassword

Syntax

```
public void setPassword(java.lang.String password)
```

Description

setPassword designates the MarketSite password, called from the Collaboration. Refer to the CommerceOne XPC documentation for details on configuring XPC authentication.

Parameters

Name	Type	Description
password	java.lang.String	The unencrypted password used for authentication with MarketSite.

Return Values

void

setRecipient

Syntax

```
public void setRecipient(java.lang.String recipient)
```

Description

setRecipient designates the MarketSite recipient TPID, called from the Collaboration. Refer to the CommerceOne XPC documentation for details on valid recipient values.

Parameters

Name	Type	Description
recipient	java.lang.String	A valid recipient string (MPID).

Return Values

void

setSender

Syntax

```
public void setSender(java.lang.String sender)
```

Description

setSender designates the MarketSite sender TPID, called from the Collaboration. Refer to the CommerceOne XPC documentation for details on valid values for sender.

Parameters

Name	Type	Description
sender	java.lang.String	A valid sender string (MPID).

Return Values

void

setUsername

Syntax

```
public void setUsername(java.lang.String username)
```

Description

setUsername designates the MarketSite username from the Collaboration. Refer to the CommerceOne XPC documentation for details on configuring XPC authentication.

Parameters

Name	Type	Description
username	java.lang.String	The username required by MarketSite for authentication.

Return Values

void

---

setXmlString

Syntax

```
public void setXmlString(java.lang.String xmlString)
```

Description

setXmlString specifies the xmlString to be transmitted to MarketSite. Refer to the CommerceOne XPC documentation for details on valid xCBL documents that can be transmitted to MarketSite.

Parameters

Name	Type	Description
xmlString	java.lang.String	An xmlString to be sent to MarketSite.

Return Values

void

5.1.2. Class C1MXPCConfigHelper

The C1MXPCConfigHelper class contains the following methods:

C1MXPCConfigHelper

Syntax

```
public C1MXPCConfigHelper()
```

Description

Constructor.

Parameters

None.

getDocFileName

Syntax

```
public java.lang.String getDocFileName()
```

Description

getDocFileName obtains the filename of the document to be sent.

Parameters

None.

Return Values

java.lang.String
Returns the filename, relative to the RootDirectory of the file to be sent to MarketSite.

getErrorHandlerConfig

Syntax

```
public C1MXPCConfigHelper.ErrorHandlerConfigClass
getErrorHandlerConfig()
```

Description

getErrorHandlerConfig obtains the ErrorHandlerConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.ErrorHandlerConfigClass
Returns the ErrorHandlerConfigClass object.

getErrorStoreConfig

Syntax

```
public C1MXPCConfigHelper.ErrorStoreConfigClass getErrorStoreConfig()
```

Description

getErrorStoreConfig obtains the ErrorStoreConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.ErrorStoreConfigClass
Returns the ErrorStoreConfigClass object.

getFileStoreConfig

Syntax

```
public C1MXPCConfigHelper.FileStoreConfigClass getFileStoreConfig()
```

Description

getFileStoreConfig obtains the FileStoreConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.FileStoreConfigClass
Returns the FileStoreConfigClass object.

getOrderStoreConfig

Syntax

```
public C1MXPCConfigHelper.OrderStoreConfigClass getOrderStoreConfig()
```

Description

getOrderStoreConfig obtains the OrderStoreConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.OrderStoreConfigClass
Returns the OrderStoreConfigClass object.

getOriginalMessageStoreConfig

Syntax

```
public C1MXPCConfigHelper.OriginalMessageStoreConfigClass
getOriginalMessageStoreConfig()
```

Description

getOriginalMessageStoreConfig obtains the OriginalMessageStoreConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.OriginalMessageStoreConfigClass
Returns the OriginalMessageStoreConfigClass object.

getPlanningScheduleStoreConfig

Syntax

```
public C1MXPCConfigHelper.PlanningScheduleStoreConfigClass
getPlanningScheduleStoreConfig()
```

Description

getPlanningScheduleStoreConfig obtains the PlanningScheduleStoreConfigClass object.

Parameters

None.

Return Values

C1MXPCConfigHelper.PlanningScheduleStoreConfigClass
Returns the PlanningScheduleStoreConfigClass object.

getTransferMode

Syntax

```
public java.lang.String getTransferMode()
```

Description

getTransferMode obtains the transfer mode information.

Parameters

None.

Return Values

java.lang.String
Returns the mode for transfer, indicating “inbound” or “outbound”.

loadXPCServicesConfig

Syntax

```
public void loadXPCServicesConfig(java.lang.String propsFileName)
```

Description

loadXPCServicesConfig is called by external to load the XPC service configuration form the associated properties file (normally called default.prop).

Parameters

Name	Type	Description
propsFileName	java.lang.String	The name of XPC service properties file.

Return Values

void

main

Syntax

```
public static void main(java.lang.String[] args)
```

Description

main is used for testing only (internal or command line).

Parameters

Name	Type	Description
args	java.lang.String[ ]	An array of arguments

Return Values

void

Throws

java.lang.Exception

setDocFileName

Syntax

```
public void setDocFileName(java.lang.String filename)
```

Description

setDocFileName is used to set the filename of the document to be sent.

Parameters

Name	Type	Description
filename	java.lang.String	The name of the file to send to MarketSite.

Return Values

void

setErrorStoreConfig

Syntax

```
public void
setErrorStoreConfig(C1MXPCConfigHelper.ErrorStoreConfigClass
newErrorStoreConfigClass)
```

Description

setErrorStoreConfig is called by external to set the ErrorStoreConfigClass object.

Parameters

Name	Type	Description
newErrorStoreConfigClass	C1MXPCConfigHelper.ErrorStoreConfigClass	An ErrorStoreConfigClass object to be set.

Return Values

void

setFileStoreConfig

Syntax

```
public void
setFileStoreConfig(C1MXPCConfigHelper.FileStoreConfigClass
newFileStoreConfig)
```

Description

setFileStoreConfig is called by external to set the FileStoreConfigClass object.

Parameters

Name	Type	Description
newFileStoreConfigClass	C1MXPCConfigHelper.FileStoreConfigClass	An FileStoreConfigClass object to be set.

Return Values

void

setOrderStoreConfig

Syntax

```
public void
setOrderStoreConfig(C1MXPCConfigHelper.OrderStoreConfigClass
newOrderStoreConfigClass)
```

Description

setOrderStoreConfig is called by external to set the OrderStoreConfigClass object.

Parameters

Name	Type	Description
newOrderStoreConfigClass	C1MXPCConfigHelper.OrderStoreConfigClass	An OrderStoreConfigClass object to be set.

Return Values

void

setOriginalMessageStoreConfig

Syntax

```
public void
setOriginalMessageStoreConfig(C1MXPCConfigHelper.OriginalMessageStore
ConfigClass newOriginalMessageStoreConfigClass)
```

Description

setOriginalMessageStoreConfig is called by external to set the OriginalMessageStoreConfigClass object.

Parameters

Name	Type	Description
newOriginalMessageStoreConfigClass	C1MXPCConfigHelper.OriginalMessageStoreConfigClass	An OriginalMessageStore ConfigClass object to be set.

Return Values

void

setPlanningScheduleStoreConfig

Syntax

```
public void
setPlanningScheduleStoreConfig(C1MXPCConfigHelper.PlanningScheduleSto
reConfigClass newPlanningScheduleStoreConfigClass)
```

Description

setPlanningScheduleStoreConfig is called by external to set the PlanningScheduleStoreConfigClass object.

Parameters

Name	Type	Description
newPlanningScheduleStore ConfigClass	C1MXPCConfigHelper.Plannin gScheduleStoreConfigClass	A PlanningScheduleStore ConfigClass object to be set.

Return Values

void

setTransferMode

Syntax

```
public void setTransferMode(java.lang.String mode)
```

Description

setTransferMode is called by external to set transfer mode.

Parameters

Name	Type	Description
mode	java.lang.String	Either inbound or outbound.

Return Values

void

5.1.3. Class FileProperties

The C1MXPCFileProperties class contains the following methods:

FileProperties on page 94

close on page 94

load on page 94

save on page 95

FileProperties

Syntax

```
public FileProperties(java.lang.String loadsaveFileName)

public FileProperties(java.lang.String loadsaveFileName,
java.util.Properties defProp)
```

Description

Constructor. The first instance, constructs a FileProperties object given a fileName. The second instance, constructs a FileProperties object given a fileName and a list of default properties.

Parameters

Name	Type	Description
loadsaveFileName	java.lang.String	The name of the file.
defProp	java.util.Properties	The default properties.

Return Values

void

Throws

java.io.IOException

close

Syntax

```
public void close()
```

Description

close .

Parameters

None.

Return Values

void

load

Syntax

```
public void load()
```

Description

load is used to load the properties from the saved filename. If that fails, retry, including the .properties extension.

Parameters

None.

Return Values

void

Throws

java.io.IOException

---

save

Syntax

```
public void save()
```

Description

save is used to save the properties for loading at a later time.

Parameters

None.

Return Values

void

Throws

java.io.IOException

5.1.4. Class eGateRequestor

The eGateRequestor class contains the following methods:

---

eGateRequestor

Syntax

```
public eGateRequestor()
```

Description

Constructor.

Parameters

None.

setJMSTopicName

Syntax

```
public void setJMSTopicName(java.lang.String topicString)
```

Description

setJMSTopicName sets the JMS topic to which the requestor publishes messages.

Parameters

Name	Type	Description
topicString	java.lang.String	The JMS topic name to publish to.

Return Values

void

getJMSTopicName

Syntax

```
public java.lang.String getJMSTopicName()
```

Description

getJMSTopicName gets the JMS topic to which the requestor publish messages.

Parameters

None.

Return Values

java.lang.String
Returns the topicString.

setJMSHostName

Syntax

```
public void setJMSHostName(java.lang.String hostName)
```

Description

setJMSHostName sets the JMS server host where JMS messages are sent and received.

Parameters

Name	Type	Description
hostName	java.lang.String	The JMS server host name to be set.

Return Values

void

getJMSHostName

Syntax

```
public java.lang.String getJMSHostName()
```

Description

getJMSHostName gets the JMS server host where JMS messages are sent and received.

Parameters

None.

Return Values

java.lang.String
Returns the hostName.

setJMSPort

Syntax

```
public void setJMSPort(int port)
```

Description

setJMSPort sets the JMS server port where JMS messages are sent and received.

Parameters

Name	Type	Description
port	int	The JMS server port number to be set.

Return Values

void

getJMSPort

Syntax

```
public int getJMSPort()
```

Description

getJMSPort gets the JMS server port where JMS messages are sent and received.

Parameters

None.

Return Values

int
Returns the port number.

initializeEGateJMS

Syntax

```
public void initializeEGateJMS()
```

Description

initializeEGateJMS initializes the JMS settings to be able to perform the requestor function. This includes creating the connection and topic factories, and the actual connection and topic used.

Parameters

None.

Return Values

void

Throws

eGateRequestor.eGateRequestorException

publishToEGate

Syntax

```
public java.lang.String publishToEGate(java.lang.String
messageToSend)
```

Description

publishToEGate publishes the messageToSend string to the eGate JMS server.

Parameters

Name	Type	Description
messageToSend	java.lang.String	Any messages to send to e*Gate via JMS (for example, xCBL, XML string).

Return Values

java.lang.String
Returns the replyMessage.

closeEGateJMS

Syntax

```
public void closeEGateJMS()
```

Description

closeEGateJMS closes the connection to the eGate JMS server.

Parameters

None.

Return Values

void

Throws

eGateRequestor.eGateRequestorException

main

Syntax

```
public static void main(java.lang.String[] args)
```

Description

eGateRequestor can be tested as a stand-alone program via the command line: java com.stc.eways.c1mxpc.eGateRequestor topicName numOfMsgs

Parameters

Name	Type	Description
args	java.lang.String[]	1. Publisher topic name. 2. Number of messages to send.

Return Values

static void

onException

Syntax

```
public void onException(com.stc.eways.c1mxpc.JMSException e)
```

Description

onException is called when an exception occurs while waiting for a response.

Parameters

Name	Type	Description
e	`com.stc.eways.c1mxpc.JMSException`	General exception error thrown when there is a JMS related error.

Return Values

void

5.1.5. Class eGateRequestor.eGateRequestorException

The eGateRequestor.eGateRequestorException class contains the following methods:

eGateRequestor.eGateRequestorException on page 100

---

eGateRequestor.eGateRequestorException

Syntax

```
public eGateRequestor.eGateRequestorException()
```

```
public eGateRequestor.eGateRequestorException(java.lang.String msg)
```

Description

Constructor. **eGateRequestor.eGateRequestorException** extends java.lang.Exception, and implements java.io.Serializable.

Parameters

Name	Type	Description
msg	java.lang.String	Exception message.

Return Values

None.

Index

N

O

P

R

S

T

W

X