

SeeBeyond ICAN Suite

e*Way Intelligent Adapter for PeopleSoft HTTP User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version



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, and e*Way are the registered trademarks of SeeBeyond Technology Corporation in the United States and select foreign countries; the SeeBeyond logo, e*Insight, and e*Xchange 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.

© 2001–2004 by 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 This work is protected as an unpublished work under the copyright laws.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 20050406025655.

Contents

Preface	8
Intended Reader	8
Organization	8
Nomenclature	9
Online Use	9
Writing Conventions	9
Additional Documentation	10
<hr/>	
Chapter 1	
Introduction	11
Overview	11
PeopleSoft 8.1	11
PeopleSoft 8.4	12
e*Way Availability	13
<hr/>	
Chapter 2	
Installation	14
System Requirements	14
Environment Configuration	14
External System Requirements	15
External Configuration Requirements	15
Installing the e*Way	16
Windows Systems	16
Installation Procedure	16
Subdirectories and Files	18
UNIX Systems	19
Installation Procedure	19
Subdirectories and Files	20
Installing the MUX Subscription Handler	21
Windows Systems	21

UNIX Systems	24
Optional Example Files	26
Installation Procedure	26
Subdirectories and Files	27

Chapter 3

PeopleSoft 8 Setup	29
Overview	29
Configuring for Publication	30
Creating PeopleSoft 8 Message Node for the e*Way	30
PeopleSoft 8.4/PeopleTools 8.42	30
PeopleSoft 8.1/PeopleTools 8.13	30
Activating the Message Definition for Publication	34
PeopleSoft 8 Message Definitions List	34
Defining Message Channel Routing Rules	37
Configuring the Message Channel	37
Defining Routing Directions for Message Nodes	41
Adding the PeopleSoft 8 Subscription Handler	43
Configuring for Subscription	46
PeopleSoft 8/PeopleTools 8.13	46
Creating a MUX e*Way Message Node	46
Activating the Message Definition for Subscription	46
Defining the Message Channel Routing Rules	46
Adding the SeeBeyond MUX Subscription Handler	46
Configuring the SeeBeyond MUX Subscription Handler	49
PeopleSoft 8.4/PeopleTools 8.42	49

Chapter 4

System Implementation	51
Overview	51
Pre-Implementation Tasks	51
Implementation Sequence	52
Viewing e*Gate Components	52
Creating a Schema	53
Creating Event Types	54
Generating Event Type Definitions	55
Generating DTDs from PeopleSoft	55
Generating and Publishing an XML Test Message	56
Extracting and Viewing the XML Test Message	62
Generating a DTD for the XML File	67
Generating an ETD from the DTD	70
Using the DTD Builder	70
Assigning ETDs to Event Types	76

Defining Collaborations	77
The Java Collaboration Rules Editor	77
Creating Intelligent Queues	78
Using the e*Way	79
Publishing to PeopleSoft	79
XML Messages	79
Compressing the XML Message	80
Subscribing to PeopleSoft 8.1	81
Subscribing to PeopleSoft 8.4	82
Sample Schema	83
Publishing to PeopleSoft	83
Components	83
Operation	84
Subscribing to PeopleSoft 8.1	86
Components	86
Operation	86
Subscribing to PeopleSoft 8.4	88
Components	88
Operation	88

Chapter 5

e*Way Setup	89
Overview	89
Setting Up the e*Way	90
Creating the e*Way	90
Modifying e*Way Properties	91
Configuring the Inbound e*Way	92
Configuring the Outbound e*Way	93
Changing the User Name	94
Setting Startup Options or Schedules	94
Activating or Modifying Logging Options	96
Activating or Modifying Monitoring Thresholds	97
Creating e*Way Connections	98
Using the e*Way Editor	101
Section and Parameter Controls	102
Parameter Configuration Controls	102
Command-line Configuration	103
Getting Help	103
Troubleshooting the e*Way	104
Configuration Problems	104
System-related Problems	105

Chapter 6

Operational Overview	106
Introduction	106
Multi-Mode e*Way	107
Collaborations and Event Type Definitions	108
Java Collaboration Service	110
e*Way Connections	111
Establishing Connections	111

Chapter 7

Configuration Parameters	113
Overview	113
e*Gate to PeopleSoft	113
Multi-Mode e*Way	113
e*Way Connections	113
PeopleSoft to e*Gate	114
e*Gate API Kit	114
Multi-Mode e*Way	115
JVM Settings	115
General Settings	120
e*Way Connection	121
Connector	121
HTTP	122
Proxies	124
HTTP Authentication	126
SSL	127
ApplicationMessaging	132
e*Gate API Kit (Multiplexer)	137
General Settings	137

Chapter 8

Java Classes and Methods	140
Overview	140
Object Classes	141
PeopleSoft8AppMsg Class	142
PeopleSoft8AppMsgAPI Class	143
PeopleSoft8AppMsgConnector Class	144
PeopleSoft8AppMsgException Class	145
PeopleSoft8AppMsgResponse Class	146
PeopleSoft8AppMsgTester Class	147

Contents

Base64Utils Class	148
Constructors	149
Methods	154

Appendix A

MUX Subscription Handler **188**

Object Classes	188
Entry Class	188
MuxHandlerConstants Class	188
MuxHandlerEntry Class	189
AdministerMuxHandler Class	189
AdministerMuxHandlerAddMode Class	189
AdministerMuxHandlerDeleteMode Class	189
AdministerMuxHandlerEditMode Class	189
AdministerMuxHandlerError Class	189
MuxPublicationHandler Class	189
MuxHandler Class	190

Index **191**

Preface

This Preface contains information regarding the User's Guide itself.

P.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond™ e*Gate™ Integrator system, and have a working knowledge of:

- Operation and administration of the appropriate operating systems (see [Supported Operating Systems](#) on page 13)
- Windows-style GUI operations
- PeopleSoft concepts and operations
- Integrating PeopleSoft applications with external systems

P.2 Organization

This User's Guide is organized into two parts. The first part, consisting of Chapters 1-5, introduces the e*Way and describes the procedures for installing and setting up the e*Way, and implementing a working system incorporating the e*Way. This part should be of particular interest to a System Administrator or other user charged with the task of getting the system up and running.

The second part, consisting of Chapters 6-8, describes the details of e*Way operation and configuration, including descriptions of the API functions. This part should be of particular interest to a Developer involved in customizing the e*Way for a specific purpose. Information contained in this part that is necessary for the initial setup of the e*Way is cross-referenced in the first part of the guide, at the appropriate points in the procedures.

P.3 Nomenclature

Note that for purposes of brevity, the e*Way Intelligent Adapter for PeopleSoft HTTP is frequently referred to as the PeopleSoft HTTP e*Way, or simply the e*Way.

P.4 Online Use

This User's Guide is provided in Adobe Acrobat's Portable Document Format (PDF). As such, it can be printed out on any printer or viewed online. When viewing online, you can take advantage of the extensive hyperlinking imbedded in the document to navigate quickly throughout the Guide.

Hyperlinking is available in:

- The Table of Contents
- The Index
- Within the chapter text, indicated by **blue print**

Existence of a hyperlink *hotspot* is indicated when the hand cursor points to the text. Note that the hotspots in the Index are the *page numbers*, not the topics themselves. Returning to the spot you hyperlinked from is accomplished by right-clicking the mouse and selecting **Go To Previous View** on the resulting menu.

P.5 Writing Conventions

The writing conventions listed in this section are observed throughout this document.

Monospaced (Courier) Font

Computer code and text to be typed at the command line are set in Courier as shown below.

```
Configuration for BOB_Promotion
java -jar ValidationBuilder.jar
```

Variables within a command line are set in Courier italic as shown below.

```
stcregutl -rh host-name -un user-name -up password -sf
```

Bold Sans-serif Font

- User Input: Click **Apply** to save, or **OK** to save and close.
- File Names and Paths: In the **Open** field, type **D:\setup\setup.exe**.
- Parameter, Function, and Command Names: The default parameter **localhost** is normally only used for testing; the Monk function **iq-put** places an Event into an IQ.

P.6 Additional Documentation

- Many of the procedures included in this User's Guide are described in greater detail in the *e*Gate Integrator User's Guide*
- For more information on the Java Collaboration Service, see the *e*Gate Integrator Collaboration Services Reference*
- For additional information on the Multi-Mode e*Way, see the *Standard e*Way Intelligent Adapter User's Guide*
- For additional information on the Multiplexer e*Way, see the *e*Gate API Kit User's Guide*
- For detailed information regarding the DTD Builder or other components of the XML Toolkit, see the *XML Toolkit User's Guide*
- For information on Application Messaging and PeopleSoft 8 Integration Technology, please refer to the *PeopleSoft 8 PeopleTools* documentation

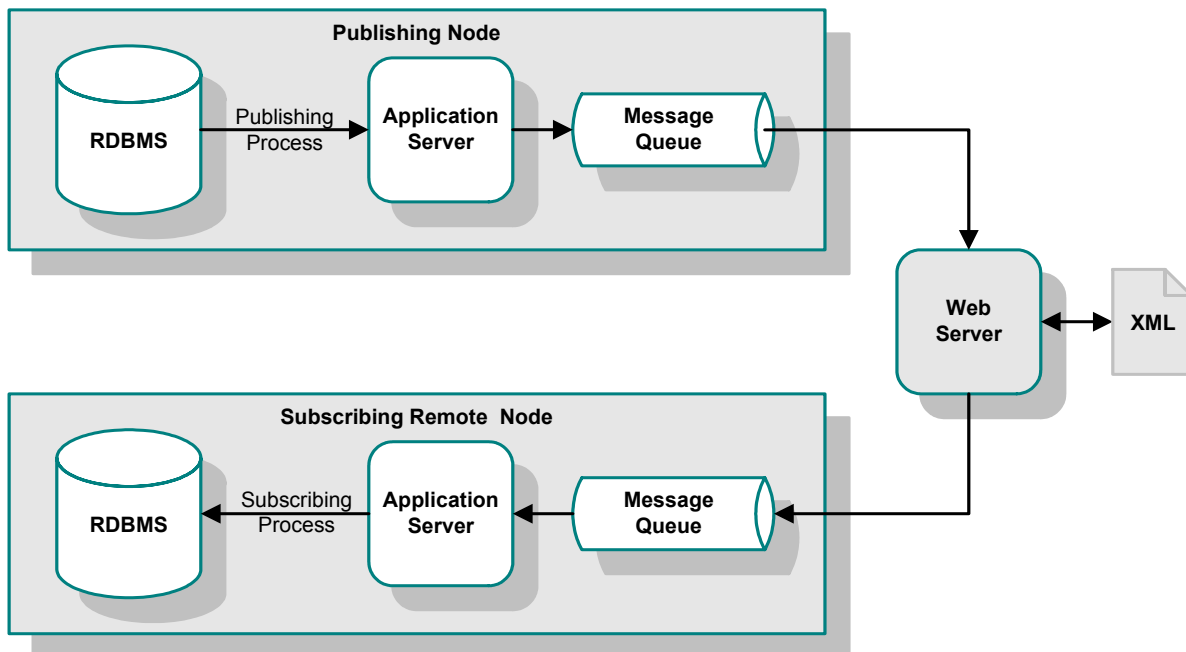
Introduction

1.1 Overview

The PeopleSoft HTTP e*Way provides a means of sending data to PeopleSoft 8 by using PeopleSoft 8 Application Messaging technology. The Application Messaging model allows for publication and subscription of XML messages using HTTP. This clean and flexible solution allows for implementation to be accomplished at the business level by means of XML messages.

Figure 1 diagrams the PeopleSoft 8 Application Messaging architecture.

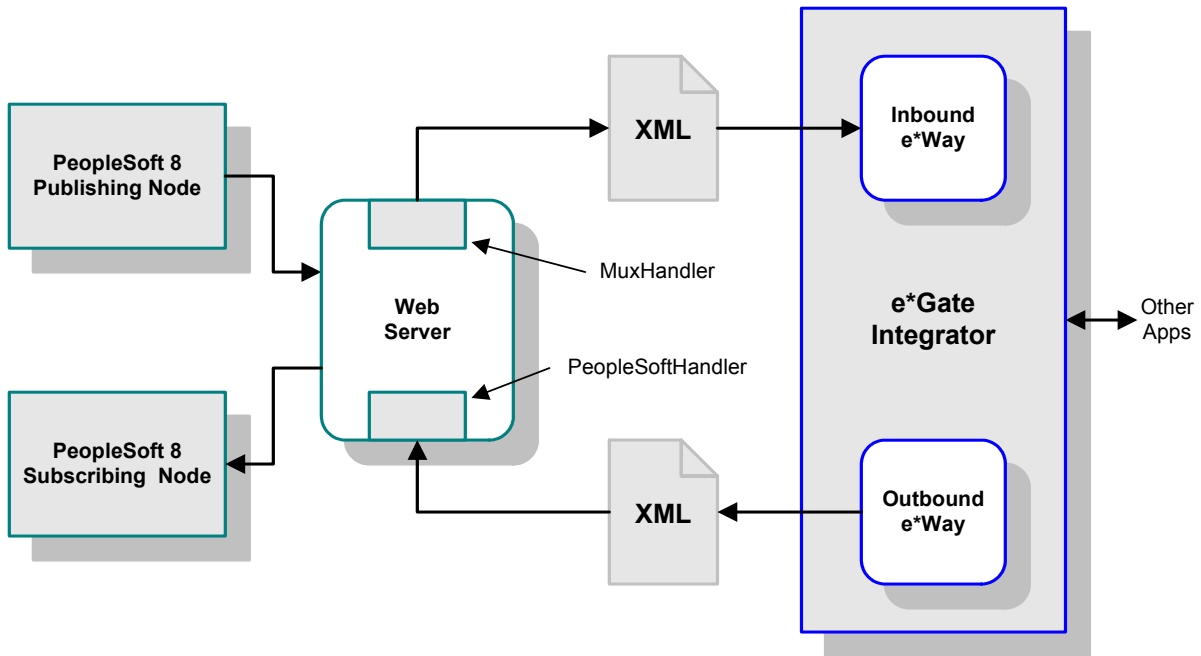
Figure 1 PeopleSoft 8 Application Messaging Architecture



1.1.1 PeopleSoft 8.1

Bidirectional data exchange with PeopleSoft 8.1 is provided by using the SeeBeyond Multi-Mode e*Way and SeeBeyond's customized PeopleSoft 8 MUX subscription handler classes. Figure 2 illustrates the integration architecture.

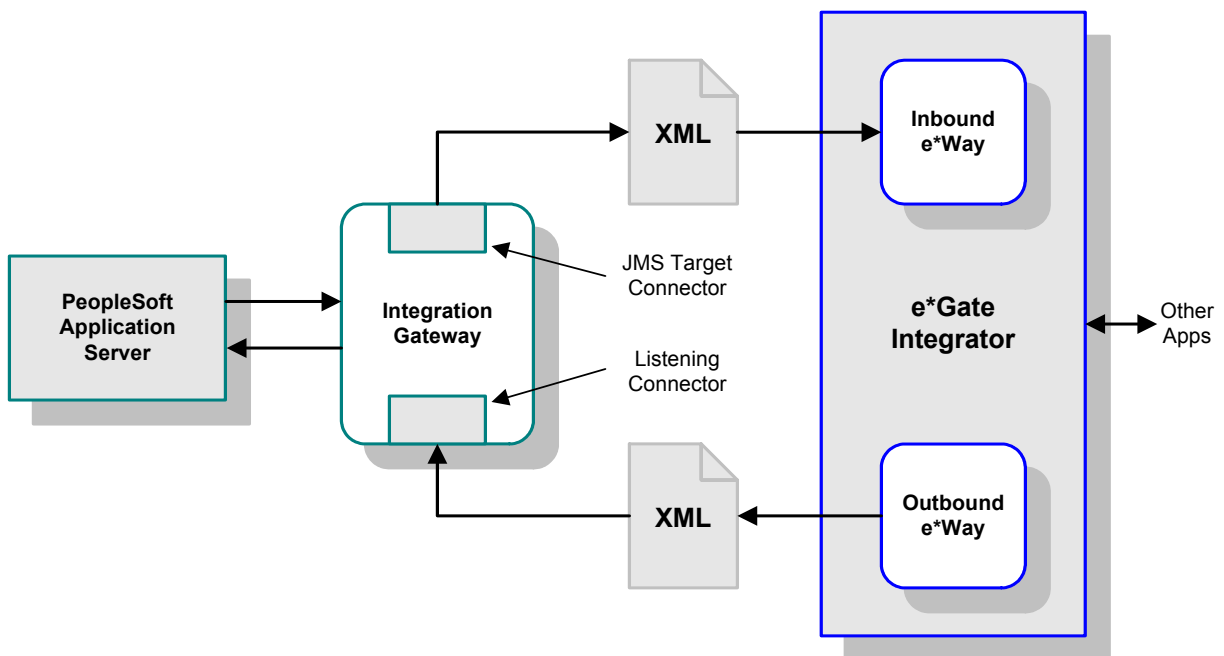
Figure 2 PeopleSoft 8.1 - e*Gate Integration



1.1.2 PeopleSoft 8.4

With PeopleSoft 8.4, the e*Way publishes to a PeopleSoft Listening Connector. For messages outbound from PeopleSoft 8.4, the MUX process is replaced by a JMS process.

Figure 3 PeopleSoft 8.4 - e*Gate Integration



1.2 Supported Operating Systems

The Java e*Way Intelligent Adapter for PeopleSoft HTTP currently supports the following combinations of and PeopleSoft system components.

- Windows 2000 and Windows Server 2003
- IBM AIX 5.1L
- Sun Solaris 8
- Japanese Windows 2000 and Windows Server 2003
- Japanese Sun Solaris 8

Note: *The e*Gate Schema Designer runs only on Windows.*

Installation

This chapter describes the requirements and procedures for installing the e*Way software. Procedures for implementing a working system, incorporating instances of the e*Way, are described in [Chapter 4](#).

***Note:** Please read the `readme.txt` file located in the `addons\ewpsoft8` directory on the installation CD-ROM for important information regarding this installation.*

2.1 System Requirements

To use the e*Way Intelligent Adapter for PeopleSoft HTTP, you need the following:

- 1 An e*Gate Participating Host.
- 2 A TCP/IP network connection.
- 3 Sufficient free disk space to accommodate e*Way files:
 - ♦ Approximately 4 MB on Windows systems
 - ♦ Approximately 34 MB on Solaris systems
 - ♦ Approximately 28 MB on AIX systems

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.

2.1.1 Environment Configuration

No changes are required to the Participating Host's operating environment to support this e*Way.

2.2 External System Requirements

The Java e*Way Intelligent Adapter for PeopleSoft HTTP requires the following external system components:

- PeopleSoft 8 with PeopleTools 8.13
- PeopleSoft 8.4 with PeopleTools 8.42
- For IBM AIX systems, a back-end Oracle 8.1.6 RDBMS

Note: PeopleSoft 8.4 is not supported on Solaris 2.6.

2.2.1 External Configuration Requirements

To publish XML messages to, or subscribe to XML messages from PeopleSoft 8, Message Nodes, Messages, Message Channels, and Subscription Handlers or JMS Target Connectors must be defined and configured within the PeopleSoft 8 environment. See [Chapter 3](#) for procedural details.

2.3 Installing the e*Way

2.3.1 Windows Systems

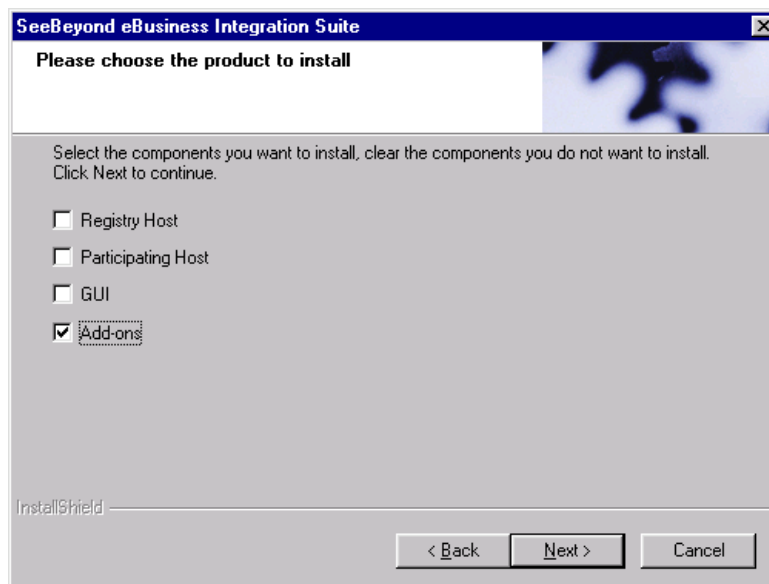
Installation Procedure

Note: *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond.*

To Install the e*Way on a Microsoft Windows System

- 1 Log in as an Administrator on the workstation on which you want to install the e*Way (you must have Administrator privileges to install this e*Way).
- 2 Exit all Windows programs and disable any anti-virus applications before running the setup program.
- 3 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 4 Launch the setup program.
 - A If the CD-ROM drive's Autorun feature is enabled, the setup program should launch automatically. Follow the on-screen instructions until the **Choose Product** dialog box appears (see Figure 4). Check **Add-ons**, then click **Next**.

Figure 4 Choose Product Dialog

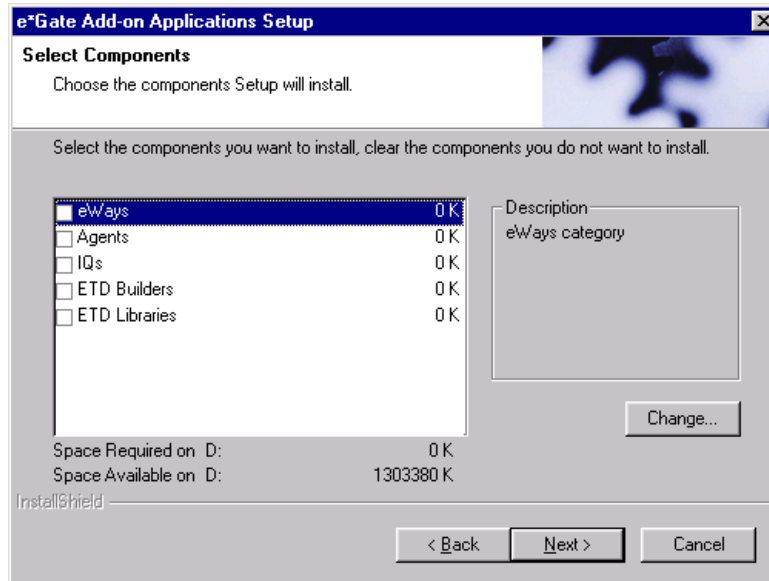


- B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (bypassing the **Choose Product** dialog):

setup\addons\setup.exe

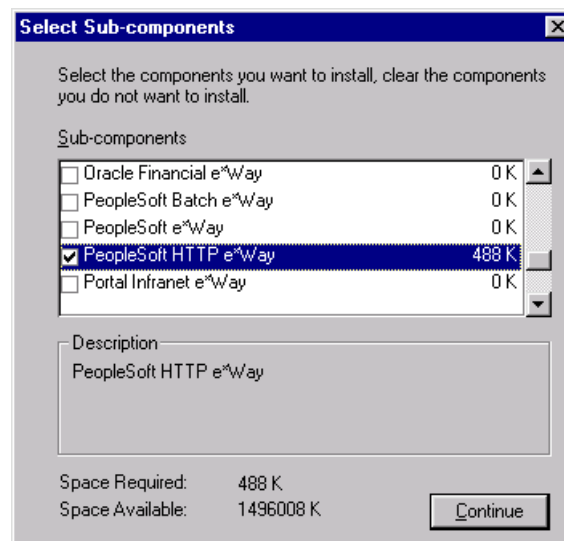
- 5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 5). Highlight—but do not check—**eWays** and then click **Change**.

Figure 5 Select Components Dialog



- 6 When the **Select Sub-components** dialog box appears (see Figure 6), check the **PeopleSoft HTTP e*Way**.

Figure 6 Select e*Way Dialog



- 7 Click **Continue**, and the **Select Components** dialog box reappears.
- 8 Click **Next** and continue with the installation.

9 Add the locations of the following files to your *classpath*:

- jdom.jar
- stchttp.jar
- jsse.jar
- jnet.jar
- jcert.jar
- xerces.jar
- gnu-getopt.jar

10 After the e*Way has been installed, you need to install the SeeBeyond MUX Subscription Handler. See [Windows Systems](#) on page 21.

Subdirectories and Files

Note: *Installing the e*Way Intelligent Adapter for PeopleSoft HTTP installs both Java and Monk versions. Only the files used by the Java version are listed in this section.*

By default, the InstallShield installer creates the following subdirectories and installs the following files within the `\eGate\client` tree on the Participating Host, and the `\eGate\Server\registry\repository\default` tree on the Registry Host.

Table 2 Participating Host & Registry Host

Subdirectories	Files
<code>\classes\</code>	stchttp.jar stcpsoft8appmsg.jar stcutil.jar
<code>\configs\psoft8appmsg\</code>	psoft8appmsg.def
<code>\etd\</code>	psoft8appmsg.ctl
<code>\etd\psoft8appmsg\</code>	psoft8appmsg.xsc
<code>\pkicerts\client\</code>	certmap.txt
<code>\pkicerts\trustedcas\</code>	GTECyberTrustGlobalRoot.cer MicrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer verisign_class3.cer
<code>\ThirdParty\gnu-getopt\classes\</code>	gnu-getopt.jar
<code>\ThirdParty\jdom\jdom-b6\classes\</code>	jdom.jar
<code>\ThirdParty\jsse\jsse1.0.2\classes\</code>	jcert.jar jnet.jar jsse.jar
<code>\ThirdParty\xerces\classes\</code>	xerces.jar

By default, the InstallShield installer also installs the following file within the \eGate\Server\registry\repository\default tree on the Registry Host.

Table 3 Registry Host Only

Subdirectories	Files
\	stcewpsoft8.ctl

Note: *Installing the e*Way Intelligent Adapter for PeopleSoft HTTP also installs the e*Gate API Kit. See the e*Gate API Kit User's Guide for information on installed components.*

2.3.2 UNIX Systems

Installation Procedure

Note: *You are not required to have root privileges to install this e*Way. Log on under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.*

- 1 Log onto the workstation containing the CD-ROM drive and, if necessary, mount the drive.
- 2 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 3 At the shell prompt, type
`cd /cdrom`
- 4 Start the installation script by typing:
`setup.sh`
- 5 A menu of options appears. Select the **Install e*Way** option and follow any additional on-screen instructions.

Note: *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond. Note also that **no spaces** should appear in the installation path name.*

- 6 Add the locations of the following files to your *classpath*:

- jdom.jar
- stchttp.jar
- jsse.jar
- jnet.jar
- jcert.jar
- xerces.jar
- gnu-getopt.jar

- 7 After the e*Way has been installed, you need to install the SeeBeyond MUX Subscription Handler. See [UNIX Systems](#) on page 24.

Subdirectories and Files

Note: *Installing the e*Way Intelligent Adapter for PeopleSoft HTTP installs both Java and Monk versions. Only the files used by the Java version are listed in this section.*

The preceding installation procedure creates the following subdirectories and installs the following files within the /eGate/client tree on the Participating Host, and the /eGate/Server/registry/repository/default tree on the Registry Host.

Table 4 Participating Host & Registry Host

Subdirectories	Files
/classes/	stchttp.jar stcpsoft8appmsg.jar stcutil.jar
/configs/psoft8appmsg/	psoft8appmsg.def
/etd\	psoft8appmsg.ctl
/etd/psoft8appmsg/	psoft8appmsg.xsc
/pkicerts/client/	certmap.txt
/pkicerts/trustedcas/	GTECyberTrustGlobalRoot.cer MicrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer verisign_class3.cer
/ThirdParty/gnu-getopt/classes/	gnu-getopt.jar
/ThirdParty/jdom/jdom-b6/classes/	jdom.jar
/ThirdParty/jsse/jsse1.0.2/classes/	jcet.jar jnet.jar jsse.jar
/ThirdParty/xerces/classes/	xerces.jar

The preceding installation procedure also installs the following file only within the /eGate/Server/registry/repository/default tree on the Registry Host.

Table 5 Registry Host Only

Subdirectories	Files
/	stcewpsoft8.ctl

Note: *Installing the e*Way Intelligent Adapter for PeopleSoft HTTP also installs the e*Gate API Kit. See the e*Gate API Kit User's Guide for information on installed components.*

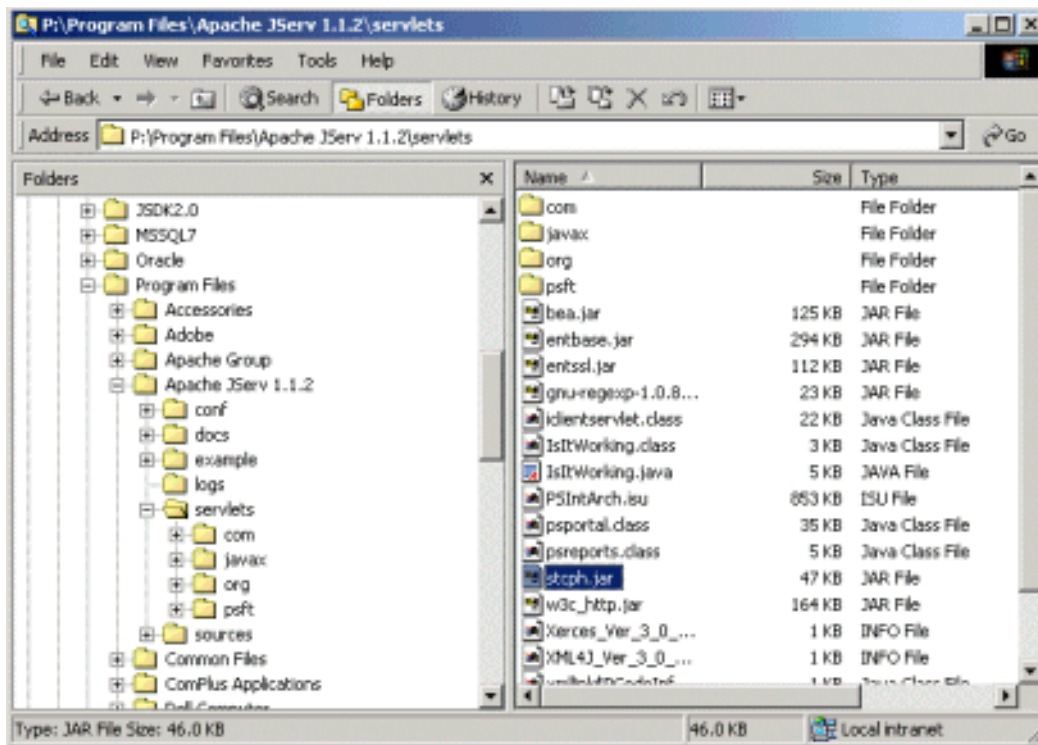
2.4 Installing the MUX Subscription Handler

The SeeBeyond MUX Subscription Handler consists of ten java classes, which are contained in the `stcph.jar` file. This file is available when the e*Gate participating host is installed. The class descriptions are given in [Appendix A](#).

2.4.1 Windows Systems

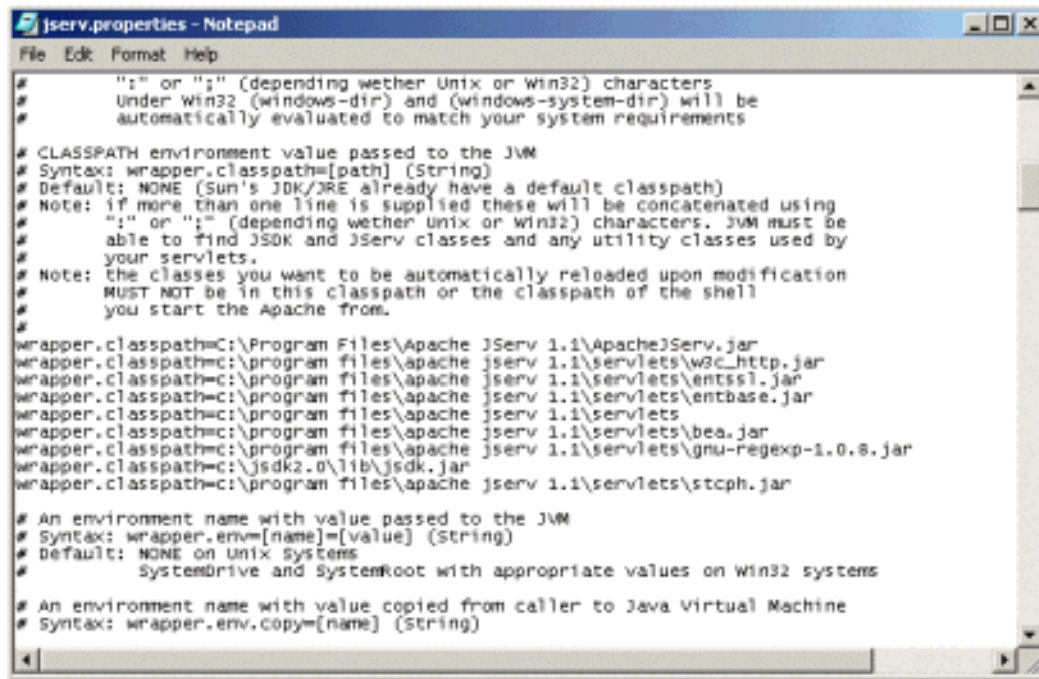
- 1 Stop the web server (Apache or WebLogic).
- 2 Stop the PeopleSoft 8 Application Server for the appropriate Domain.
- 3 Copy the `stcph.jar` file from the e*Gate Participating Host to the directory where servlets must reside. The location depends upon which Web server you are using.
 - A For Apache, copy the `stcph.jar` file to the `servlets` subdirectory under the servlet engine installation directory, as shown in Figure 7.

Figure 7 servlets Directory - Windows



Extract the contents of the `stcph.jar` file (see Figure 8).

Figure 8 jserv.properties File



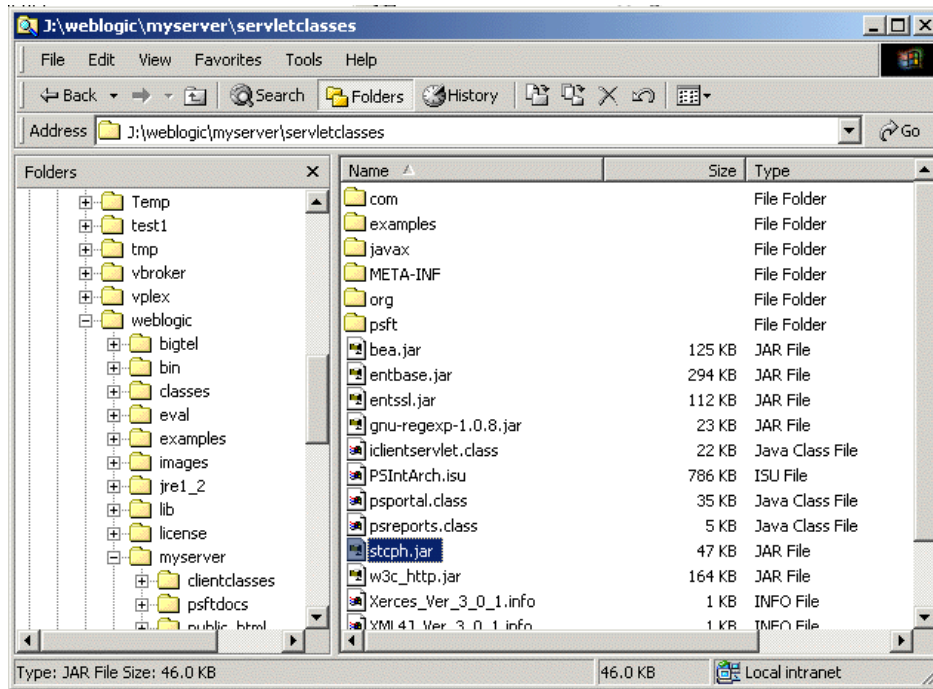
```
jserv.properties - Notepad
File Edit Format Help
#      ":" or ";" (depending wether Unix or Win32) characters
#      Under Win32 (windows-dir) and (windows-system-dir) will be
#      automatically evaluated to match your system requirements
#
# CLASSPATH environment value passed to the JVM
# Syntax: wrapper.classpath=[path] (String)
# Default: NONE (Sun's JDK/JRE already have a default classpath)
# Note: if more than one line is supplied these will be concatenated using
#       ":" or ";" (depending wether Unix or Win32) characters. JVM must be
#       able to find JSDK and JServ classes and any utility classes used by
#       your servlets.
# Note: the classes you want to be automatically reloaded upon modification
#       MUST NOT be in this classpath or the classpath of the shell
#       you start the Apache from.
#
wrapper.classpath=C:\Program Files\Apache JServ 1.1\ApacheJServ.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\w3c_http.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\entss1.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\entbase.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\bea.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\gnu-regexp-1.0.8.jar
wrapper.classpath=C:\jdk2.0\lib\jdk.jar
wrapper.classpath=C:\program files\apache jserv 1.1\servlets\stcph.jar
#
# An environment name with value passed to the JVM
# Syntax: wrapper.env=[name]=[value] (String)
# Default: NONE on Unix Systems
#         SystemDrive and SystemRoot with appropriate values on Win32 systems
#
# An environment name with value copied from caller to Java Virtual Machine
# Syntax: wrapper.env.copy=[name] (String)
```

- B For WebLogic, copy the **stcph.jar** file to the **servletclasses** subdirectory under the **weblogic\myserver** directory in the PeopleSoft Domain installation; for example:

\weblogic\myserver\servletclasses

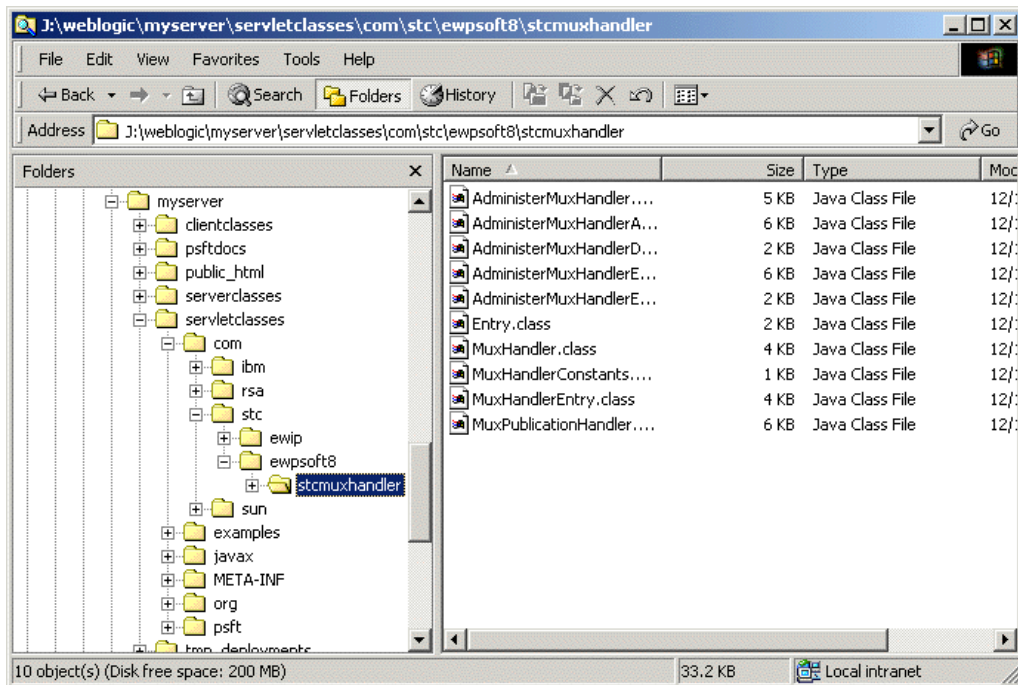
(as shown in Figure 9).

Figure 9 servletclasses Directory - Windows



Extract the contents of the stcph.jar file (see Figure 10).

Figure 10 Extracted Files



- 4 Start (boot) the Application Server for the appropriate domain.
- 5 Start the web server.

2.4.2 UNIX Systems

Note: You must have **jdk 1.2** or later installed on the host to run the **jar** command.

- 1 Stop the Web server (Apache or WebLogic).
- 2 Stop the PeopleSoft 8 Application Server for the appropriate Domain.
- 3 Copy the **stcph.jar** file from the e*Gate Participating Host to the directory where servlets must reside. The location depends upon which Web server you are using.
 - A For Apache, copy the **stcph.jar** file to the **servlets** subdirectory under the **webserv** directory in the PeopleSoft Domain installation; for example:

/psoft/FDM80/webserv/servlets

(as shown in Figure 11).

Figure 11 servlets Directory - Apache

```

total 1976
drwxr-sr-x  7 psoft  sys      1024 Jun 25 16:47 .
drwxr-sr-x 15 psoft  sys      512 Jun 19 15:16 ..
-rw-r--r--  1 psoft  sys     1134 Oct 08 2000 Hello.class
-rw-r--r--  1 psoft  sys     1303 Oct 08 2000 Hello.java
-rw-r--r--  1 psoft  sys     2119 Oct 08 2000 IsItWorking.class
-rw-r--r--  1 psoft  sys     4542 Oct 08 2000 IsItWorking.java
drwxr-sr-x  2 psoft  sys      512 Jun 25 16:47 META-INF
-rwxr-xr-x  1 psoft  sys       7 Feb 28 20:15 XML4J_Ver_3_0_1.info
-rwxr-xr-x  1 psoft  sys       7 Feb 28 20:15 Xerces_Ver_3_0_1.info
-rwxr-xr-x  1 psoft  sys    127587 Feb 28 16:46 bea.jar
drwxr-sr-x  6 psoft  sys      512 Jun 25 16:47 com
-rwxr-xr-x  1 psoft  sys   300083 Feb 28 16:26 entbase.jar
-rwxr-xr-x  1 psoft  sys   113748 Feb 28 16:26 entssl.jar
-rwxr-xr-x  1 psoft  sys   23153 Feb 28 16:46 gnu-regexp-1.0.8.jar
-rwxr-xr-x  1 psoft  sys   22165 Feb 28 17:02 iclientervlet.class
drwxr-sr-x  4 psoft  sys      512 Jun 19 15:18 javax
-r--r--r--  1 psoft  sys   76453 Oct 08 2000 jsdk.jar
drwxr-sr-x  6 psoft  sys      512 Feb 28 20:15 org
drwxr-sr-x  3 psoft  sys      512 Feb 28 20:15 psft
-rwxr-xr-x  1 psoft  sys   35724 Feb 28 17:13 psportal.class
-rwxr-xr-x  1 psoft  sys    5020 Feb 28 17:02 psreports.class
-rw-r----- 1 psoft  sys    47191 Jun 25 16:41 stcph.jar
-rwxr-xr-x  1 psoft  sys  167275 Feb 28 16:26 w3c_http.jar
-rwxr-xr-x  1 psoft  sys     539 Feb 28 17:03 xlink$PCoInfo.class
-rwxr-xr-x  1 psoft  sys    10037 Feb 28 17:03 xlink.class
$ pwd
/psoft/FDM80/webserv/servlets
$
  
```

- B For WebLogic, copy the **stcph.jar** file to the **servletclasses** subdirectory under the **weblogic/myserver** directory in the PeopleSoft Domain installation; for example:

/do1/psoft/fdm80/weblogic/myserver/servletclasses

(as shown in Figure 12).

- 4 Extract the contents of the **stcph.jar** file.

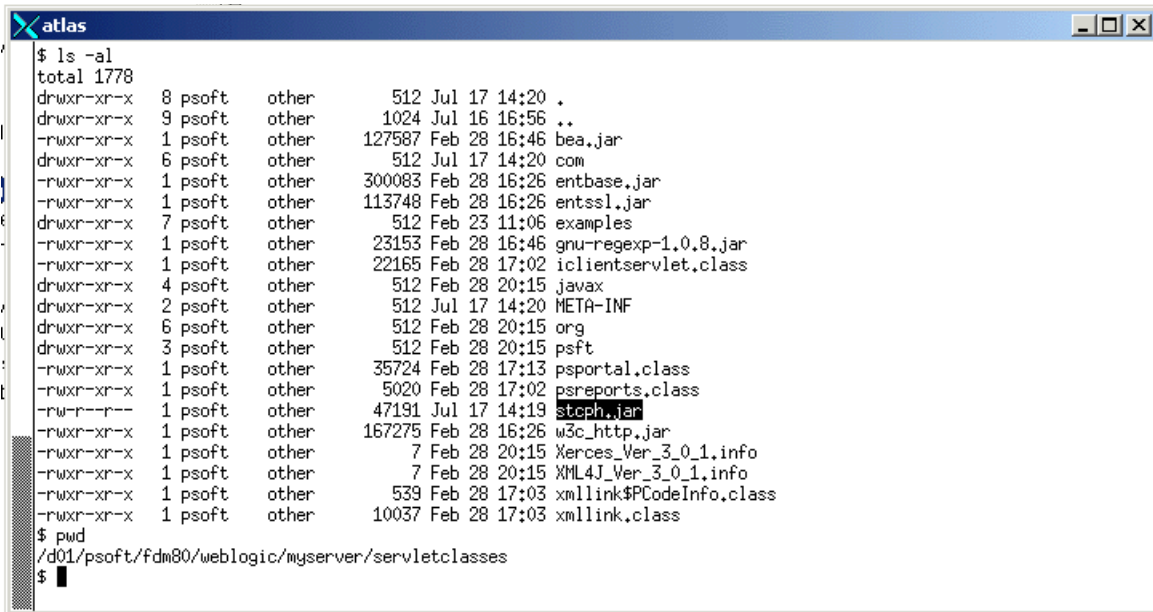
Use the command:

```
jar -tf stcph.jar
```

to extract the classes.
- 5 Start (boot) the Application Server for the appropriate domain.

- 6 Start the Web server.

Figure 12 servlets Directory - WebLogic



```
atlas
$ ls -al
total 1778
drwxr-xr-x  8 psoft  other      512 Jul 17 14:20 .
drwxr-xr-x  9 psoft  other     1024 Jul 16 16:56 ..
-rwxr-xr-x  1 psoft  other    127587 Feb 28 16:46 bea.jar
drwxr-xr-x  6 psoft  other      512 Jul 17 14:20 com
-rwxr-xr-x  1 psoft  other    300083 Feb 28 16:26 entbase.jar
-rwxr-xr-x  1 psoft  other    113748 Feb 28 16:26 entssl.jar
drwxr-xr-x  7 psoft  other      512 Feb 23 11:06 examples
-rwxr-xr-x  1 psoft  other    23153 Feb 28 16:46 gnu-regexp-1.0.8.jar
-rwxr-xr-x  1 psoft  other    22165 Feb 28 17:02 iclientservlet.class
drwxr-xr-x  4 psoft  other      512 Feb 28 20:15 javax
drwxr-xr-x  2 psoft  other      512 Jul 17 14:20 META-INF
drwxr-xr-x  6 psoft  other      512 Feb 28 20:15 org
drwxr-xr-x  3 psoft  other      512 Feb 28 20:15 psft
-rwxr-xr-x  1 psoft  other    35724 Feb 28 17:13 psportal.class
-rwxr-xr-x  1 psoft  other    5020 Feb 28 17:02 psreports.class
-rw-r--r--  1 psoft  other    47191 Jul 17 14:19 stoph.jar
-rwxr-xr-x  1 psoft  other   167275 Feb 28 16:26 w3c_http.jar
-rwxr-xr-x  1 psoft  other      7 Feb 28 20:15 Xerces_Ver_3_0_1.info
-rwxr-xr-x  1 psoft  other      7 Feb 28 20:15 XML4J_Ver_3_0_1.info
-rwxr-xr-x  1 psoft  other    539 Feb 28 17:03 xlink$PCCodeInfo.class
-rwxr-xr-x  1 psoft  other   10037 Feb 28 17:03 xlink.class
$ pwd
/d01/psft/fdm80/weblogic/myserver/servletclasses
$
```

2.5 Optional Example Files

The installation CD contains sample schemas, **psoft8AppMsg.zip** and **JMSQueue.zip**, located in the **samples\ewpsoft8** directory. To use these schemas, you must load them onto your system using the following procedure. See **Sample Schema** on page 83 for descriptions of the sample schema and instructions regarding its use.

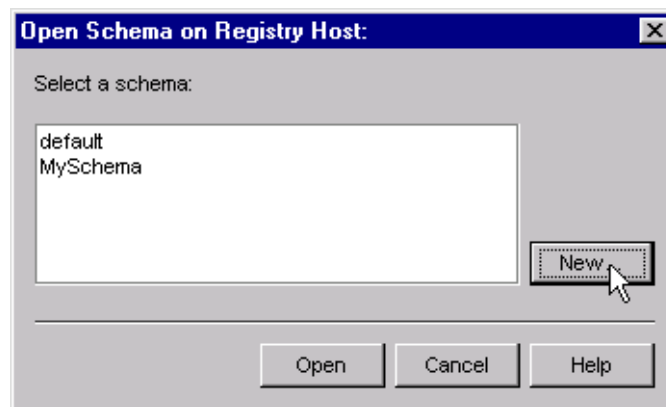
Note: *The PeopleSoft HTTP e*Way must be properly installed on your system before you can run the sample schema.*

2.5.1 Installation Procedure

To load a sample schema

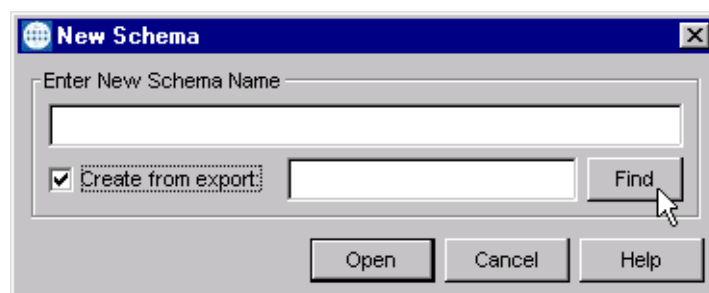
- 1 Invoke the **Open Schema** dialog box and select **New** (see Figure 13).

Figure 13 Open Schema Dialog



- 2 Type the name you want to give to the schema (for example, **PS8.Sample**)
- 3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 14).

Figure 14 New Schema Dialog



- 4 Select **psoft8AppMsg.zip** and click **Open**.

Note: The schema installs with the host name **localhost** and control broker name **localhost_cb**. If you want to assign your own names, copy the file **psoft8AppMsg.zip** to a local directory and extract the files. Using a text editor, edit the file **psoft8AppMsg.exp**, replacing all instances of the name **localhost** with your desired name. Add the edited **.exp** file back into the **.zip** file.

2.5.2 Subdirectories and Files

The preceding procedure creates the following subdirectories and installs the following files within the `\eGate\Server\registry\repository\<SchemaName>` tree on the Registry Host, where `<SchemaName>` is the name you have assigned to the schema in step 2.

Table 6 Subdirectories and Files - psoft8AppMsg

Subdirectories	Files
\	psoft8AppMsg.ctl
\runtime\collaboration_rules\	colPostXML.class colPostXML.ctl colPostXML.java colPostXML.xpr colPostXML.xts colPostXMLBase.class
\runtime\configs\http\client\	conHttp.cfg conHttp.sc
\runtime\configs\psoft8AppMsg\	conPsoft8AppMsg.cfg conPsoft8AppMsg.sc
\runtime\configs\stcewfile\	ewEater.cfg ewEater.sc ewFeeder.cfg ewFeeder.sc ewMUXEater.cfg ewMUXEater.sc
\runtime\configs\stcewimp\	ewMUX.cfg ewMUX.sc
\runtime\etd\	blob.jar blob.ssc blob.xsc
\sandbox\configs\psoft8appmsg\	conPsoft8AppMsg.cfg conPsoft8AppMsg.sc
\sandbox\etd\	common.ctl rtjar.ctl
\sandbox\etd\psoft8appmsg\	psoft8appmsg.xsc

Table 7 Subdirectories and Files - JMSQueue

Subdirectories	Files
\	jms_test.ctl
\jms_test\runtime\configs\messageservice\	localhost_iqmgr.cfg localhost_iqmgr.sc
\jms_test\sandbox\configs\messageservice\	test.cfg test.sc
\jms_test\sandbox\configs\stcewfile\	eater.cfg eater.sc feeder.cfg feeder.sc

PeopleSoft 8 Setup

This chapter describes procedures for configuring the PeopleSoft 8.1 Application to interact properly with the PeopleSoft HTTP e*Way.

3.1 Overview

To publish data to, or subscribe to data from, PeopleSoft 8, the following must be created and configured within the PeopleSoft 8 environment:

- Message Nodes
- Messages
- Message Channels
- Subscription Handlers (PeopleSoft 8.1 only)

PeopleSoft 8 comes with a set of predefined Message Definitions and Message Channels that can be used as-is. You can also create your own Message Definitions and Message Channels. You must know in advance which Message Definition(s) and which Message Channel(s) to use during the external configuring process.

For PeopleSoft 8.1, please refer to the *PeopleSoft 8 EIP Catalog* for more information. Also, the PeopleSoft 8 documentation on *Adding and Configuring Subscription Handlers* contains valuable information regarding Subscription Handlers.

In PeopleSoft 8.4, many of these procedures have been moved into the PeopleTools section of the main PeopleSoft user interface. Please refer to the PeopleSoft *Integration Broker PeopleBook* for more information.

For purposes of this User's Guide, *publishing* refers to sending outbound messages to PeopleSoft 8 and *subscribing* refers to receiving inbound messages from PeopleSoft 8.

3.2 Configuring for Publication

To enable the PeopleSoft HTTP e*Way to publish XML Messages to PeopleSoft 8, the following configuration steps must be performed.

[Creating PeopleSoft 8 Message Node for the e*Way](#) on page 30

[Activating the Message Definition for Publication](#) on page 34

[Defining Message Channel Routing Rules](#) on page 37

[Defining Routing Directions for Message Nodes](#) on page 41

[Adding the PeopleSoft 8 Subscription Handler](#) on page 43

3.2.1 Creating PeopleSoft 8 Message Node for the e*Way

Complete the following procedures to create a PeopleSoft 8 Message Node in order to configure the PeopleSoft HTTP e*Way to publish XML messages.

PeopleSoft 8.4/PeopleTools 8.42

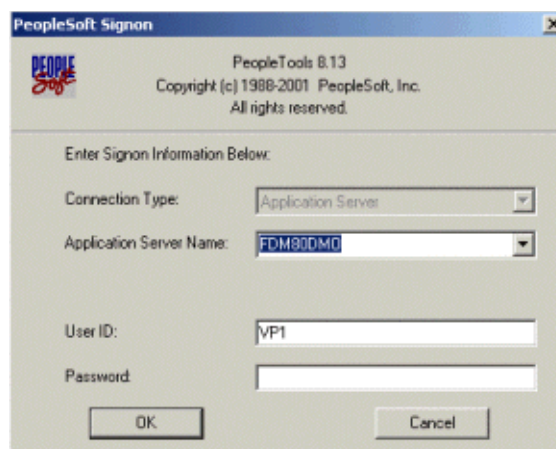
You must define a source node in PeopleSoft (see “Administering Basic Integrations: Configuring Nodes: Defining a Node” in the PeopleTools *PeopleBooks*). In PeopleSoft 8.4, the Node Definition facility has been moved into the PeopleTools section of the main PeopleSoft user interface; otherwise, the procedure is analogous to that shown for PeopleSoft 8.1.

PeopleSoft 8.1/PeopleTools 8.13

To define the Message Node to publish XML messages

- 1 Sign on to PeopleTools, and start the Application Designer.

Figure 15 PeopleSoft Signon



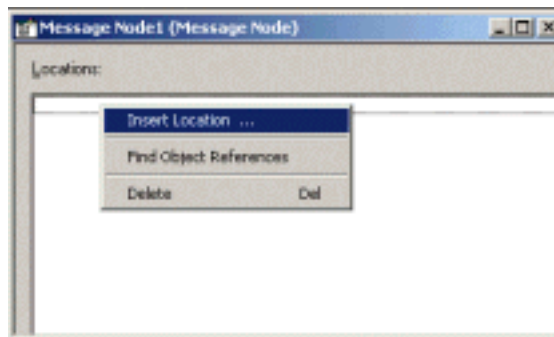
- 2 From the Application Designer File menu, select **New**.
- 3 In the **New** dialog box, select **Message Node** and click **OK** to display the Message Node dialog box for Node 1.

Figure 16 New Pop-Up Menu - Message Node



- 4 Right-click within the **Locations** pane, and a pop-up menu appears. Select **Insert Location** from the pop-up menu to enter the PeopleSoft 8 Application Messaging Gateway (servlet) URL.

Figure 17 Message Node Window - Insert Location



- 5 Type the following URL of the PeopleSoft 8 Gateway Servlet into the Location dialog box.

A For Apache:

http://PSFTHOST/servlets/psft.pt8.gateway.GatewayServlet

B For WebLogic:

http://PSFTHOST/servlets/gateway

Note: You must replace the name **PSFTHOST** in the above URL with the actual name of the host computer on which PeopleSoft 8 is installed.

Figure 18 Location Dialog Box



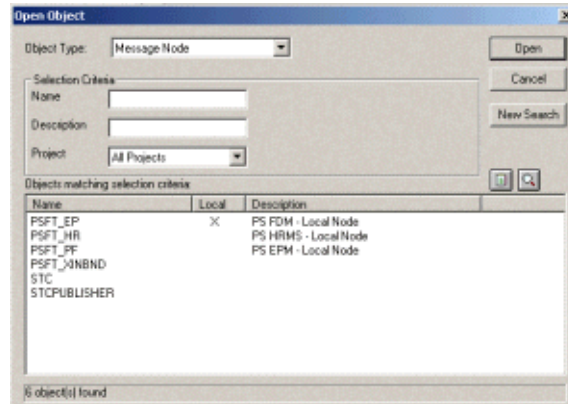
- 6 Click **OK** to save the URL. The URL name you have entered then appears in the Message Node dialog box.
- 7 Save the Message Node and commit it to the PeopleSoft 8 database as follows:
 - A Select **Save As** from the **File** drop-down menu.
 - B Type the name of the Message Node you are saving into the text box. (The example above uses **STCPUBLISHER**.) This name is needed for the e*Way configuration as the **From Node** parameter.

Figure 19 Save As Dialog Box



- 8 From the Application Designer **File** menu, select **Open** to invoke the **Open Object** window.
- 9 Verify that the Message Node you created is ready for use by PeopleSoft 8.
 - A In the Open Object window, select **Message Node** from the Object Type list and click the **Open** button.
 - B A list of all Message Nodes within the system appears. The name of the newly-created message node should appear in the **Objects matching selection criteria** pane. If so, you have successfully completed creating a Message Node for SeeBeyond to publish data to PeopleSoft 8.

Figure 20 Open Object Window - Message Node



Note: All Message Nodes with **PSFT** prefixes were created by the PeopleSoft 8 installation. **PSFT_EP** is the PeopleSoft Local Node for the Financials application. It is specified as a subscriber to messages sent from the HTTP e*Way, and a publisher of messages to the Multiplexer e*Way.

3.2.2 Activating the Message Definition for Publication

As mentioned previously, PeopleSoft 8 comes with a set of predefined Message Definitions. The desired Message Definition is configurable in the e*Way as the **Subject** parameter. The following instructions describe how to activate the Message Definition for subscription to the SeeBeyond Multiplexer e*Way.

Note: For purposes of this example, the ADVANCED_SHIPPING_RECEIPT Message Definition is activated for publish / subscribe.

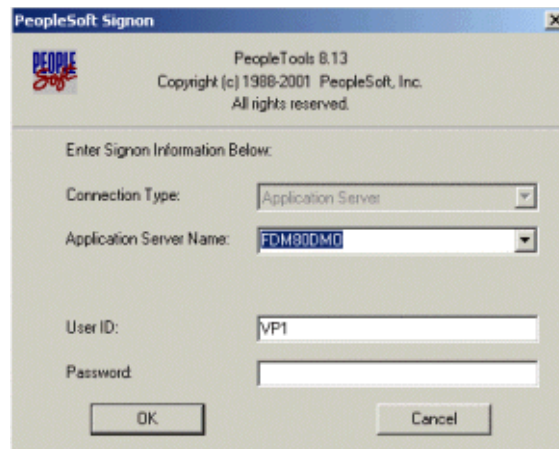
PeopleSoft 8 Message Definitions List

Each message used for publication must be defined. This definition corresponds to the XML Message the e*Way publishes, and contains the elements of the data to be published. However, before the e*Way can publish any data, the Message Definition must be activated. A list of these definitions can be found within the Application Designer.

To activate the Message Definition for subscription to the e*Way

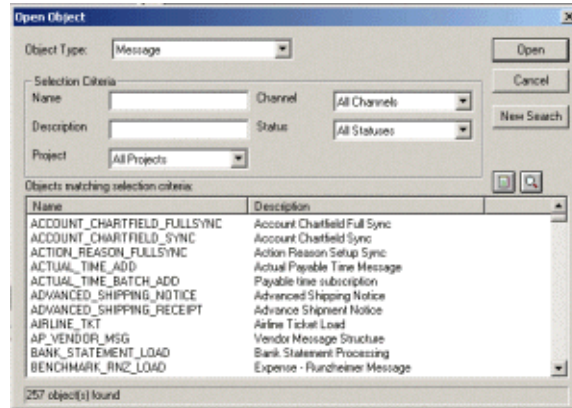
- 1 Sign on to PeopleTools Application Designer.

Figure 21 PeopleSoft 8 Signon



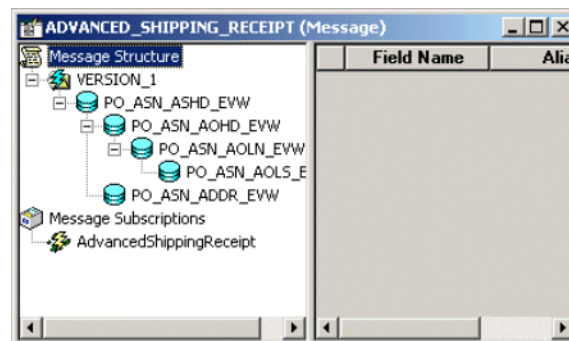
- 2 From the Application Designer **File** menu, select **Open** to invoke the **Open Object** window.
- 3 Select **Message** from the **Object Type** list, and you are presented with a list of all available PeopleSoft 8 Message Definitions.

Figure 22 Open Object Window



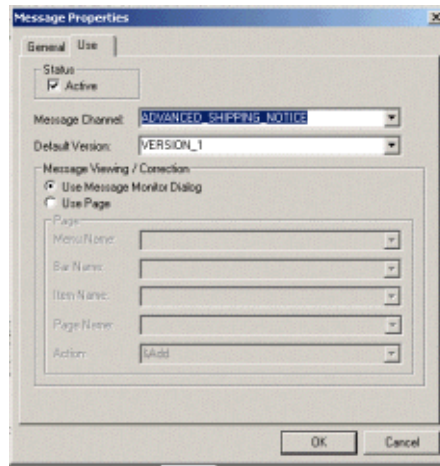
- 4 Select the Message Definition you want and double-click on that selection; for example, **ADVANCED_SHIPPING_RECEIPT**.
- 5 The Message window appears, displaying the complete record details of the chosen Message Structure.

Figure 23 **ADVANCED_SHIPPING_RECEIPT** Details



- 6 From the Application Designer **File** menu, select **Object Properties** to open the Message Properties dialog box.
- 7 In the Message Properties dialog box, click the **Use** tab to display the Status field.

Figure 24 Message Properties Dialog Box - Use Tab



- 8 Check the Active button and click OK to save the settings.
- 9 From the File menu, select Save to save and commit the changes to the Message Definition. You have now activated the Message Definition for publishing or subscribing.

3.2.3 Defining Message Channel Routing Rules

Before proceeding with this process, you should determine which Message Channel to use. The Message Channel to use is configurable in the e*Way as the **Channel** parameter.

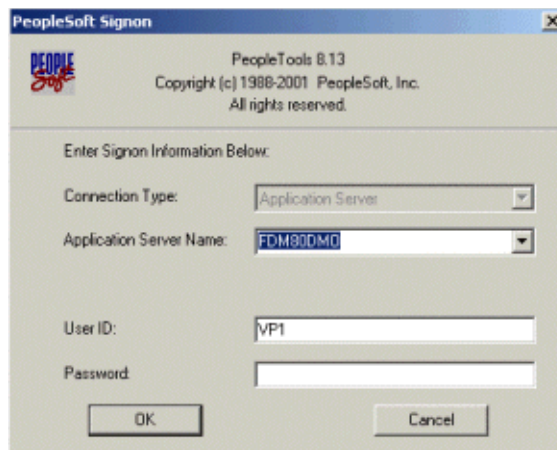
Configuring the Message Channel

Each Message Channel logically groups Messages together. For purposes of this documentation, **ADVANCED_SHIPPING_RECEIPT** Message is grouped into the **ADVANCED_SHIPPING_NOTICE** Message Channel.

To configure the Message Channel

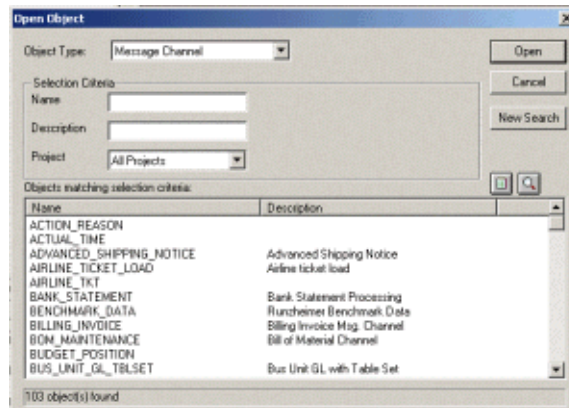
- 1 Sign on to PeopleTools Application Designer.

Figure 25 PeopleSoft Signon



- 2 From the Application Designer **File** menu, select **Open** to invoke the **Open Object** window.

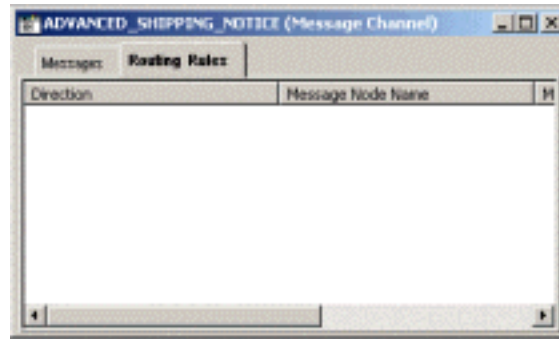
Figure 26 Open Object Window



- 3 Select **Message Channel** from the **Object Type** list.

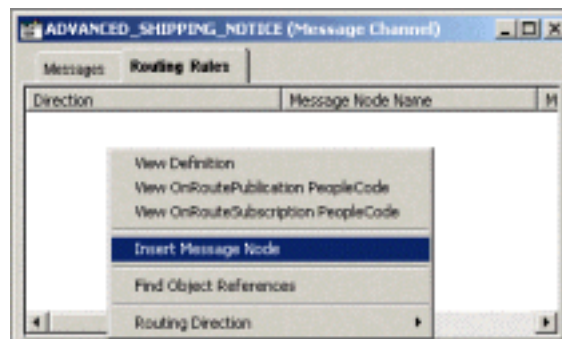
- 4 Click **Open**, and a list of all available Message Channels on the system appears.
- 5 Double-click on the name of the desired Message Channel. The Message Channel window appears for that channel.

Figure 27 Message Channel - ADVANCED_SHIPPING_NOTICE (1)



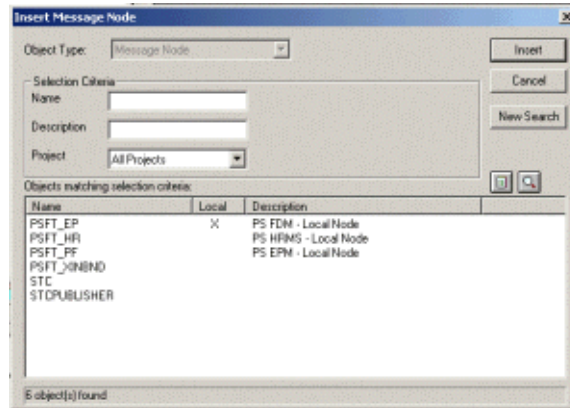
- 6 Left-click the **Routing Rules** tab, then right-click in the data pane. A pop-up menu appears with the following options:
 - ◆ View Definition
 - ◆ View **OnRoutePublication** PeopleCode
 - ◆ View **OnRouteSubscription** PeopleCode
 - ◆ Insert Message Node
 - ◆ Find Object References
 - ◆ Routing Direction

Figure 28 Message Channel - ADVANCED_SHIPPING_NOTICE (2)



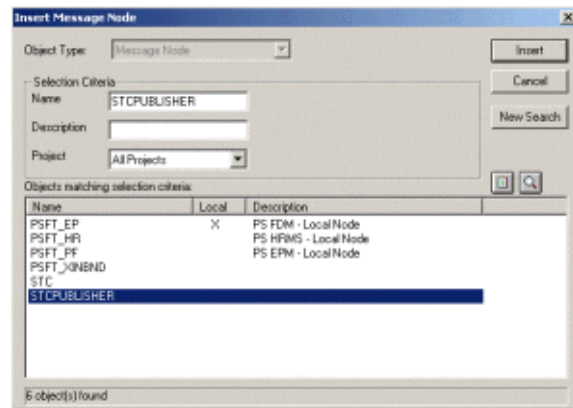
- 7 Left-click to select **Insert Message Node**. The Insert Message Node window appears, displaying the available Message Nodes.

Figure 29 Insert Message Node Window



- 8 Click on **PSFT_EP**, then click the **Insert** button. This inserts the message into the Routing Rules Table.
- 9 Click on **STCPUBLISHER**, then click **Insert**.

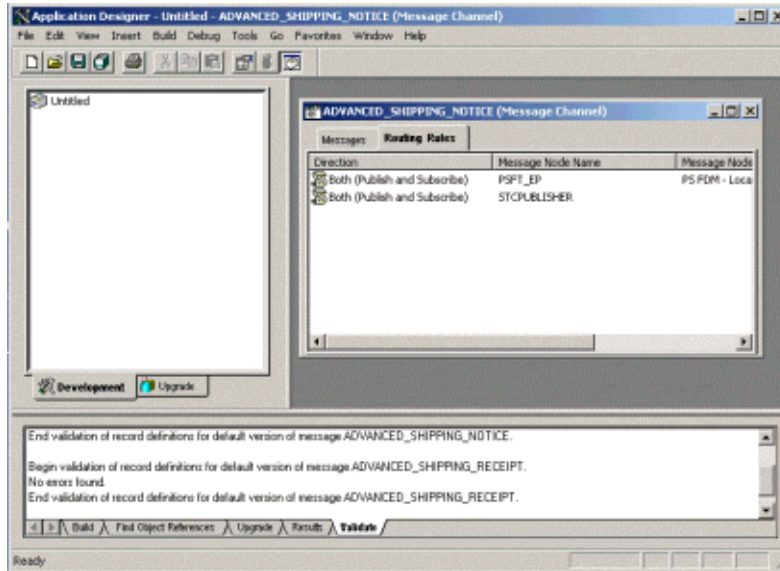
Figure 30 Insert Message Node Window - STCPUBLISHER



- 10 Click **Cancel** to exit the Insert Message Node window.

The Message Nodes are now defined on the Routing Rules tab of the Message Channel window.

Figure 31 Message Channel - ADVANCED_SHIPPING_NOTICE (3)



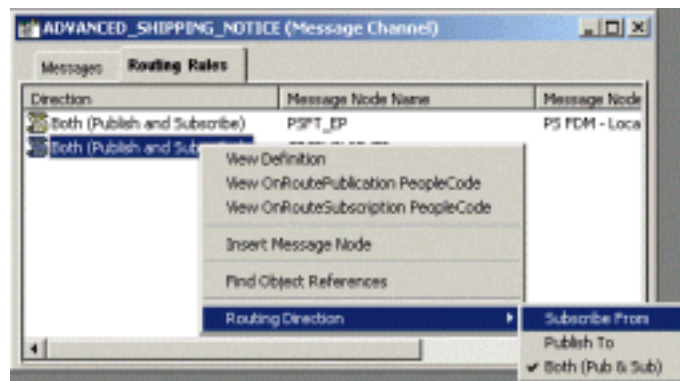
3.2.4 Defining Routing Directions for Message Nodes

Routing Directions provide you with the ability to assign destinations, either Publish To or Subscribe From, a Message Node. This section describes the procedure for defining the Routing Directions for the SeeBeyond Message Node, **Subscribe From**.

To define the Routing Directions

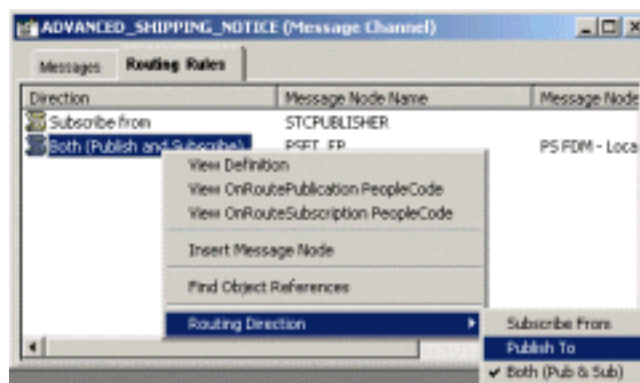
- 1 From the **Direction** column of the **Routing Rules** tab, right-click on **Both (Publish and Subscribe)** for the Message Node Name **STCPUBLISHER**. A pop-up menu appears with several options.

Figure 32 Subscribe From Selection



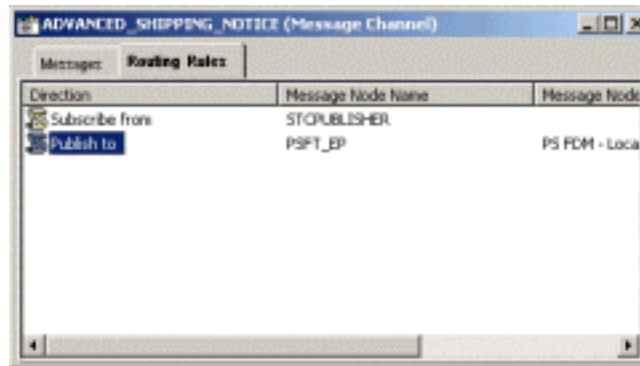
- 2 On the menu, left-click **Routing Direction**, then left-click **Subscribe From** on the secondary menu.
- 3 Within the Message Channel window, right-click on **Both (Publish and Subscribe)** for the Message Node Name **PSFT_EP**. Again, the pop-up menu appears.

Figure 33 Publish To Selection



- 4 Left-click **Routing Direction**, then left-click **Publish To** on the secondary menu.

Figure 34 Routing Rules > Verify



- 5 Under the Routing Rules tab, you now see that the SeeBeyond Message Node is subscribing to messages from **STCPUBLISHER** and is publishing to **PSFT_EP**.
- 6 From the **File** menu, select **Save** to save and commit the changes to the Message Definition. You have now defined the Routing Rule that allows the appropriate Message to be published from the e*Way to PeopleSoft 8.

3.2.5 Adding the PeopleSoft 8 Subscription Handler

Note: In performing the procedures in this section you need to know the following

- PeopleSoft 8 parameters:
- Jolt Listener Host
- Jolt Listener Port
- People Tools version
- Operator ID
- Operator ID password

To Access the PeopleSoft 8 Handler Directory (ConfigServlet)

- 1 Start up any supported browser and open the Handler Directory by typing the following URL, where **PSFTHOST** represents the host on which PeopleSoft 8 Application Messaging Gateway is installed.

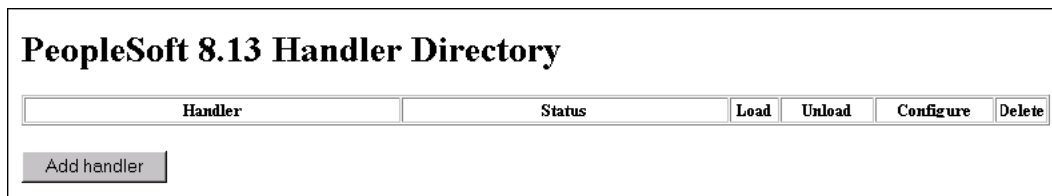
A For Apache:

http://PSFTHOST/servlets/psft.pt8config.ConfigServlet

B For WebLogic:

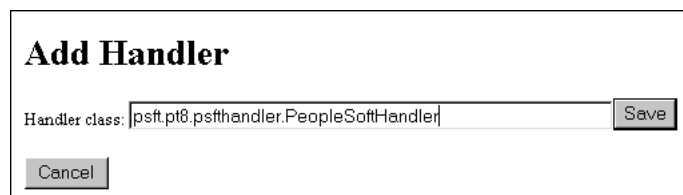
http://PSFTHOST/servlets/gateway.administration

Figure 35 PeopleSoft 8 Handler Directory Page (1)



- 2 On the initial page, click the **Add handler** button to add a new handler. The **Add Handler** page appears (see Figure 36).

Figure 36 Add Handler Page



- 3 On the **Add Handler** page, type the PeopleSoft 8 Handler class into the Handler class text box:

psft.pt8.psfthandler.PeopleSoftHandler

- 4 Click the **Save** button to save the newly-added PeopleSoft 8 Handler.

After the Handler has been saved, the **Handler Directory** page reappears, showing the newly-added Handler. Also, two additional buttons appear: **Load** and **Delete** (see Figure 37).

Figure 37 PeopleSoft 8 Handler Directory Page (2)

PeopleSoft 8.13 Handler Directory

Handler	Status	Load	Unload	Configure	Delete
psft.pt8.psfthandler.PeopleSoftHandler	Not loaded	Load			Delete

Add handler

- Click the **Load** button to load the PeopleSoft 8 Handler class just added. The Status changes from **Not loaded** to **Loaded successfully** and two new buttons appear: **Unload** and **Configure** (see Figure 38).

Figure 38 PeopleSoft 8 Handler Directory Page (3)

PeopleSoft 8.13 Handler Directory

Handler	Status	Load	Unload	Configure	Delete
psft.pt8.psfthandler.PeopleSoftHandler	Loaded successfully		Unload	Configure	

Add handler

- Click the **Configure** button to configure the handler. The **Manage Lookup Table** page for the PeopleSoft 8 Handler appears (see Figure 39).

Figure 39 Manage Lookup Table Page (1)

Manage Lookup Table

Node	Machine address:port#	Tools Version	OPRID	Actions
Add a new node				

- Click the **Add a new node** button to associate the node with this subscription handler. The **Add an address** window appears (see Figure 40).

Figure 40 Add an Address Page

Add an address

Node	Machine address:port#	Tools Version	OPRID	Password
PSFT_EP <small>e.g., BGEE_REMOTE</small>	//solutions9:9000 <small>e.g., //AKTT:9000,...</small>	8.13 <small>e.g., 8.10</small>	VP1 <small>e.g., PTDMO</small>	<input type="password"/> <small>e.g., PASSWORD</small>

Save address

Cancel

- 8 Type the values for the new node, PSFT_EP, associated with the subscription handler.

Note: These values are required, and must be entered.

- 9 Click **Save address** to save the values just entered. You automatically return to the Manage Lookup Table page, now showing the node just added and configured (see Figure 41).

Figure 41 Manage Lookup Table Page (2)

Manage Lookup Table

Node	Machine address:port#	Tools Version	OPRID	Actions		
PSFT_EP	//solutions9:9000	8.13	VP1	Edit	Delete	Add

Add a new node

The Application Messaging Gateway is now ready to receive XML messages from the e*Way and publish the XML messages to PeopleSoft 8.

3.3 Configuring for Subscription

3.3.1 PeopleSoft 8/PeopleTools 8.13

To enable PeopleSoft 8 to publish XML Messages to the PeopleSoft HTTP e*Way, the following configuration steps must be performed.

[Creating a MUX e*Way Message Node](#) on page 46

[Activating the Message Definition for Subscription](#) on page 46

[Defining the Message Channel Routing Rules](#) on page 46

[Adding the SeeBeyond MUX Subscription Handler](#) on page 46

[Configuring the SeeBeyond MUX Subscription Handler](#) on page 49

Creating a MUX e*Way Message Node

Refer to the section [“Creating PeopleSoft 8 Message Node for the e*Way” on page 30](#) to create a message node associated with the Multiplexer e*Way. A Message Node called STCMUX is used as an example.

Activating the Message Definition for Subscription

Refer to the section [“Activating the Message Definition for Publication” on page 34](#) to activate the appropriate Message to be published to the Multiplexer e*Way. For this example, the message PO-EXPECTED_RECEIPT_SHIPTO is activated.

Defining the Message Channel Routing Rules

Refer to the section [“Defining Message Channel Routing Rules” on page 37](#) to define the Routing rules for the Message Channel to be used.

- Insert the PSFT_EP Message Node and the MUX Message Node previously created.
- Define the Routing Direction. Select **Subscribe From** for PSFT_EP and **Publish To** for the MUX Message Node (STCMUX).

Adding the SeeBeyond MUX Subscription Handler

Obtain the MUX configuration values for the e*Way which is to receive the XML message(s) from PeopleSoft. These are required when configuring the Message Node corresponding to the Subscription Handler.

To Obtain the MUX Configuration Values

- 1 Access the PeopleSoft 8 Handler Directory (ConfigServlet).
- 2 Start up any supported browser.

- 3 Open the Handler Directory page by typing the following URL, where **PSFTHOST** represents the host on which PeopleSoft 8 Application Messaging Gateway is installed (see Figure 42):

A For Apache:

http://PSFTHOST/servlets/psft.pt8config.ConfigServlet

B For WebLogic:

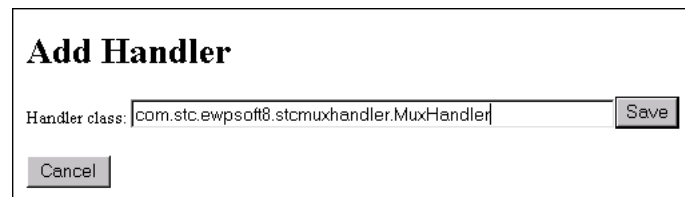
http://PSFTHOST/servlets/gateway.administration

Figure 42 PeopleSoft Handler Directory Page (1)



- 4 Click the **Add handler** button to add the MUX handler.

Figure 43 Add Handler Page

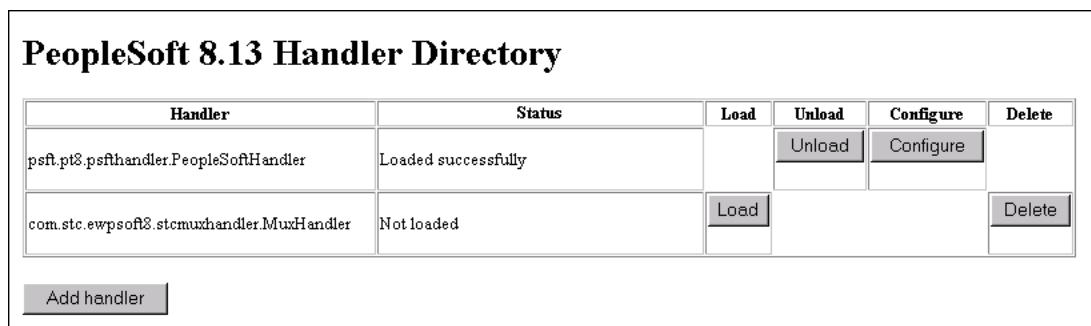


- 5 Add the SeeBeyond MUX Subscription Handler by typing the following into the Handler class box:

com.stc.ewpsoft8.stcmuxhandler.MuxHandler

- 6 Click the **Save** button. The system returns you to the **Handler Directory** page, now showing the MUX Subscription Handler you just added (see Figure 44).

Figure 44 PeopleSoft Handler Directory Page (2)



- Click the **Load** button. The Status field changes to **Loaded successfully** and two additional buttons appear: **Unload** and **Configure** (see Figure 45).

Figure 45 PeopleSoft Handler Directory Page (3)

PeopleSoft 8.13 Handler Directory

Handler	Status	Load	Unload	Configure	Delete
psft.pt8.psftHandler.PeopleSoftHandler	Loaded successfully		Unload	Configure	
com.stc.ewpsof8.stcmuxhandler.MuxHandler	Loaded successfully		Unload	Configure	

Add handler

- Click the **Configure** button for the SeeBeyond MUX Handler. The **SeeBeyond MUX Handler Directory** page for the MUX Handler opens (see Figure 46).

Figure 46 SeeBeyond MUX Handler Directory Page (1)

SeeBeyond MUX Handler Directory

Node Name	MUX Host	MUX Port	MUX Expire [sec]	MUX Timeout [msec]	Uncompress?	Base64 Decode?	Include Headers?	Log File	Edit	Delete
Add a SeeBeyond MUX node										

- Click the **Add a SeeBeyond MUX node** button to associate a node with this Subscription Handler.

Figure 47 Add SeeBeyond MUX Handler Page

Add SeeBeyond MUX Handler

Node Name	MUX Host	MUX Port	MUX Expire [secs]	MUX Timeout [msecs]	Uncompress?	Base64 Decode?	Include Headers?	Log File
STCMUX	XXX	26051	10	10000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	C:\20020122035008.log

Save

Cancel

- Enter the values for the new node associated with the subscription handler. Scroll to the right to access additional columns.
- In the **Include Headers?** column, indicate whether or not you want header information to be retained in the received messages.
 - Selecting the check box preserves the header information.

B Deselecting the check box strips the header information.

Note: All values are required.

12 Click the **Save** button to save the values entered.

Note: You should now be able to ping the MUX host from the machine where the **stcph.jar** file is installed. You may need to use the full machine host name (for example, **johndoe.seebeyond.com**).

Click the **Save** button to display the SeeBeyond MUX Handler Directory page with the **STCMUX** node entries added and configured (see Figure 48).

Figure 48 SeeBeyond MUX Handler Directory Page (2)

SeeBeyond MUX Handler Directory										
Node Name	MUX Host	MUX Port	MUX Expire [sec]	MUX Timeout [msec]	Uncompress?	Base64 Decode?	Include Headers?	Log File	Edit	Delete
STCMUX	XXX	26051	10	10000	Yes	Yes	No	C:\psft_logs\WAMESMUX.log	Edit	Delete

Add a SeeBeyond MUX node

If the entries are correct, the Application Messaging Gateway now can receive XML messages from PeopleSoft 8 and publish the XML messages to the PeopleSoft HTTP e*Way (in MUX mode).

Configuring the SeeBeyond MUX Subscription Handler

The SeeBeyond MUX Subscription Handler itself may require configuration. See the discussion in [Subscribing to PeopleSoft 8.1](#) on page 81.

3.3.2 PeopleSoft 8.4/PeopleTools 8.42

Subscribing to PeopleSoft 8.4 makes use of the PeopleSoft JMS Target Connector, which can publish messages to a topic or insert a message into a queue. To enable PeopleSoft 8 to publish XML Messages to the PeopleSoft HTTP e*Way, the following steps need to be followed:

- 1 Import your schema.
- 2 Bind the e*Gate JMS objects for JNDI lookup using the File System Context on the e*Gate participating host (see "Binding Objects for JNDI Lookup" in the *SeeBeyond JMS Intelligent Queue User's Guide* for instructions).
- 3 Configure the PeopleSoft system for JMS:
 - C** Copy the following files into the `\c$\bea\wserver6.1\config\peoplesoft\applications\PSIGW\WEB-INF\lib` directory on the PeopleSoft server:

- ◆ **stcjms.jar** (found in `\eGate\client\classes\`)
- ◆ The necessary **.jar** files for your JDNI service provider, as described in “Binding Objects for JNDI Lookup” in the *SeeBeyond JMS Intelligent Queue User’s Guide* (see step 2, above).
- D Define a target node (see “Administering Basic Integrations: Configuring Nodes: Defining a Node” in the PeopleTools *PeopleBooks*).
- E Modify the PeopleSoft gateway properties file (**integrationGateway.properties**) to publish to the JMS target connector. See the PeopleTools *PeopleBook* sections “Administering Basic Integrations: Specifying a Gateway and Connector” and “Managing the Integration Gateway: Using the PeopleSoft 8.1 Target Connector.”

System Implementation

In this chapter we summarize the procedures required for implementing a working system incorporating the Java-enabled PeopleSoft HTTP e*Way. Please refer to the *e*Gate Integrator User's Guide*.

4.1 Overview

This e*Way provides a specialized transport component for incorporation into an operational Schema. The schema also contain Collaborations, linking different data or Event types, and Intelligent Queues. Typically, other e*Way types also are used as components of the Schema.

One or more sample schemas, included in the software package, are described at the end of this chapter. These can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema.

4.1.1 Pre-Implementation Tasks

Install the SeeBeyond Software

The first task is to install the SeeBeyond software as described in [Chapter 2](#).

Import the Sample Schema

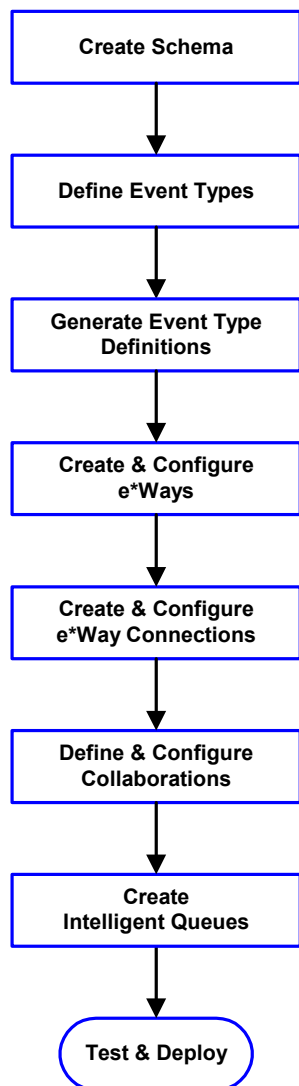
If you want to use the sample schema supplied with the e*Way, the schema files must be imported from the installation CD-ROM (see [Optional Example Files](#) on page 26).

Note: *It is highly recommended that you make use of the sample schemas to familiarize yourself with e*Way operation, test your system, and use as templates for your working schemas.*

Configure the PeopleSoft HTTP System

Follow the procedure described in [PeopleSoft 8 Setup](#) on page 29.

4.1.2 Implementation Sequence



- 1 The first step is to create a new Schema—the subsequent steps apply only to this Schema (see [Creating a Schema](#) on page 53).
- 2 The second step is to define the Event Types you are transporting and processing within the Schema (see [Creating Event Types](#) on page 54).
- 3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see [Generating Event Type Definitions](#) on page 55).
- 4 The fourth step is to create and configure the required e*Ways (see [Setting Up the e*Way](#) on page 90).
- 5 The fifth step is to configure the e*Way Connections (see [Creating e*Way Connections](#) on page 98).
- 6 Next you need to define and configure the Collaborations between Event Types (see [Defining Collaborations](#) on page 77).
- 7 Now you need to create Intelligent Queues to hold published Events (see [Creating Intelligent Queues](#) on page 78).
- 8 Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

4.1.3 Viewing e*Gate Components

Use the Navigator and Editor panes of the e*Gate Schema Designer to view the various e*Gate components. Note that you may only view components of a single schema at one time, and that all operations apply only to the current schema. All procedures in this chapter should be performed while displaying the **Components** Navigator pane. See the *e*Gate Integrator User's Guide* for a detailed description of the features and use of the Schema Designer.

4.2 Creating a Schema

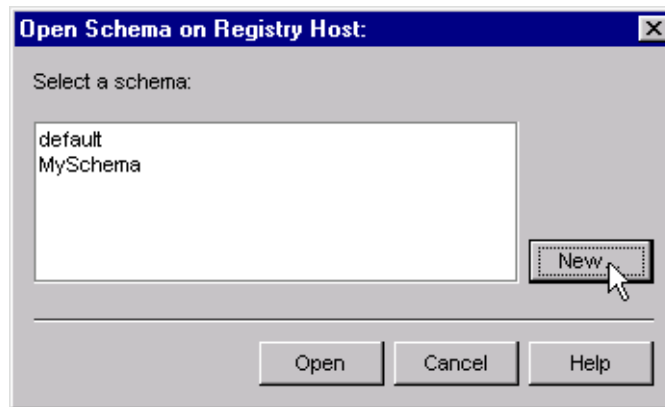
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

Because all setup and configuration operations take place within an e*Gate schema, a new schema must be created, or an existing one must be started before using the system. Schemas store all their configuration parameters in the e*Gate Registry.

To select or create a schema

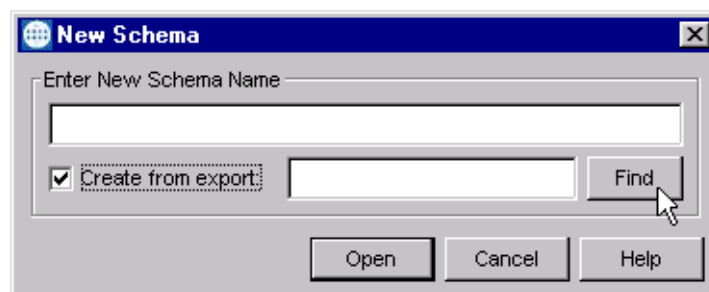
- 1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

Figure 49 Open Schema Dialog



- 2 Clicking **New** invokes the **New Schema** dialog box (Figure 50).

Figure 50 New Schema Dialog




- 3 Enter a new schema name and click **Open**.
- 4 The e*Gate Schema Designer then opens under your new schema name.
- 5 From the **Options** menu, click on **Default Editor** and select **Monk**.
- 6 Select the **Components** tab, found at the bottom of the Navigator pane of the e*Gate Schema Designer window.
- 7 You are now ready to begin creating the necessary components for this new schema.

4.3 Creating Event Types

Within e*Gate, messages and/or packages of data are defined as Events. Each Event must be categorized into a specific Event Type within the schema.

To define the Event Types

- 1 In the e*Gate Schema Designer's Navigator pane, select the **Event Types** folder.
- 2 On the Palette, click the **New Event Type** button .
- 3 In the **New Event Type Component** box, enter the name for the input Event Type and click **Apply**. Use this method to create all required Event Types, for example:
 - ◆ **InboundEvent**
 - ◆ **ValidEvent**
 - ◆ **InvalidEvent**
- 4 After you have created the final Event Type, click **OK**.

4.4 Generating Event Type Definitions

As the name implies, an Event Type Definition (ETD) defines the structure of the Event Types employed in your Schema. Any one ETD can be associated with more than one Event Type within the schema. In the PeopleSoft HTTP e*Way, ETDs are created semi-automatically using the DTD Builder, once a DTD has been generated.

4.4.1 Generating DTDs from PeopleSoft

PeopleTools 8.13 does not contain a DTD generation utility, nor does it include any sample DTDs; however, PeopleTools Application Designer can be extended to generate DTDs using third-party software. These DTDs can then be converted to ETDs using SeeBeyond's DTD Converter. This chapter describes a "workaround" procedure that "reverse-engineers" a DTD from a sample XML message generated within PeopleSoft.

***Note:** The procedure described may not work for all Message Definitions, and you need to know the data constraints for a particular Message Definition in order to correctly populate the message with sample data. You should also be familiar with XML messaging and working with DTDs.*

The workaround procedure involves several sequential steps, which are described under the following headings:

- 1 **Generating and Publishing an XML Test Message** on page 56.
- 2 **Extracting and Viewing the XML Test Message** on page 62.
- 3 **Generating a DTD for the XML File** on page 67.

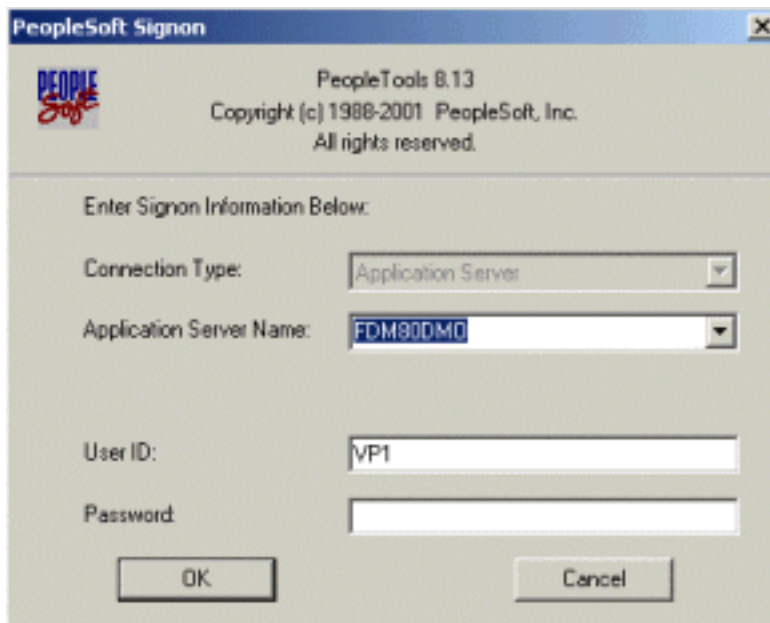
Generating and Publishing an XML Test Message

The first step is to use the PeopleSoft 8 Application Designer to generate a PeopleSoft 8 XML test message based on a particular Message Definition.

To Generate a PeopleSoft 8 XML Message

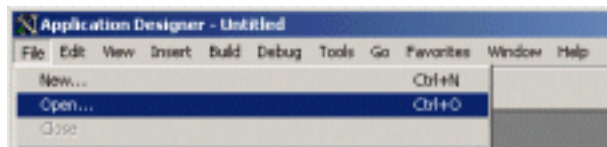
- 1 Sign onto People Tools.

Figure 51 People Tools Signon



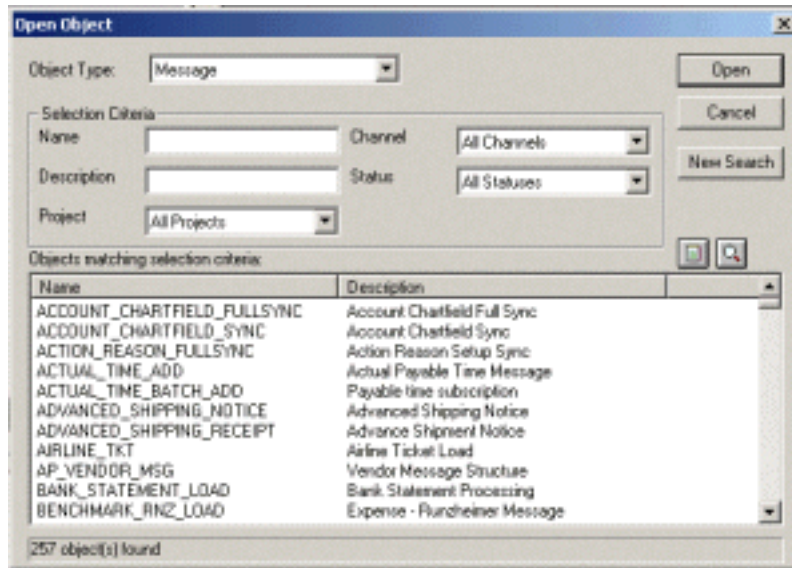
- 2 Log into the Application Designer

Figure 52 File Menu - Open



- 3 In the Application Designer, select **File** from the Menu and **Open** from the drop-down menu. The Open Object window appears (see Figure 53).

Figure 53 Open Object Window - Object Type Message



- 4 Select **Message** for the Object Type, and then click **Open**. A list of all available Message Definitions on the system appears in the bottom pane.
- 5 Find the desired Message Definition and double-click the selection; for example, **ADVANCED_SHIPPING_RECEIPT**. The Message window appears with **Message Structure** highlighted (see Figure 54).

Figure 54 ADVANCED_SHIPPING_RECEIPT Details

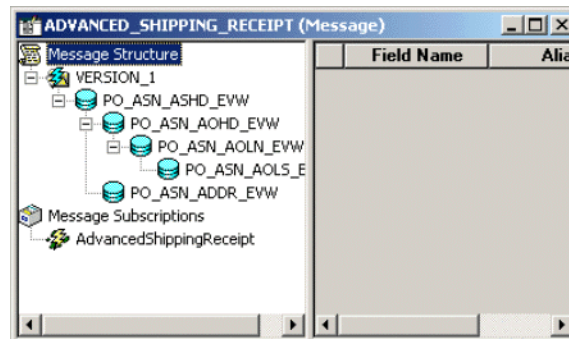
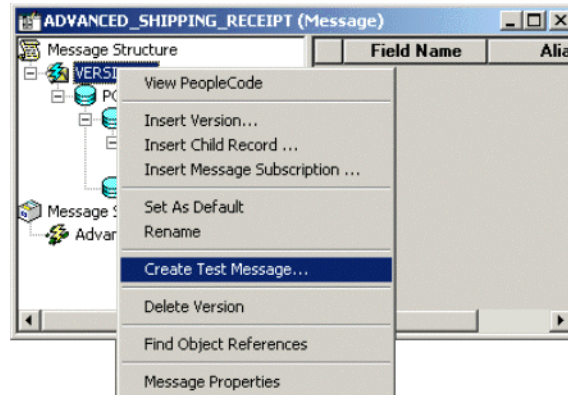
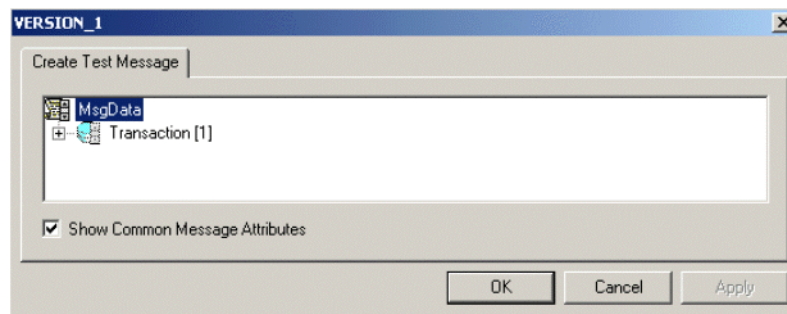


Figure 55 Message Window and Menu



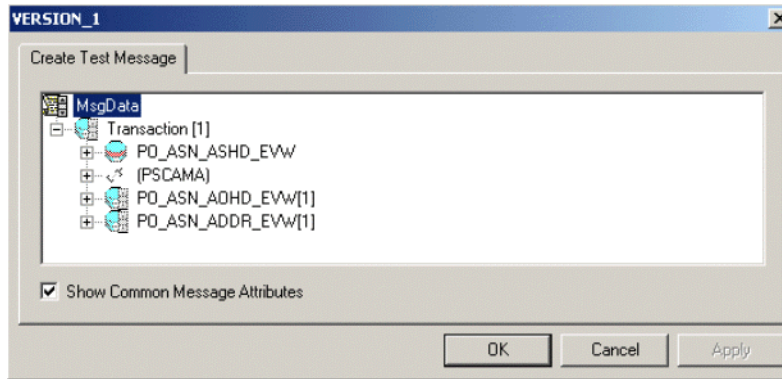
- 6 Highlight (left-click) the **Version_1** entry within the Message window (partially hidden by overlapping menu).
- 7 Right-click the **Version_1** entry to invoke the pop-up menu.
- 8 Select **Create Test Message** from the menu. The **Version_1** window appears showing the records contained in the message **ADVANCED_SHIPPING_RECEIPT**. (see Figure 56).

Figure 56 Version 1 - Create Test Message (1)



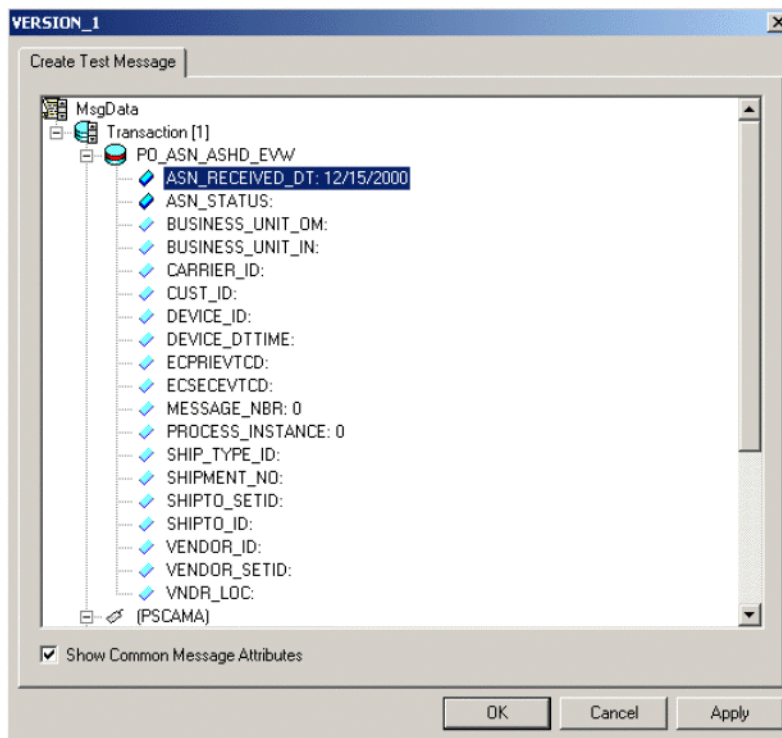
- 9 Expand the **Transaction** record by clicking on the (+) symbol. This reveals all sub-records within the transaction record (see Figure 57).

Figure 57 Version 1 - Create Test Message (2)



Records can nest to more than one level. Any record preceded by a (+) sign can be opened to verify the contents (see Figure 58).

Figure 58 Version 1 - Create Test Message (3)

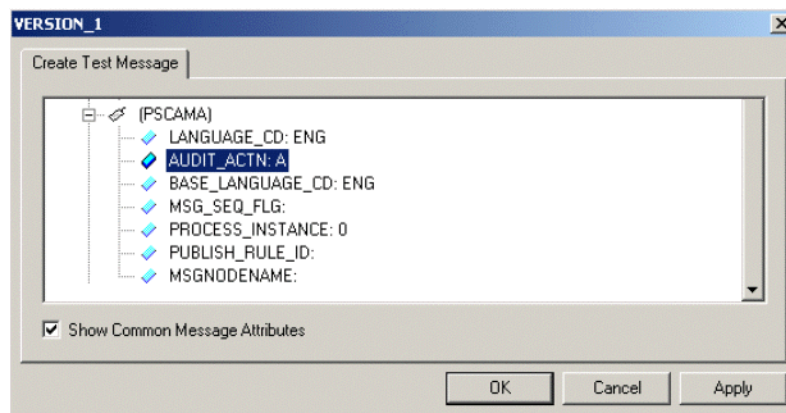


For purposes of this example, only the fields **ASN_RECEIVED_DT: 12/15/2000** and **ASN_STATUS:** have data contained within them.

Note: You need to know the data constraints and types for each field before proceeding (the following information is specific to PeopleSoft 8).

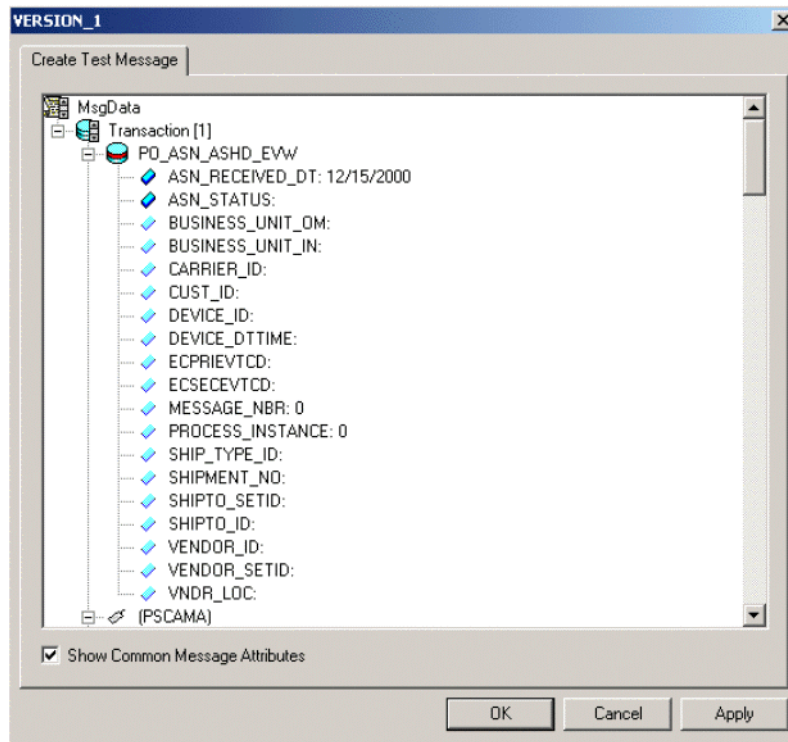
- If there are no constraints requiring you to populate all fields in a record, then generate a well-formed XML message by populating only one field in each record and sub-record.
 - If there are constraints, then all fields in each record and sub-record must be populated.
 - For most (but not necessarily all) Message Definitions, only one field is required to be populated with data. Also, some have values by default.
- 10 Enter data for the **PSCAMA** records (see Figure 59) as follows:
- A Double-click on a specific field. If the field displays empty, it is available for data input.
 - B Add the sample data.
 - C Continue populating all other records and sub-records.

Figure 59 Version 1 - Create Test Message (4)



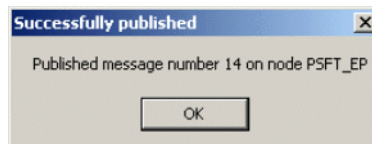
- 11 Continue entering data until all other required records and sub-records are populated, using the same method as above.
- 12 Once all records and sub-records of the message have been populated with data, click **Apply** to have the updates published to the **PSFT_EP** Message Node. Now the message can be viewed (see Figure 60).

Figure 60 Version 1 - Create Test Message (5)



- 13 A pop-up dialog box indicates successful publication (see Figure 61).

Figure 61 Success Dialog Box



- 14 Click OK to close the dialog box.

Extracting and Viewing the XML Test Message

The XML test message that you generated and published in the prior section can now be viewed by using a supported Web browser.

Note: See *PeopleSoft PeopleBooks* for more information on using the *PeopleSoft 8 Application*.

To View the XML Message

- 1 Within a supported web browser, log onto the PeopleSoft 8 Application.

Figure 62 PeopleSoft 8 Application Initial Page



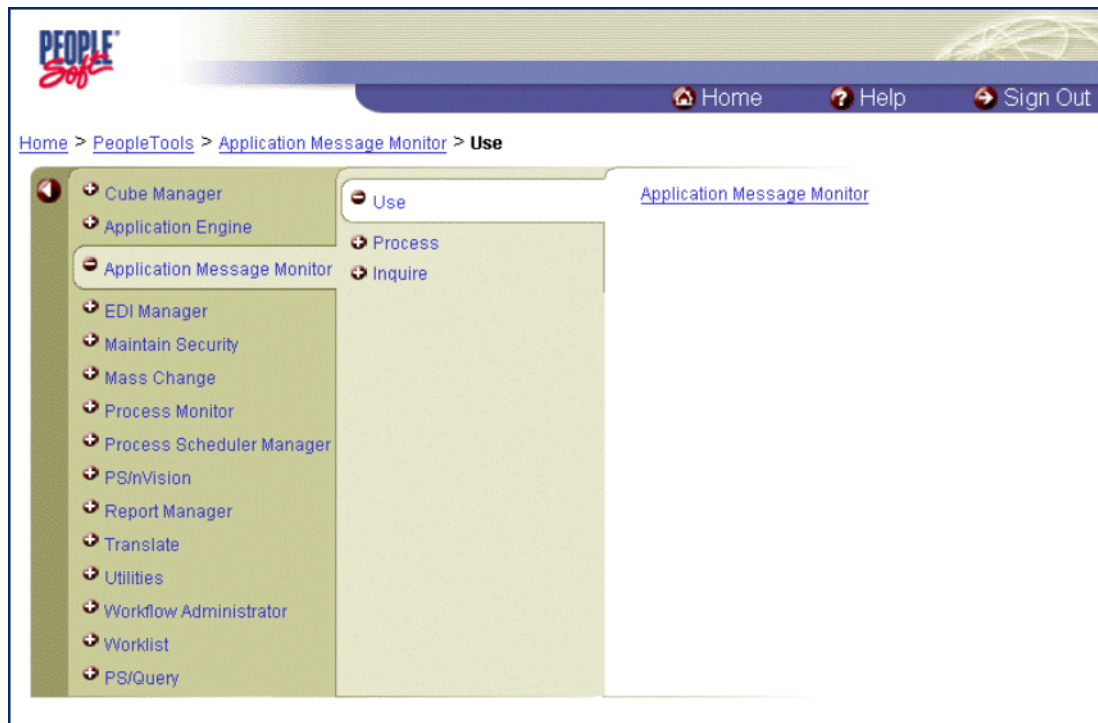
- 2 Once in PeopleSoft 8, scroll down to **PeopleTools** (see Figure 63).

Figure 63 PeopleSoft 8 Application Contents Page



3 Click on **PeopleTools** to open the PeopleTools application.

Figure 64 PeopleTools Directory Tree

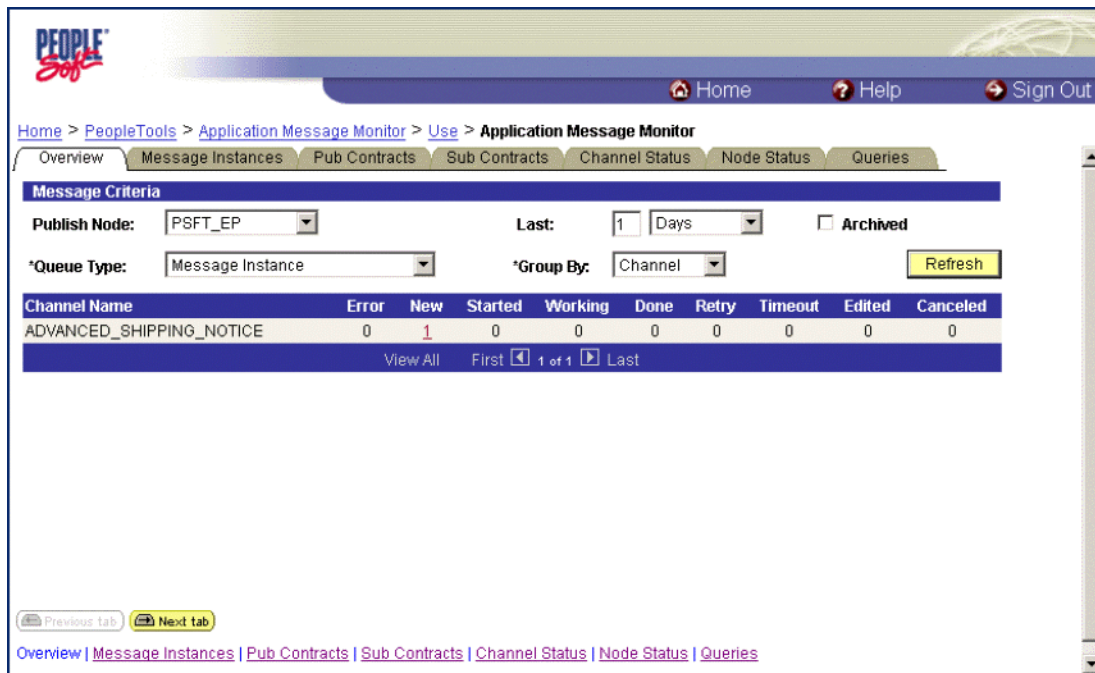


- 4 Follow the directory path

Application Message Monitor > Use > Application Message Monitor

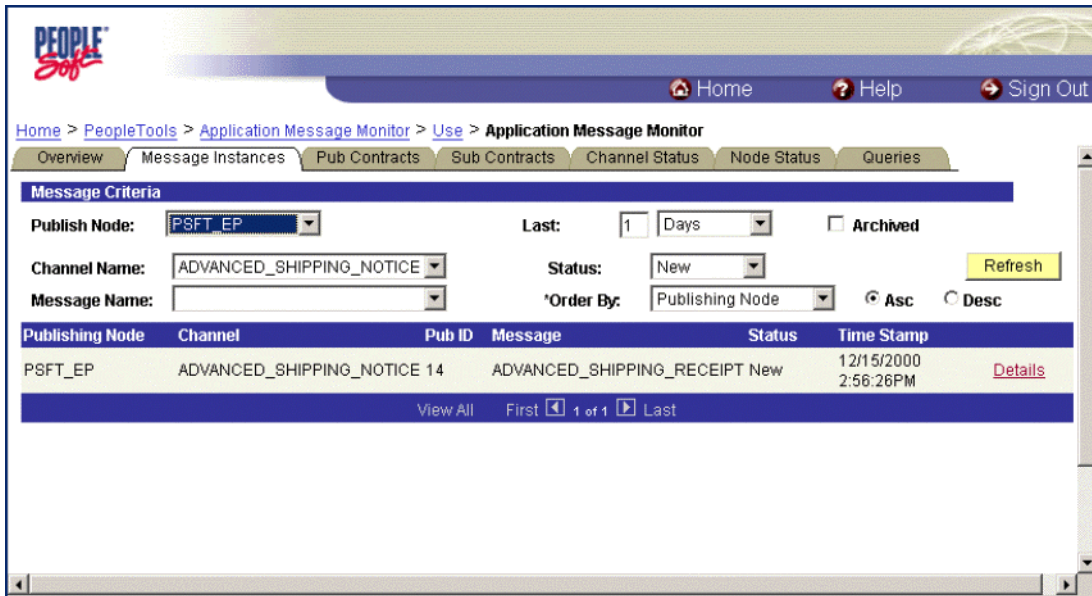
and click the hyperlink. The Application Message Monitor page opens to the Overview tab.

Figure 65 Application Message Monitor - Overview Tab



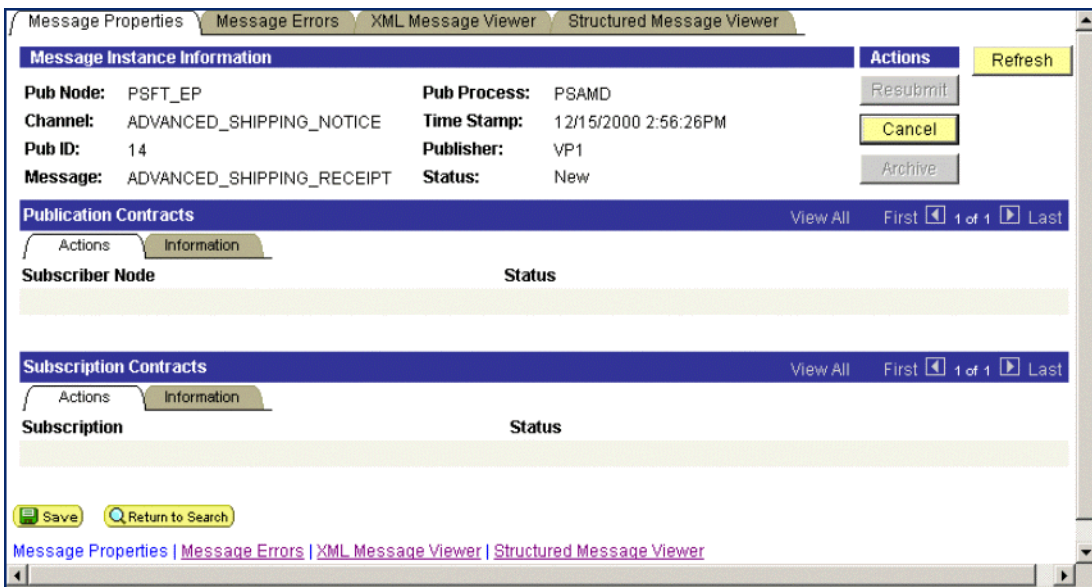
- 5 On the Overview tab, retrieve the list of published messages for the PSFT_EP Message Node.
 - A Within the Publish Node box, select PSFT_EP.
 - B Click the Refresh button, and the number of messages published for the selected grouping using the Create Test Message tool is indicated (in Figure 65, the Channel grouping was selected).
 - C Click the link indicated by the number of messages in the New, Done, or Working column (in Figure 65, the number 1 in the New column was selected). The Message Instances tab appears, showing a summary of the published messages (see Figure 66).

Figure 66 Application Message Monitor - Message Instances Tab



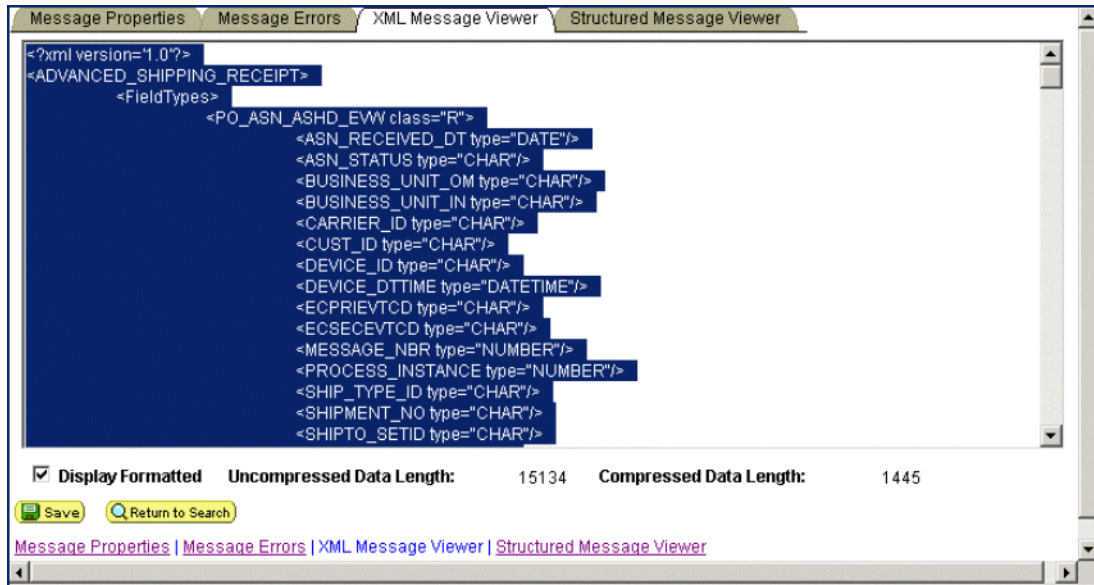
- Click on the **Details** link (far right side, beneath **Refresh** button) to view properties of the XML message that was published (see Figure 67).

Figure 67 Message Properties Tab



- Click the **XML Message Viewer** tab to review the message itself.

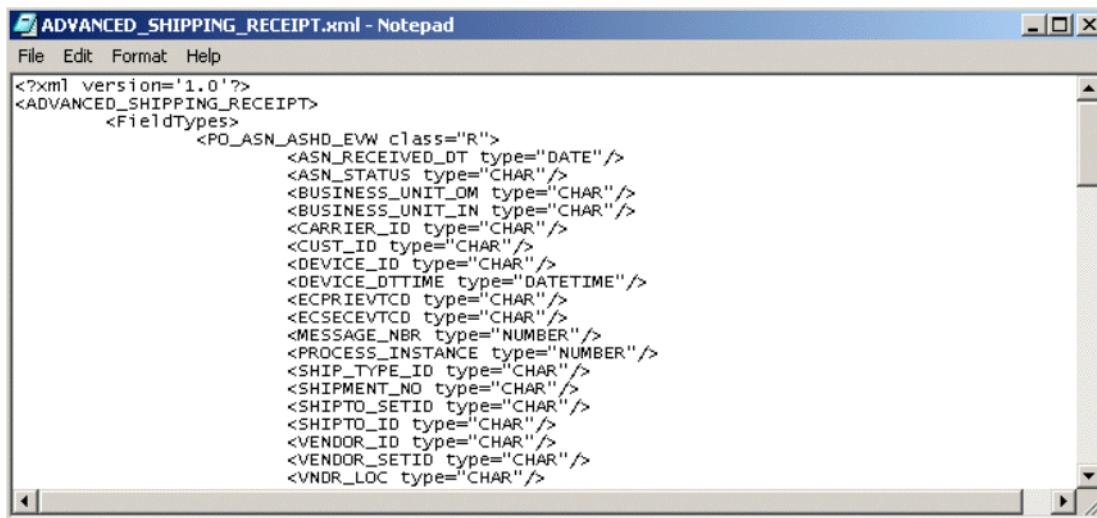
Figure 68 XML Message Viewer Tab



- 8 Save the message as an XML file.
 - A Select the entire XML message.
 - B Copy it to the clipboard.
 - C Paste the XML message into a text editor such as **Notepad** (Windows) and save it, with a **.xml** extension, to a temporary location (see Figure 69).

Note: Use the same naming convention used to name the Message Definition. This example shows that the XML Message **ADVANCED_SHIPPING_RECEIPT** was saved.

Figure 69 ADVANCED_SHIPPING_RECEIPT.xml



Generating a DTD for the XML File

The structure of the XML message now must be described in a Document Type Definition (DTD), from which a SeeBeyond Event Type Definition (ETD) is subsequently generated. PeopleSoft does not provide a DTD generation facility, but third-party utilities are available to accomplish this task.

A free, online DTD Generator utility is available at the following URL:

<http://www.hitsw.com/Xmltools/>

This utility is shown to illustrate the general procedure of DTD generation for the purposes of this User's Guide (see Figure 70 - Figure 71).

Note: SeeBeyond has no connection with, and does not support, this product.

Figure 70 Example DTD Generator (1)

The screenshot shows a web application titled "XML Tools and Utilities". It contains three main sections, each with a form for file selection and a "Generate" button:

- DTD to XML Schema:** A form with a text input labeled "DTD File:", a "Browse..." button, and a "Generate XML Schema" button.
- XML Document to XML Schema:** A form with a text input labeled "XML Document:", a "Browse..." button, and a "Generate XML Schema" button.
- XML Document to DTD:** A form with a text input labeled "XML Document:", a "Browse..." button, and a "Generate DTD" button.

At the bottom of the page, there is a footer note: "These conversion tools are based on work by Paul Tchistopolskii (www.pault.com) and use the SAXON [DTDGenerator](#), developed by Michael Kay."

- 1 In the **XML Document to DTD** section, click the **Browse** button to open a navigator window.

- 2 Locate the .xml file where you saved the XML Message, in this example,
`c:\temp\ADVANCED_SHIPPING_RECEIPT.xml`
- 3 Click **Open**, and the DTD Generator page reappears with the path and file displayed in the **XML Document** box (see Figure 71).

Figure 71 Example DTD Generator (2)

XML Tools and Utilities

Supply a file name and click the "Generate" button to display output in a browser page. Save the output to a file on your system.

Please ensure that any file you supply below does not contain references to other local files such as DTDs, external entities or XML schemas.

DTD to XML Schema

DTD File:

XML Document to XML Schema

XML Document:

XML Document to DTD

XML Document:

These conversion tools are based on work by Paul Tchistopolskii (www.pault.com) and use the SAXON [DTDGenerator](#) developed by Michael Kay.

- 4 Click the **Generate DTD** button to generate the DTD.
- 5 The DTD appears as shown in Figure 72.

Figure 72 Resulting DTD

```
Processing: C:\temp\ADVANCED_SHIPPING_RECEIPT.xml
<!ELEMENT ADDRESS1 EMPTY >
<!ATTLIST ADDRESS1 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS2 EMPTY >
<!ATTLIST ADDRESS2 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS3 EMPTY >
<!ATTLIST ADDRESS3 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS4 EMPTY >
<!ATTLIST ADDRESS4 type NMTOKEN #IMPLIED >

<!ELEMENT ADVANCED_SHIPPING_RECEIPT ( FieldTypes, MsgData ) >

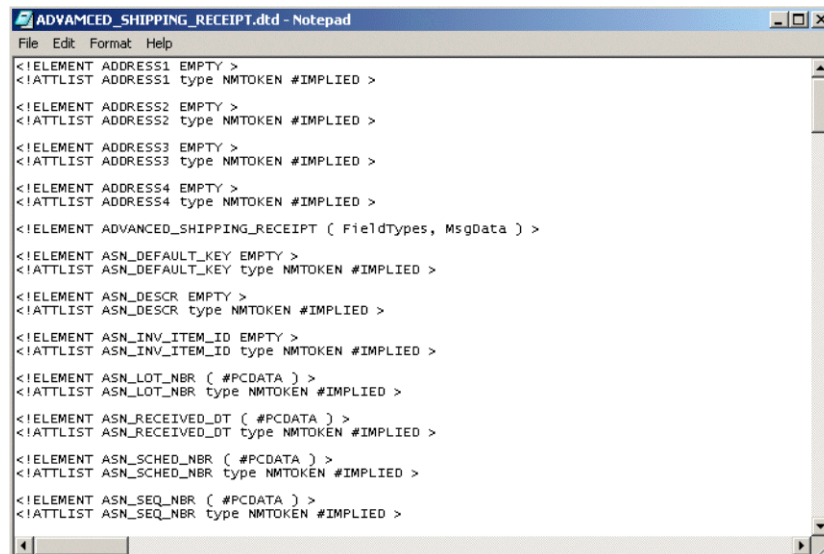
<!ELEMENT ASN_DEFAULT_KEY EMPTY >
<!ATTLIST ASN_DEFAULT_KEY type NMTOKEN #IMPLIED >

<!ELEMENT ASN_DESCR EMPTY >
<!ATTLIST ASN_DESCR type NMTOKEN #IMPLIED >
```

- 6 Save the message as an XML DTD file.
 - A Select only the DTD-related information (usually all information except the first line), as shown in Figure 72.
 - B Copy it to the clipboard.
 - C Paste the text into a text editor such as Notepad (Windows) and save it, with a .dtd extension, to a temporary location (see Figure 73).

Note: Use the same naming convention used to name the Message Definition (in the example, ADVANCED_SHIPPING_RECEIPT).

Figure 73 DTD File



```
ADVANCED_SHIPPING_RECEIPT.dtd - Notepad
File Edit Format Help
<!ELEMENT ADDRESS1 EMPTY >
<!ATTLIST ADDRESS1 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS2 EMPTY >
<!ATTLIST ADDRESS2 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS3 EMPTY >
<!ATTLIST ADDRESS3 type NMTOKEN #IMPLIED >

<!ELEMENT ADDRESS4 EMPTY >
<!ATTLIST ADDRESS4 type NMTOKEN #IMPLIED >

<!ELEMENT ADVANCED_SHIPPING_RECEIPT ( FieldTypes, MsgData ) >

<!ELEMENT ASN_DEFAULT_KEY EMPTY >
<!ATTLIST ASN_DEFAULT_KEY type NMTOKEN #IMPLIED >

<!ELEMENT ASN_DESCR EMPTY >
<!ATTLIST ASN_DESCR type NMTOKEN #IMPLIED >

<!ELEMENT ASN_INV_ITEM_ID EMPTY >
<!ATTLIST ASN_INV_ITEM_ID type NMTOKEN #IMPLIED >

<!ELEMENT ASN_LOT_NBR ( #PCDATA ) >
<!ATTLIST ASN_LOT_NBR type NMTOKEN #IMPLIED >

<!ELEMENT ASN_RECEIVED_DT ( #PCDATA ) >
<!ATTLIST ASN_RECEIVED_DT type NMTOKEN #IMPLIED >

<!ELEMENT ASN_SCHED_NBR ( #PCDATA ) >
<!ATTLIST ASN_SCHED_NBR type NMTOKEN #IMPLIED >

<!ELEMENT ASN_SEQ_NBR ( #PCDATA ) >
<!ATTLIST ASN_SEQ_NBR type NMTOKEN #IMPLIED >
```

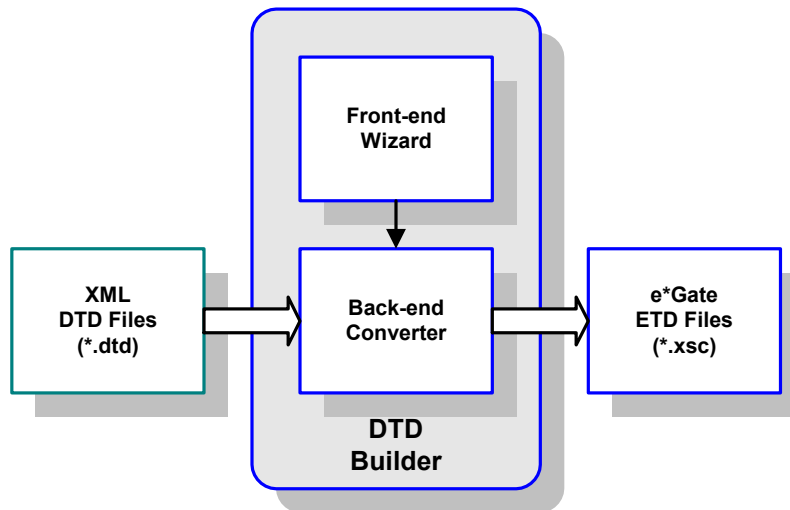
4.4.2 Generating an ETD from the DTD

Use the SeeBeyond DTD Builder to produce an Event Type Definition for the DTD that you have generated. See the SeeBeyond *XML Toolkit* documentation for detailed information on the DTD Builder.

Using the DTD Builder

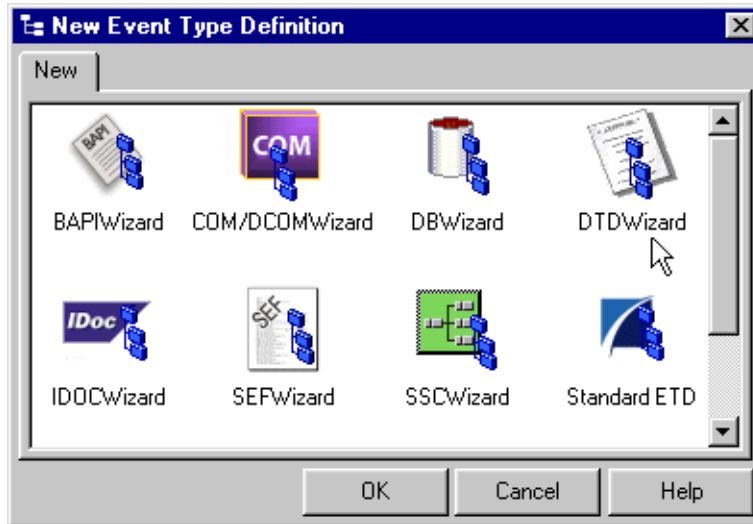
The ETD Editor contains a DTD Builder, which takes an XML DTD and converts it to a .xsc file.

Figure 74 DTD Builder



To access the DTD Builder's front-end Wizard, select the **New** option in the ETD Editor's **File** menu. The New Event Type Definitions window appears, displaying all installed ETD Wizards (see Figure 75).

Figure 75 New Event Type Definitions Window



To run the DTD Builder

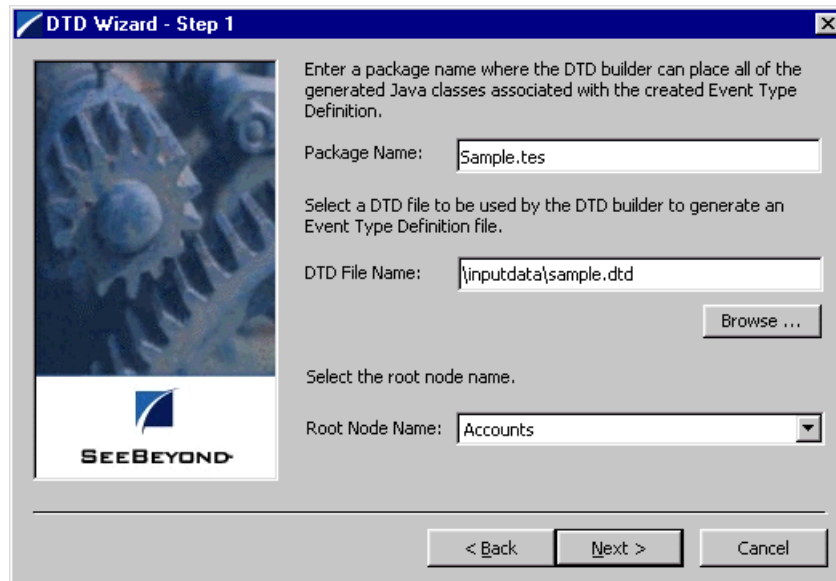
- 1 Invoke the DTD Wizard by clicking its icon.

Figure 76 DTD Wizard – Introduction



- 2 Read the instructions carefully, and click Next. Step 1 of the DTD Wizard dialog appears (see Figure 77).

Figure 77 DTD Wizard — Step 1



3 Enter the following information:

♦ **Java Package Name**

Type in the name you want to give the Java package, for example, **Sample.tes**. This name must conform to Java package name requirements. See the appropriate Java documentation for details.

♦ **DTD File Name**

Type in the name of the DTD file you want to convert. Click **Browse** to access an Open (file selection) dialog box, allowing you to choose the desired file.

♦ **Root Node Name**

This text box is a pull-down menu. Select the desired root node name from the menu. For more information on root nodes and ETDs, see the *e*Gate Integrator User's Guide*.

4 When you are finished, click **Next**. Step 2 of the DTD Wizard dialog appears (see Figure 78).

Figure 78 DTD Wizard — Step 2



- 5 Specify the options you want used by the DTD Builder.
 - ◆ Allow whitespace in EMPTY elements
 - ◆ Ignore #FIXED attributes
 - ◆ Ignore all attributes
 - ◆ Include XML declarations
 - ◆ Include DOC Type Reference (if selected, specify DTR name)
- 6 When you are finished, click Next. Step 3 of the DTD Wizard dialog appears (see Figure 79).

Figure 79 DTD Wizard — Step 3



- 7 Review the information you have entered in the Wizard. If it is correct, click **Finish** to generate a Java ETD (.xsc file) from the original DTD file.

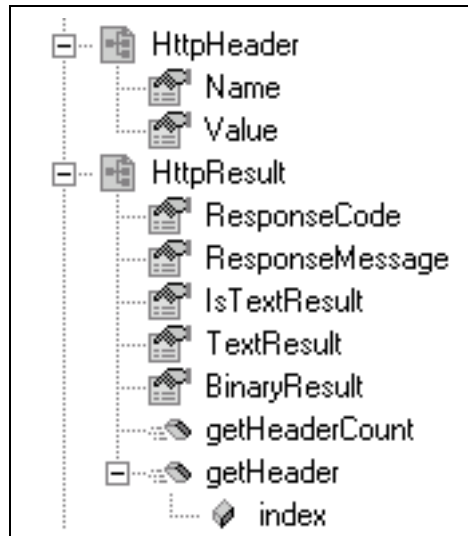
The Wizard closes, and the new ETD appears in the ETD Editor Main window. See the *e*Gate Integrator User's Guide* for details on how to use this editor, including an explanation of the information it shows.

- 8 To save the new ETD, click the **Save** button on the Toolbar or select the **Save** command from the **File** menu. A Save dialog box appears.
- 9 Select the desired directory location, give the new ETD your desired name, and click **Save**. The ETD Editor saves the new Java ETD.
- 10 You can continue to use the ETD Editor or select the **Close** command from the **File** menu to exit the GUI.

Note: *The ETD nodes created using the DTD Builder appear shaded in the ETD Editor, indicating that you cannot edit an ETD created by the Builder.*

After converting the DTD to an ETD, return to the e*Gate Schema Designer to verify the process (see Figure 80).

Figure 80 ETD Structure Example



4.5 Assigning ETDs to Event Types

After you have created the e*Gate system's ETD files, you can assign them to Event Types you have already created.

To assign ETDs to Event Types


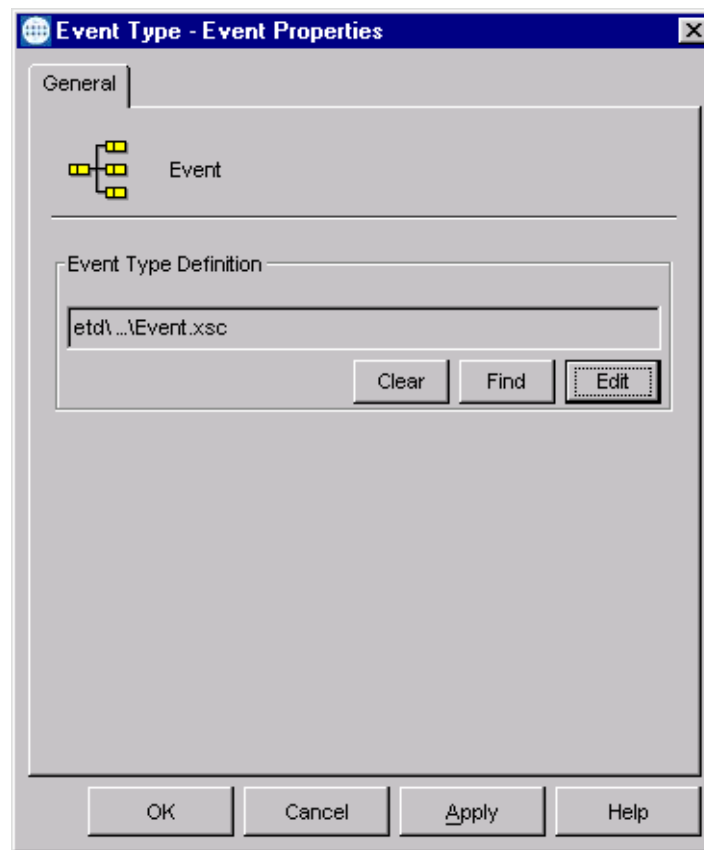
- 1 In the Schema Designer window, select the **Event Types** folder in the Navigator/Components pane.
- 2 In the Editor pane, select one of the Event Types you created.
- 3 Right-click on the Event Type and select **Properties** (or click  in the toolbar). The Event Type Properties dialog box appears. See Figure 81.

Figure 81 Event Type Properties Dialog Box



- 4 Under Event Type Definition, click **Find**.

The Event Type Definition Selection dialog box appears; it is similar to the Windows Open dialog box.

Note: Clicking **New** in the Event Type Properties dialog box opens the ETD Editor window, allowing you to create a new ETD.

- 5 Open the **etd** folder, then select the desired file name (.xsc).
- 6 Click **Select**. The file populates the Event Type Definition field.
- 7 To save any work in the properties dialog box, click **Apply** to enter it into the system.
- 8 When finished assigning ETDs to Event Types, click **OK** to close the properties dialog box and apply all the properties.

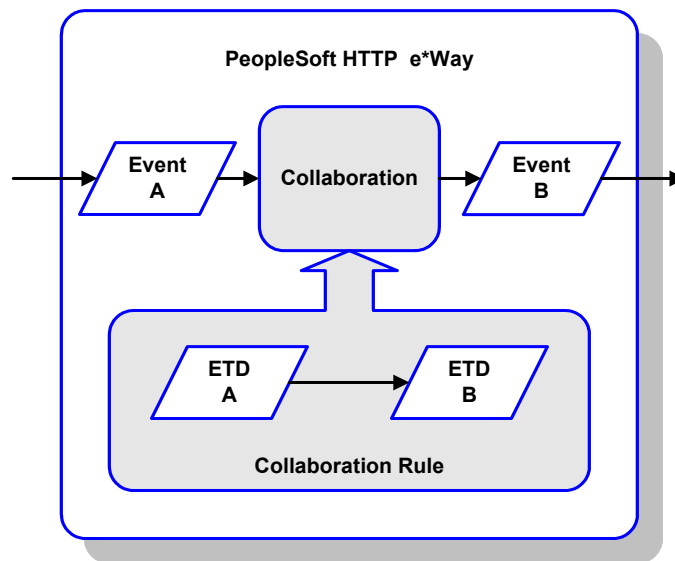
Each Event Type is associated with the specified Event Type Definition.

4.6 Defining Collaborations

After you have created the required Event Type Definitions, you must define a Collaboration to transform the incoming Event into the desired outgoing Event.

Collaborations are e*Way components that receive and process Event Types, then forward the output to other e*Gate components. Collaborations consist of the Subscriber, which “listens” for Events of a known type or from a given source, and the Publisher, which distributes the transformed Event to a specified recipient. The same Collaboration cannot be assigned to more than one e*Gate component.

Figure 82 Collaborations



4.6.1 The Java Collaboration Rules Editor

Java Collaborations are defined using the e*Gate Java Collaboration Rules Editor. Note that the Java Collaboration environment supports multiple source and destination ETDs. The file extension for Java Collaboration Rules is **.xpr**. See the *e*Gate Integrator User's Guide* for descriptions of the Java Collaboration Rules Editor and its use.

4.7 Creating Intelligent Queues

IQs are components that provide nonvolatile storage for Events within the e*Gate system as they pass from one component to another. IQs are *intelligent* in that they are more than just a “holding tank” for Events. They actively record information about the current state of Events.

Each schema must have an IQ Manager before you can add any IQs to it. You must create at least one IQ per schema for published Events within the e*Gate system. Note that e*Ways that publish Events externally do not need IQs.

For more information on how to add and configure IQs and IQ Managers, see the *e*Gate Integrator System Administration and Operations Guide*. See the *e*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User’s Guide* for complete information on working with IQs.

4.8 Using the e*Way

4.8.1 Publishing to PeopleSoft

Publishing data to PeopleSoft 8 using the PeopleSoft HTTP e*Way consists of four primary steps:

- 1 Compress the “business data” and base-64 encode it (optional).
- 2 Wrap the “business data” with PeopleSoft 8 XML “container” information.
- 3 Publish the wrapped XML data by using **http-post**.
- 4 Obtain the PeopleSoft 8 XML response by using **http-get-result-data** and check for successful publication.

The PeopleSoft HTTP e*Way does *not* contain any logic to create PeopleSoft 8 XML messages, nor does it contain any logic to map external data to PeopleSoft 8 XML messages. The e*Way merely accepts and publishes the raw PeopleSoft 8 XML message, irrespective of how the data may appear. The only difference between publishing to PeopleSoft 8.1 or 8.4 is the value for the **DefaultURL** configuration parameter.

XML Messages

The following is a description of the XML message, as described in the PeopleSoft 8 PeopleTools documentation, that is created by the e*Way prior to publishing to PeopleSoft 8:

```
<?xml version="1.0"?>
<request version="peopletools-version">
<to node="destination-node-name"/>
<from node="source-node-name" password="source-node-group-password"/>
<operations namespace="PublishSubscribe"
interface="PublishSubscribeSystem">
<invoke member="Publish">
<variable type="object" interface="Publication">
<publication>
<publishingnode>source-node-name</publishingnode>
<channel>channel-name</channel>
<publicationid>publication-id</publicationid>
<subchannel>subchannel</subchannel>
<subject>message-name</subject>
<subjectdetail>message-detail</subjectdetail>
<originatingnode>originating-node-name</originatingnode>
<publisher>publishing-operator-id</publisher>
<publicationprocess>publication-process</publicationprocess>
<publishtimestamp>publish-timestamp</publishtimestamp>
<status>publication-status</status>
<defaultdataversion>default-message-version-name</defaultdataversion>
<dataversions>
<publicationdataversion>
<version>VERSION_1</version>
<data length="identity-length" encoding="base64(deflate)"
encodedlength="base64-encoded-length(deflated-length)">deflated,
base-64 encoded XML data</data>
</publicationdataversion>
```

```
</dataversions>  
</publication>  
</variable>  
</invoke>  
</operations>  
</request>
```

Compressed (“deflated”), base-64 encoded XML data is the data that is to be published. The e*Way can be configured to compress and/or base-64 encode the data. The remainder of the XML data is PeopleSoft 8 “container” information, and most of these parameters are configurable in the e*Way. By default, the e*Way compresses and base-64 encodes the XML data. See PeopleSoft 8 PeopleTools documentation for detailed information on each of the elements of the XML message.

In response to a published message, PeopleSoft 8 Application Messaging Gateway returns a reply XML message with the following format:

```
<?xml version="1.0" ?>  
<reply>  
<operations namespace="PublishSubscribe"  
  interface="PublishSubscribeSystem">  
<invoke opnum="1" member="Publish">  
<return type="number">0</return>  
<variable type="object" interface="Publication">  
<publication>  
<publishingnode>publisher_node</publishingnode>  
<channel>published_channel</channel>  
<publicationid>published_id</publicationid>  
<publishtimestamp>date_time_of_publish</publishtimestamp>  
</publication>  
</variable>  
</invoke>  
</operations>  
</reply>
```

The e*Way checks the value of <return>, which can be:

- **0**, which means the message was delivered to PeopleSoft 8 successfully
- **6**, which means the message was already delivered successfully
- **8**, which means the message was *not* delivered successfully—in which case the e*Way marks the message for re-sending.

Compressing the XML Message

The PeopleSoft subscribing node can accept XML data in either a compressed and encoded, or an uncompressed and unencoded format.

- The compression algorithm that PeopleSoft accepts is base64(deflate). See *Uncompressing Messages* for more information.

- When compressed:
 - ♦ The encoding attribute of the <data> element contains information about how many bytes each of the compressed routines produced.
 - ♦ The length attribute of the <data> element is the number of Unicode characters when uncompressed.

Because PeopleTools 8 uses Unicode, two bytes represent a character that was previously represented by a single byte. One way to derive the Unicode byte length is to multiply the character length of the inflated contents of the <data> tags by 2 to get the correct length that PeopleTools can use to inflate to Unicode.

For example, the following code describes data that contains 4126 Unicode characters when uncompressed. When deflated, the size of the data is 532 bytes and when base64 encoded, becomes 712 bytes in size:

```
<data encoding="base64(deflate)" encodedlength="712 (532)"
      length="4126">
```

If you do not deflate and base64-encode the <data> element contents, then you must remove the <-xml version="1.0" -> processing instruction tag. Otherwise, the PeopleSoft application server fails (because XML does not support nested documents).

4.8.2 Subscribing to PeopleSoft 8.1

Subscribing to PeopleSoft 8.1 requires the use of the e*Gate API Kit and SeeBeyond's customized PeopleSoft 8 MUX subscription handler classes. The API Kit (Multiplexer) is used in conjunction with the subscription handler classes, written by SeeBeyond, to receive data from PeopleSoft 8 and publish the data to e*Gate.

To send data from PeopleSoft 8 to e*Gate, a servlet subscription handler called the **MuxHandler** is loaded into PeopleSoft's Application Messaging Gateway servlet. The **MuxHandler** implements the IPS Handler interface allowing it to intercept messages published from PeopleSoft 8 and directing them to e*Gate API Kit (Multiplexer e*Way). Publishing data to e*Gate consists of four steps:

- 1 Decompress and/or base-64 decode the "business data" (optional).
- 2 Instantiate an **IPMPReqReply** object and connect to the appropriate Multiplexer e*Way.
- 3 Send the "business data" to the Multiplexer e*Way by using **sendMessage**.
- 4 Ensure **sendMessage** completes successfully and send successful reply to PeopleSoft 8.

The **MuxHandler** subscription handler has eight configuration parameters:

- **Node Name** (Message Node that was created with the PeopleSoft 8 Application Designer, and is associated with the configured MUX Handler)
- **MUX Host**
- **MUX Port**
- **MUX Expire** (default is 10 sec.)
- **MUX Timeout** (default is 10000 msec.)

- **Uncompress** (default is YES)
- **Base64 Decode** (default is YES)
- **Log File** (the default value is C:\\yyyymmddhhmmss on Windows and /tmp/yyyymmddhhmmss on UNIX).

The **Uncompress** and **Base64 Decode** configurations can be used to decompress and/or decode the XML data before publishing to the Multiplexer e*Way. By default these are set to **YES**. Publishing encoded and compressed data is highly recommended since it reduces both the network bandwidth required and the processing load for the Multiplexer e*Way.

Note: **Base64 Decode** *must be selected if Uncompress is selected.*

If **Base64 Decode** and **Uncompress** are *not* selected, then the XML message, along with its “container” information, is sent to the Multiplexer e*Way as-is. In this case, you must configure the Multiplexer e*Way to decode and decompress the data when the data is received. See the *e*Gate API Kit User’s Guide* for details on the Multiplexer configurations.

The Publication ID and the Subject (Message Definition) of every message published is written to the log file as well as any errors that may have occurred. Note that the log file should be cleared out periodically to prevent it from growing too large. By default, the log file is named using the creation date and time of the subscription handler. You have the option of renaming the log file to something more meaningful to you. See [Installing the MUX Subscription Handler](#) on page 21 for more details.

4.8.3 Subscribing to PeopleSoft 8.4

Subscribing to PeopleSoft 8.4 makes use of the SeeBeyond JMS e*Way Connection. See the *SeeBeyond JMS Intelligent Queue User’s Guide* for information.

4.9 Sample Schema

A sample implementation, **psoft8AppMsg**, is available in the **\samples\ewpsoft8** directory of the e*Gate CD-ROM. The sample schema contains examples of both Publishing and Subscribing to PeopleSoft 8. It can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema. See **Optional Example Files** on page 26 for installation instructions.

Also in the **\samples\ewpsoft8** directory is a sample XML message, **INTERUNIT_EXPECTED_RECEIPT_16.xml**.

4.9.1 Publishing to PeopleSoft

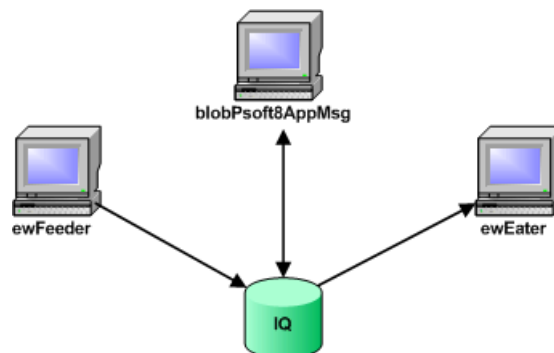
Before running this schema, set up PeopleSoft 8 for publishing to the **PSFT_EP** message node from the **SEEBEYOND** message node. See **Generating and Publishing an XML Test Message** on page 56, or the PeopleSoft 8 PeopleTools documentation, for details on how to create message nodes, activate a message, and set up the routing rules on a message channel using PeopleSoft Application Designer.

Components

The sample schema sets up one instance of the Multi-Mode e*Way and two instances of the File e*Way, with logical names as shown in the following table. It also sets up a standard SeeBeyond IQ. The component relationships are shown in Figure 83.

e*Way Type	Logical Name
Multi-Mode e*Way	blobPsoft8AppMsg
File e*Way	ewEater
	ewFeeder

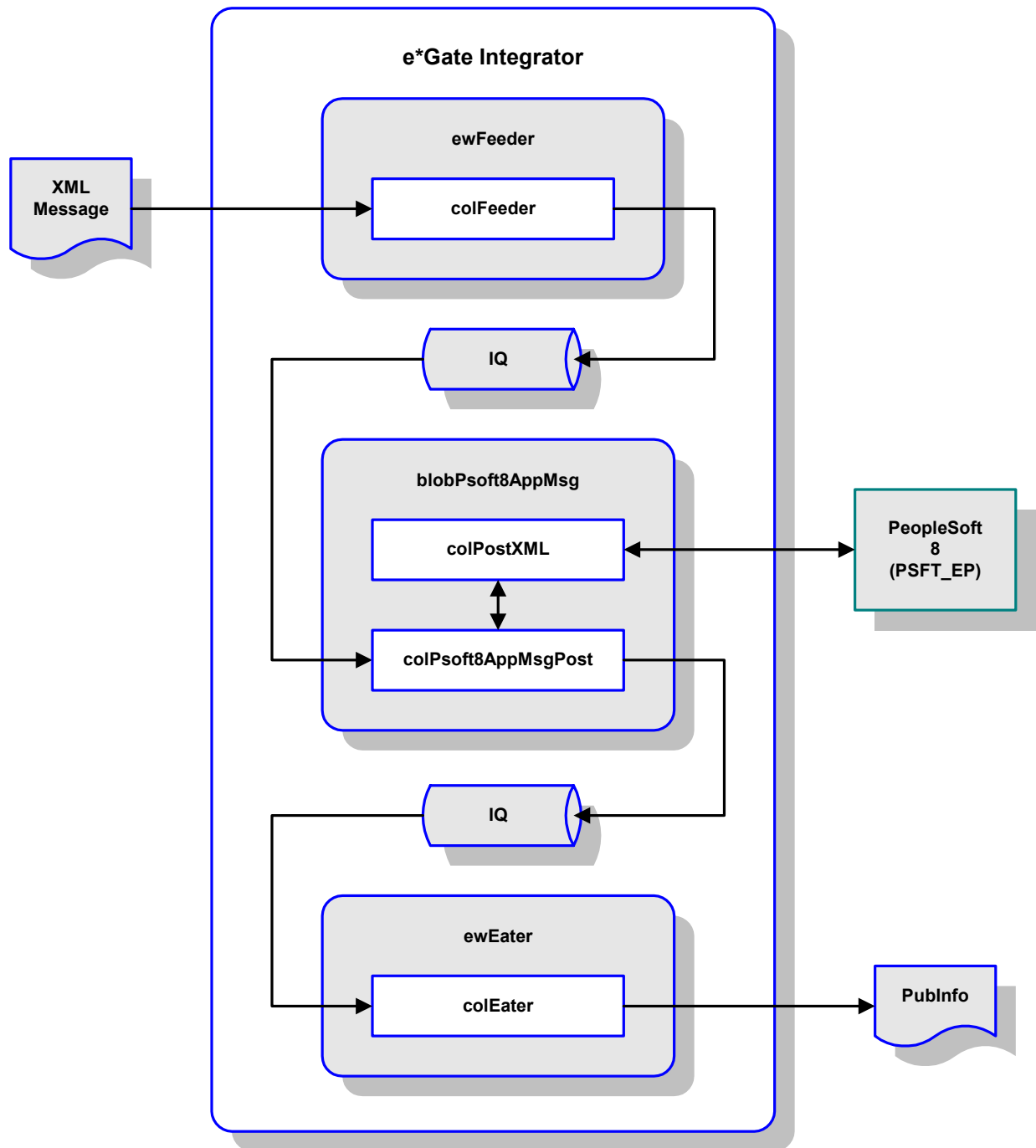
Figure 83 Components - Publishing Example



Operation

The schema is set up to send an INTERUNIT_EXPECTED_RECEIPT message using the INTERUNIT_EXPECTED_RECEIPT message channel. Included with this sample schema is a sample XML message, INTERUNIT_EXPECTED_RECEIPT_16.xml.

Figure 84 Operation - Publishing Example



As shown in Figure 84, the e*Way **ewFeeder** reads a file containing the PeopleSoft 8 data to send. A feeder Collaboration, **colFeeder**, subscribes to an external source and publishes to the IQ. A PeopleSoft Collaboration, **colPsoft8AppMsgPost**, subscribes to the feeder Collaboration via the IQ for the BLOB Event and then publishes the Event to the PeopleSoft 8 HTTP e*Way Connection. The Java collaboration **colPostXML** is then triggered, which:

- 1 Sets the XML message to send.
- 2 Sends the XML message by calling **sendMessage**.
- 3 Retrieves the publication information returned by PeopleSoft.

The Collaboration also publishes the publication information to a standard IQ. An eater Collaboration, **colEater**, subscribes to the PeopleSoft Collaboration for the publication information as a BLOB Event and publishes it to a file. An example of this publication information is shown in the following table.

Parameter	Value
Publication ID	11
Publishing Node	SEEBEYOND
PublishTimeStamp	2001-12-06T17:52:49.827000-0800

The PeopleSoft 8 e*Way Connection is configured to have the following publication information under the **ApplicationMessaging** section:

Parameter	Value
RequestVersion	8.13
ToNode	PSFT_EP
FromNode	SEEBEYOND
Channel	INTERUNIT_EXPECTED_RECEIPT
Subject	INTERUNIT_EXPECTED_RECEIPT

4.9.2 Subscribing to PeopleSoft 8.1

Before running this schema, set up PeopleSoft 8 for publishing to the MUX message node from the PSFT_EP message node. See [Generating and Publishing an XML Test Message](#) on page 56, or the PeopleSoft 8.13 PeopleTools documentation, for details on how to create message nodes, activate a message, and set up the routing rules on a message channel using PeopleSoft Application Designer.

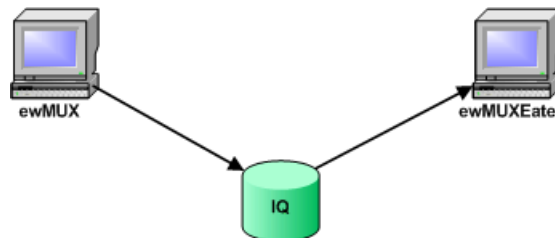
Also, see [Installing the MUX Subscription Handler](#) on page 21 for information on installing the MUX Subscription Handler and adding a MUX Subscription Handler using the PeopleSoft 8 Configuration Servlet.

Components

The sample schema sets up one instance of the Multiplexer e*Way (e*Gate API Kit) and one of the File e*Way, with logical names as shown in the following table. It also sets up a standard SeeBeyond IQ. The component relationships are shown in Figure 85.

e*Way Type	Logical Name
Multiplexer e*Way	ewMUX
File e*Way	ewMUXEater

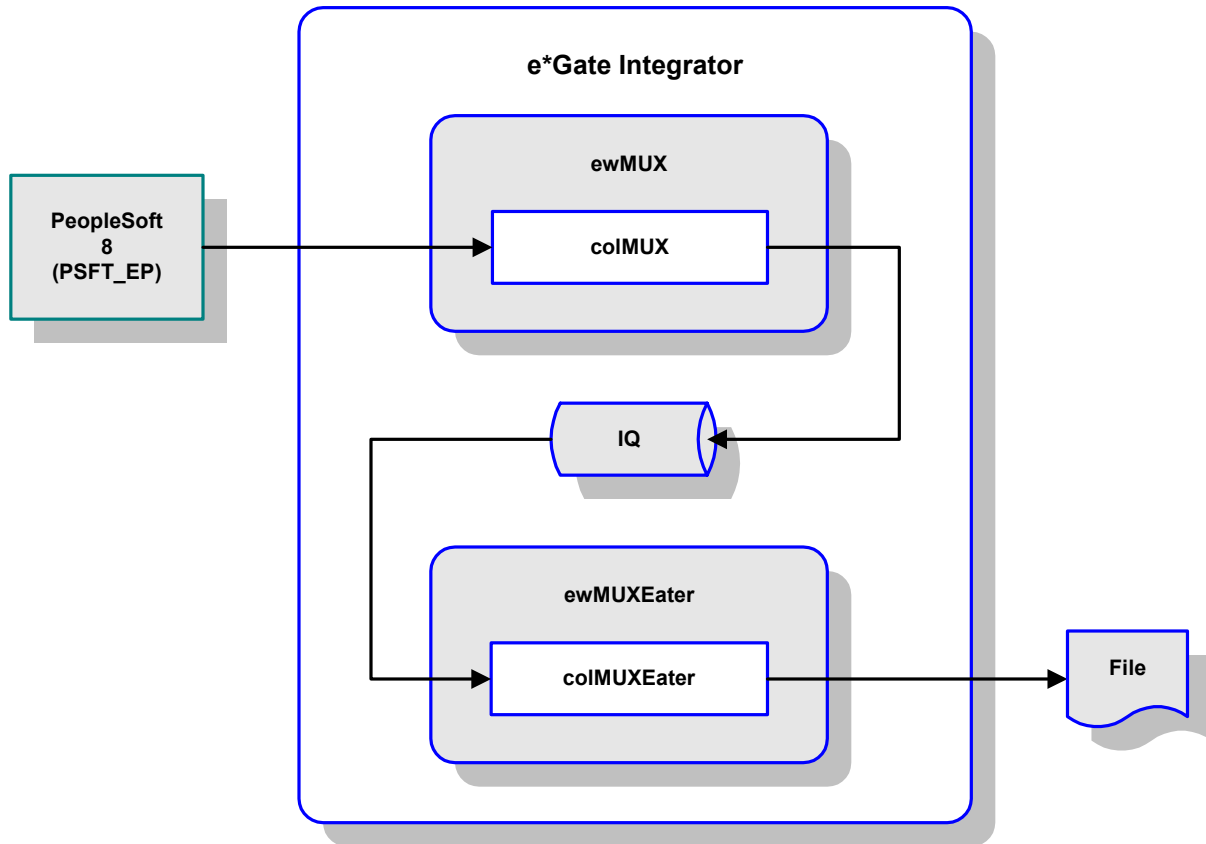
Figure 85 Components - Subscribing Example



Operation

The e*Way ewMUX, in combination with the MUX Subscription Handler, is used to subscribe to messages inbound to e*Gate. The MUX Subscription Handler intercepts messages published from PeopleSoft and routes the messages to the e*Way. A MUX collaboration, colMUX, subscribes to PeopleSoft and publishes the PeopleSoft 8 XML message into the IQ.

Figure 86 Operation - Subscribing Example



Another collaboration, *colMUXEater*, subscribes to that Event from the IQ and publishes it as a file. Since the Eater collaboration is **PassThru**, the data written to the file by *ewMUXEater* will also contain the MUX header information (which will be seen as binary). The publication information, such as the publication ID for the Event, is written to a log file by the MUX Subscription Handler. An example of an entry in the MUX Subscription Handler log is as follows:

```
2001.December.07 11:32:00.468 Pacific Standard Time:Publishing
publicationid [67] subject [AIRLINE_TKT].
```

This example log entry contains the following information.

Parameter	Value
Publication Name	AIRLINE_TKT
Publication ID	67
PublishTimeStamp	2001.December.07 11:32:00.468 PST

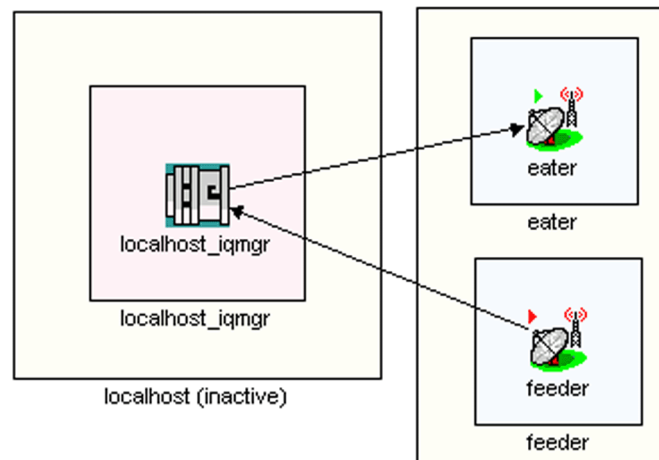
4.9.3 Subscribing to PeopleSoft 8.4

Before running this schema, **JMSQueue**, set up PeopleSoft 8.4 for publishing to the SeeBeyond JMS connector from the PeopleSoft JMS Target Connector. See [PeopleSoft 8.4/PeopleTools 8.42](#) on page 49.

Components

The sample schema sets up two instances of the File e*Way, with logical names **feeder** and **eater** (see Figure 87).

Figure 87 JMS Test Schema



Operation

The e*Way **feeder** is used to send an inbound message to e*Gate. A test collaboration, **jms_test**, subscribes to the message and publishes it to the JMS IQ. The e*Way **eater** subscribes to the IQ and saves the message as a file.

To test the PeopleSoft JMS Target Connector

- 1 Run the JMS test schema, **JMSQueue**.
- 2 Stop both e*Ways.
- 3 Use the PeopleSoft **StartSendMaster.bat** program under `\peoplesoft\c$\bea\wlserver6.1\config\peoplesoft\applications\PSIGW` to generate a test message to be sent to the JMS Server.
- 4 Click the **POST** button, which should post the message to the SeeBeyond JMS Server.
- 5 View the message using the e*Gate JMS Administrator.

e*Way Setup

This chapter describes the procedures required to customize the SeeBeyond e*Way Intelligent Adapter for PeopleSoft HTTP to operate within your production system.

5.1 Overview

After creating a schema, you must instantiate and configure the e*Way Intelligent Adapter for PeopleSoft HTTP to operate within the schema. A wide range of setup options allow the e*Way to conform to your system's operational characteristics and your facility's operating procedures.

The topics discussed in this chapter include the following:

[Setting Up the e*Way](#) on page 90

[Creating e*Way Connections](#) on page 98

[Using the e*Way Editor](#) on page 101

[Troubleshooting the e*Way](#) on page 104

5.2 Setting Up the e*Way

Note: The e*Gate Schema Designer GUI runs only on the Windows operating system.

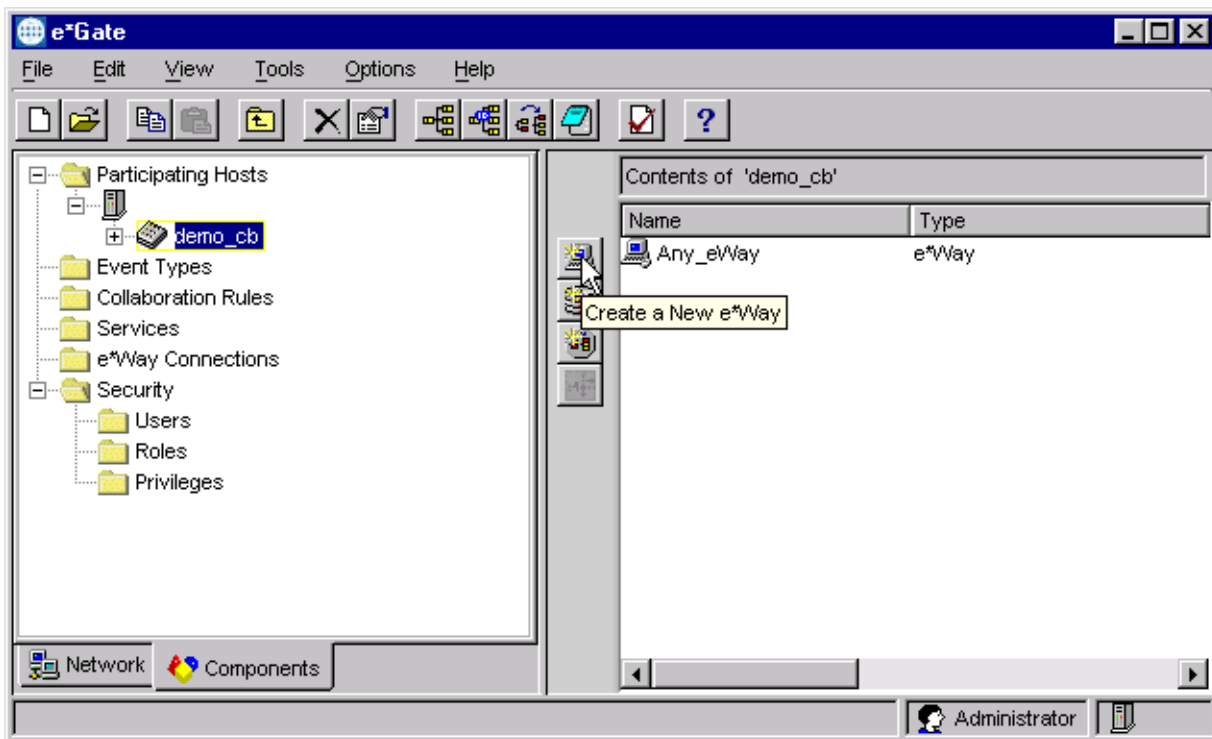
5.2.1 Creating the e*Way

The first step in implementing an e*Way is to define the e*Way component using the e*Gate Schema Designer.

To create an e*Way

- 1 Open the schema in which the e*Way is to operate.
- 2 Select the e*Gate Schema Designer Navigator's **Components** tab.
- 3 Open the host on which you want to create the e*Way.
- 4 Select the Control Broker you want to manage the new e*Way.

Figure 88 e*Gate Schema Designer Window (Components View)



- 5 On the Palette, click **Create a New e*Way**.
- 6 Enter the name of the new e*Way, then click **OK**.
- 7 All further actions are performed in the e*Gate Schema Designer Navigator's **Components** tab.

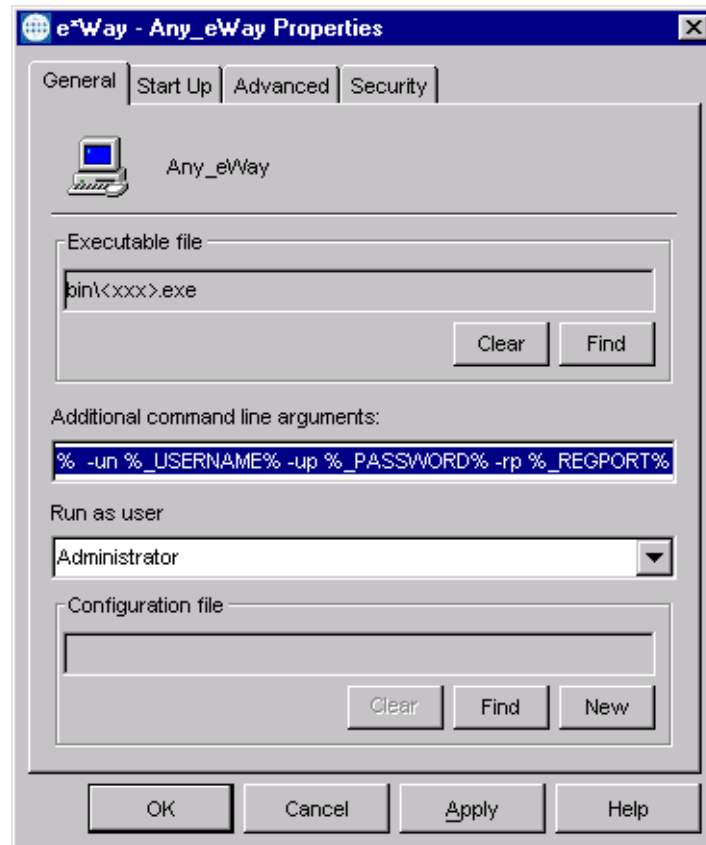
5.2.2 Modifying e*Way Properties

To modify any e*Way properties

- 1 Right-click on the desired e*Way and select **Properties** to edit the e*Way's properties. The properties dialog opens to the **General** tab (shown in Figure 89).

Note: The executable files used by this e*Way are **stceway.exe** and **stcewipmp.exe**.

Figure 89 e*Way Properties (General Tab)



- 2 Make the desired modifications, then click **OK**.

5.2.3 Configuring the Inbound e*Way

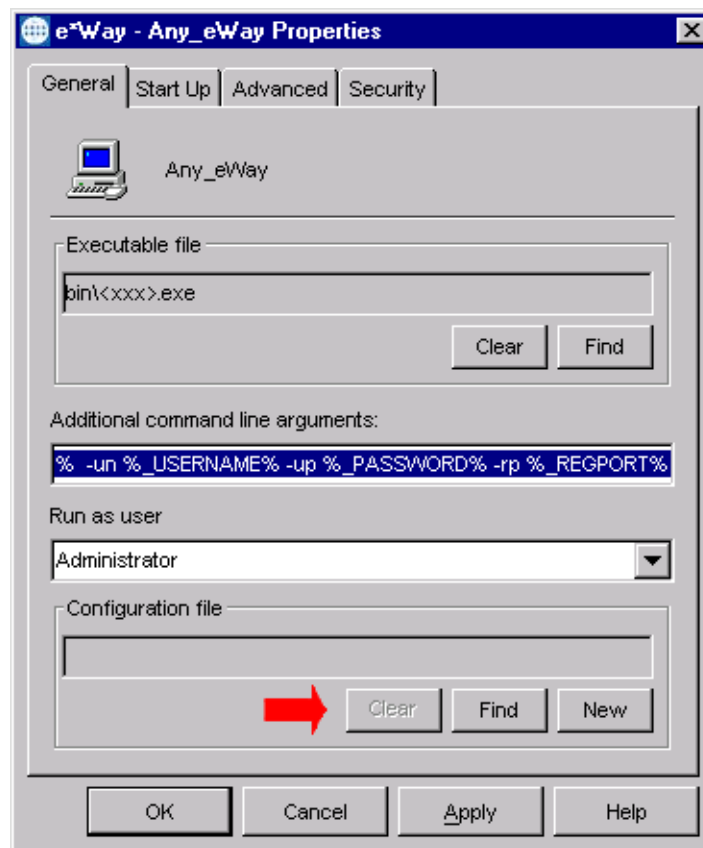
Note: This section applies only to an Inbound (PeopleSoft-to-e*Gate) e*Way. See [Configuring the Outbound e*Way](#) on page 93 and [Creating e*Way Connections](#) on page 98 for configuring an Outbound (e*Gate-to-PeopleSoft) e*Way.

The e*Way's default configuration parameters are stored in an ASCII text file, `stcewipmp.def`. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (`.cfg`) file.

To change the Inbound e*Way's configuration parameters

- 1 In the e*Gate Schema Designer's Component editor, select the desired e*Way.

Figure 90 e*Way Properties - General Tab



- 2 Under Executable File, click Find to locate `stcewipmp.exe`.
- 3 Under Configuration File, click New to create a new file or Find to select an existing configuration file. If you select an existing file, an Edit button appears, which you can click to edit the currently selected file.
- 4 You are now in the e*Way Configuration Editor (see [Using the e*Way Editor](#) on page 101 for more information).
- 5 See [e*Gate API Kit](#) on page 114 for the appropriate configuration parameters.

5.2.4 Configuring the Outbound e*Way

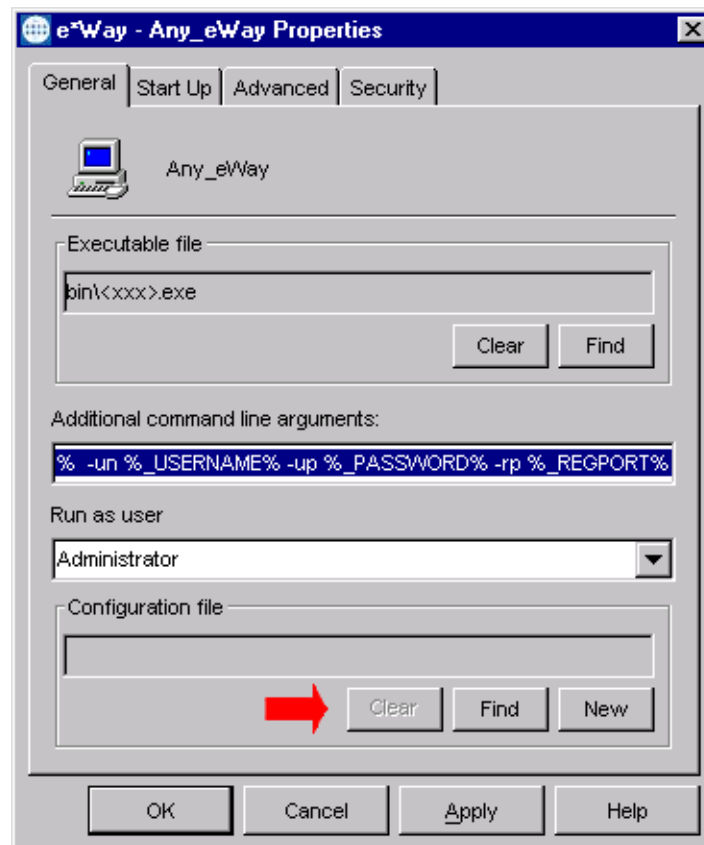
Note: This section applies only to an Outbound (e*Gate-to-PeopleSoft) e*Way; also see [Creating e*Way Connections](#) on page 98. See [Configuring the Inbound e*Way](#) on page 92 for configuring an Inbound (PeopleSoft-to-e*Gate) e*Way.

The e*Way's default configuration parameters are stored in an ASCII text file, `pssoft8appmsg.def`. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (`.cfg`) file.

To change the Outbound e*Way's configuration parameters

- 1 In the e*Gate Schema Designer's Component editor, select the desired e*Way.

Figure 91 e*Way Properties - General Tab



- 2 Under Executable File, click Find to locate `stceway.exe`.
- 3 Under Configuration File, click New to create a new file or Find to select an existing configuration file. If you select an existing file, an Edit button appears, which you can click to edit the currently selected file.
- 4 You are now in the e*Way Configuration Editor (see [Using the e*Way Editor](#) on page 101 for more information).
- 5 See [Multi-Mode e*Way](#) on page 113 for the appropriate configuration parameters.

5.2.5 Changing the User Name

Like all e*Gate executable components, e*Ways run under an e*Gate user name. By default, all e*Ways run under the **Administrator** user name. You can change this if your site's security procedures so require.

To change the user name

- 1 Display the e*Way's properties dialog.
- 2 On the **General** tab, use the **Run as user** list to select the e*Gate user under whose name you want this component to run.

See the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate security system.

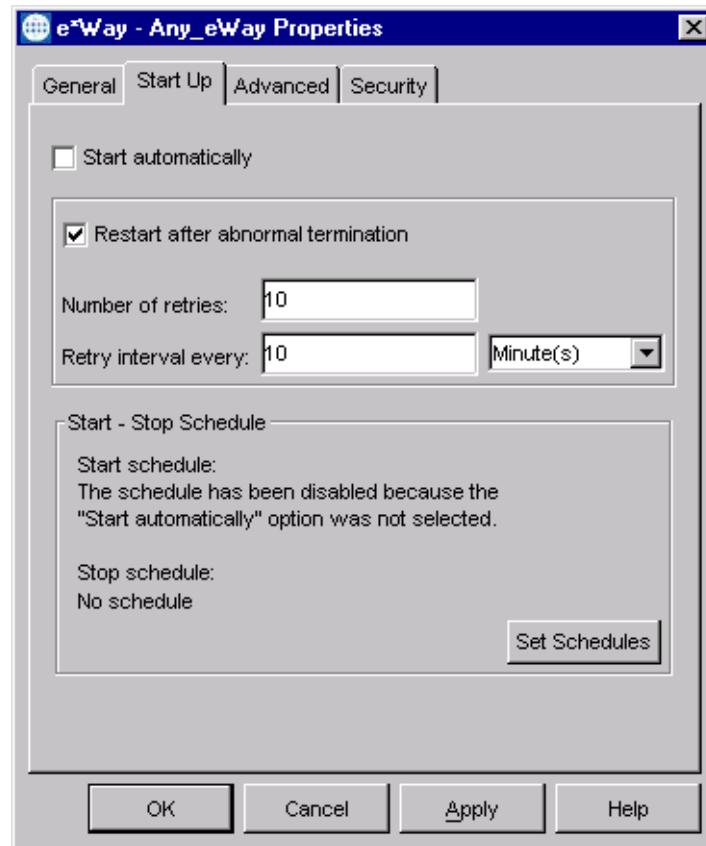
5.2.6 Setting Startup Options or Schedules

SeeBeyond e*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the e*Way automatically whenever the Control Broker starts.
- The Control Broker can start the e*Way automatically whenever it detects that the e*Way terminated execution abnormally.
- The Control Broker can start or stop the e*Way on a schedule that you specify.
- Users can start or stop the e*Way manually using an interactive monitor.

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 92). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 92 e*Way Properties (Start-Up Tab)



To set the e*Way's startup properties

- 1 Display the e*Way's properties dialog.
- 2 Select the **Start Up** tab.
- 3 To have the e*Way start automatically when the Control Broker starts, select the **Start automatically** check box.
- 4 To have the e*Way start manually, clear the **Start automatically** check box.
- 5 To have the e*Way restart automatically after an abnormal termination:
 - A Select **Restart after abnormal termination**.
 - B Set the desired number of retries and retry interval.
- 6 To prevent the e*Way from restarting automatically after an abnormal termination, clear the **Restart after abnormal termination** check box.
- 7 Click **OK**.

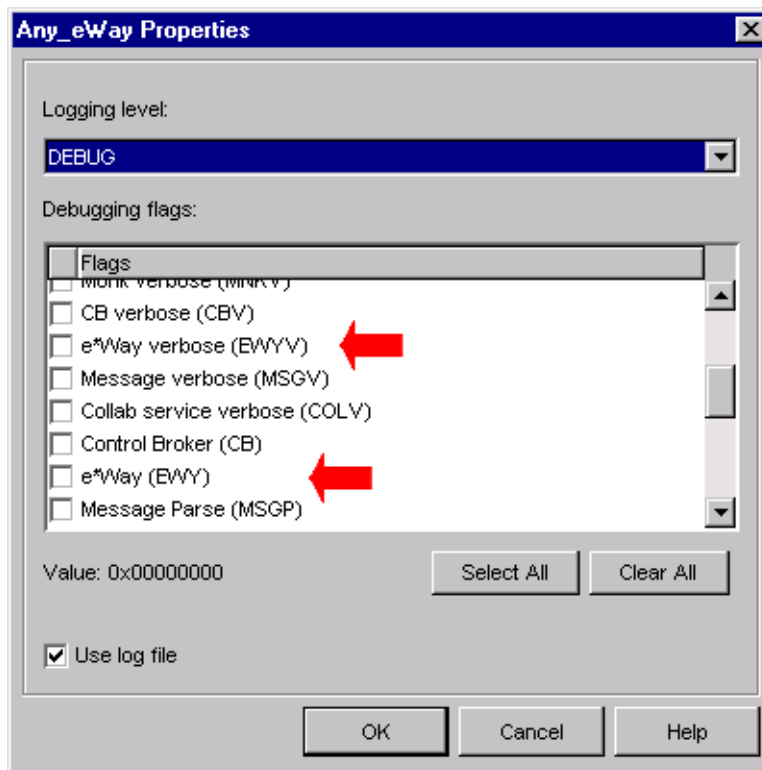
5.2.7 Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the e*Way and other e*Gate components.

To set the e*Way debug level and flag

- 1 Display the e*Way's Properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Log**, and the dialog window appears (see Figure 93).

Figure 93 e*Way Properties (Advanced Tab - Log Option)



- 4 Select **DEBUG** for the **Logging level**.
- 5 Select either **e*Way (EWY)** or **e*Way Verbose (EWYV)** for the **Debugging flag**. Note that the latter has a significant negative impact on system performance.
- 6 Click **OK**.

The other options apply to other e*Gate components and are activated in the same manner. See the *e*Gate Integrator Alert and Log File Reference* for additional information concerning log files, logging options, logging levels, and debug flags.

5.2.8 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a Monitoring Event to the Control Broker, which routes it to the Schema Manager and any other configured destinations.

- 1 Display the e*Way's properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Thresholds**.
- 4 Set the desired threshold options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference* for more information concerning threshold monitoring, routing specific notifications to specific recipients, or for general information about e*Gate's monitoring and notification system.

5.3 Creating e*Way Connections

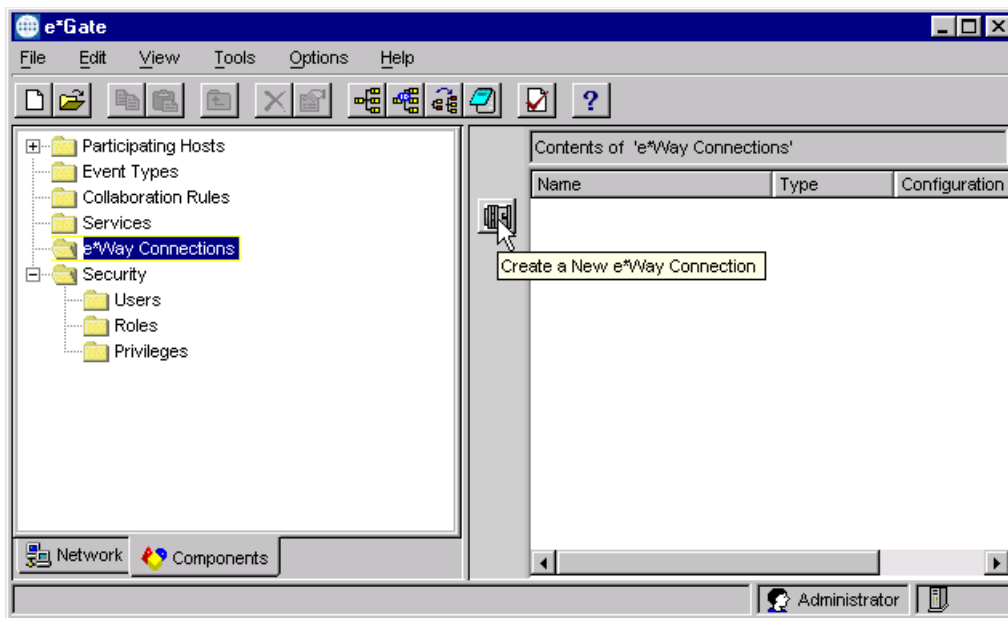
Note: This section applies only to an Outgoing (e*Gate-to-PeopleSoft) e*Way. See [Configuring the Inbound e*Way](#) on page 92 for configuring an Incoming (PeopleSoft-to-e*Gate) e*Way.

The e*Way Connections are created and configured in the Schema Designer.

To create and configure the Outgoing e*Way Connections

- 1 In the Schema Designer's Component editor, select the **e*Way Connections** folder.

Figure 94 Schema Designer - e*Way Connections Folder (1)




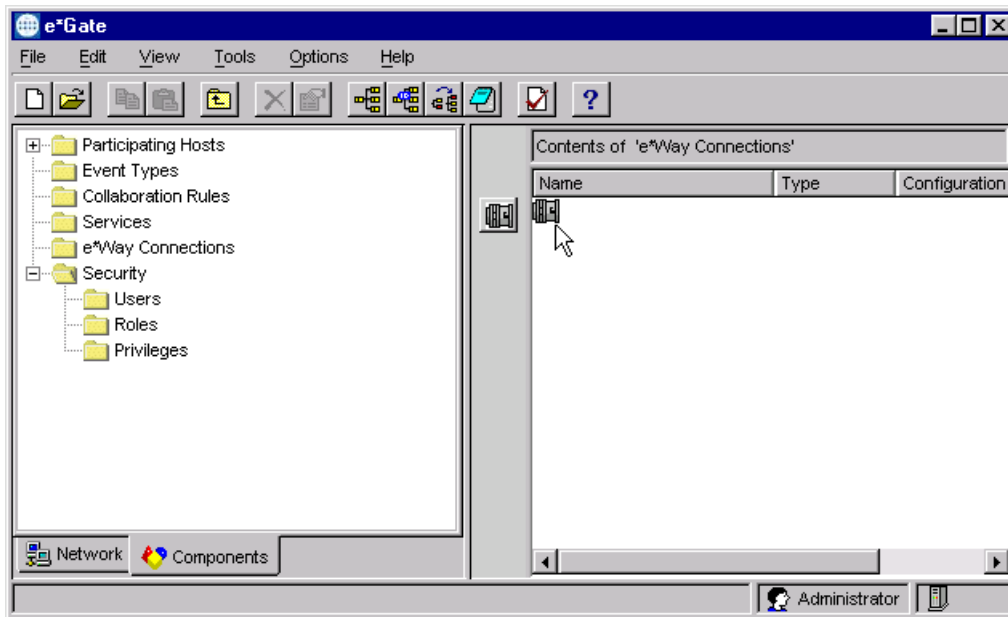
- 2 On the Palette, click the **Create a New e*Way Connection** button , which opens the **New e*Way Connection Component** dialog box.

Figure 95 New e*Way Connection Component Dialog Box



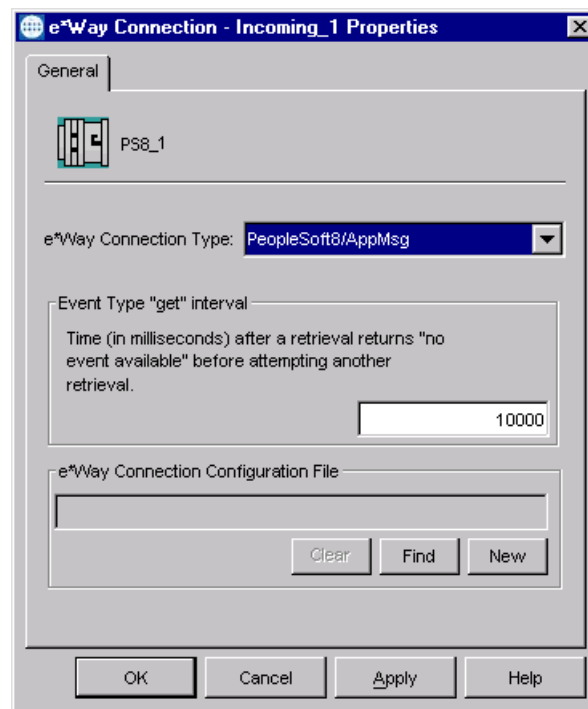
- 3 Enter a name for the e*Way Connection and click **OK**. The new e*Way Connection appears in the Schema Designer Contents pane.

Figure 96 Schema Designer - e*Way Connections Folder (2)



- 4 Double-click the new e*Way Connection icon to open the e*Way Connection Properties dialog box.

Figure 97 e*Way Connection Properties Dialog Box



- 5 From the e*Way Connection Type drop-down box, select PeopleSoft8/AppMsg for an Outgoing (e*Gate-to-PeopleSoft) e*Way Connection.

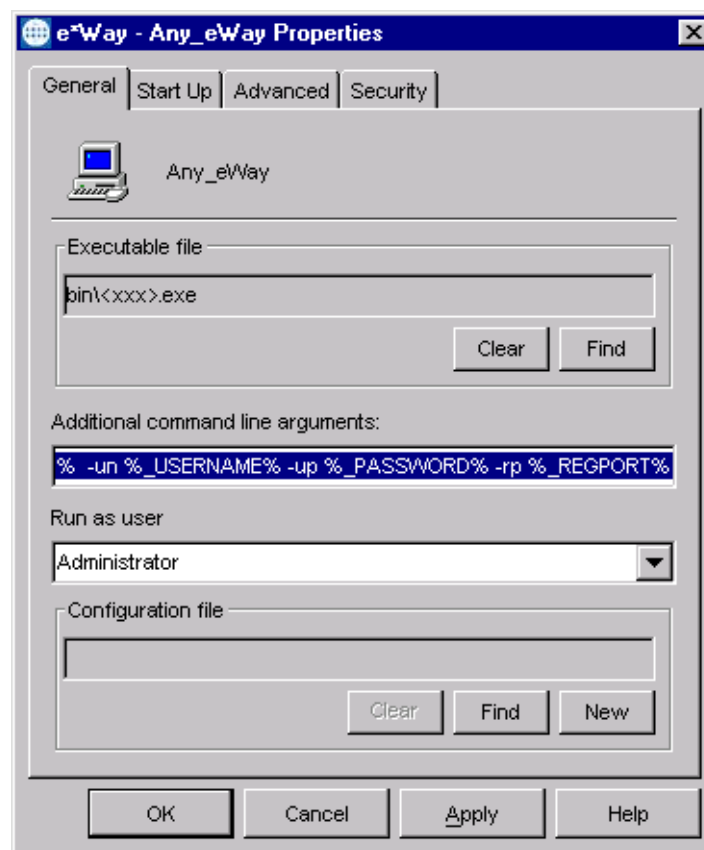
- 6 Enter the **Event Type “get” interval** in the dialog box provided (optional).
- 7 Click **New** to invoke the e*Way Connection Configuration File Editor, where you can create a new e*Way Connection Configuration File.

To modify the Outgoing e*Way Connections

- 1 In the e*Gate Schema Designer’s Component editor, select the e*Way you want to reconfigure and display its properties.

Note: The executable and default configuration files used by this e*Way are **stceway.exe** (e*Gate to PeopleSoft), **stcewimp.exe** (PeopleSoft to e*Gate), and **pssoft8appmsg.def**.

Figure 98 e*Way Properties - General Tab



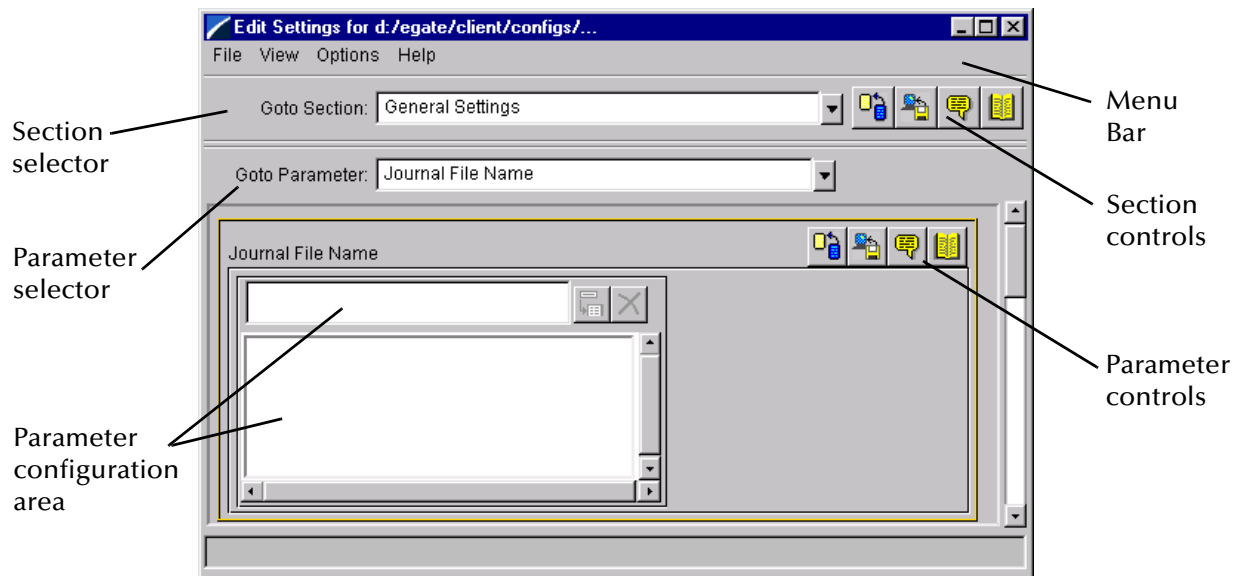
- 2 Under **Configuration File**, click **Find** to select an existing configuration file. An **Edit** button appears; click the button to edit the currently selected file.
- 3 You are now in the e*Way Configuration Editor (see [Using the e*Way Editor](#) on page 101 for more information).
- 4 See [e*Way Connections](#) on page 113 for the appropriate configuration parameters.

Note: You must restart the e*Way after modifying the e*Way connection.

5.4 Using the e*Way Editor

The e*Way's default configuration parameters are stored in an ASCII text file with a **.def** extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (**.cfg**) file.

Figure 99 The e*Way Configuration Editor







The e*Way Editor controls fall into one of six categories:

- The **Menu bar** allows access to basic operations (e.g., saving the configuration file, viewing a summary of all parameter settings, and launching the Help system)
- The **Section selector** at the top of the Editor window enables you to select the category of the parameters you wish to edit
- **Section controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section
- The **Parameter selector** allows you to jump to a specific parameter within the section, rather than scrolling
- **Parameter controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter
- **Parameter configuration controls** enable you to set the e*Way's various operating parameters

5.4.1 Section and Parameter Controls

The section and parameter controls are shown in Table 8 below.

Table 8 Parameter and Section Controls

Button	Name	Function
	Restore Default	Restores default values
	Restore Value	Restores saved values
	Tips	Displays tips
	User Notes	Enters user notes



Note: The section controls affect all parameters in the selected section, whereas the parameter controls affect only the selected parameter.

5.4.2 Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Option buttons
- Selection lists, which have controls as described in Table 9

Table 9 Selection List Controls

Button	Name	Function
	Add to List	Adds the value in the text box to the list of available values.
	Delete Items	Displays a “delete items” dialog box, used to delete items from the list.

5.4.3 Command-line Configuration

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.

5.4.4 Getting Help

To launch the e*Way Editor's Help system

From the **Help** menu, select **Help topics**.

To display tips regarding the general operation of the e*Way

From the **File** menu, select **Tips**.

To display tips regarding the selected Configuration Section

In the **Section Control** group, click .

To display tips regarding the selected Configuration Parameter

In the **Parameter Control** group, click .

Note: *“Tips” are displayed and managed separately from the Help system that launches from the Toolbar's Help menu. You cannot search for Tips within the Help system, or view Help system topics by requesting Tips.*

For detailed descriptions and procedures for using the e*Way Configuration Editor, see the *e*Gate Integrator User's Guide*.

5.5 Troubleshooting the e*Way

In the initial stages of developing your e*Gate Integrator system administration system, most problems with e*Ways can be traced to configuration.

5.5.1 Configuration Problems

In the Schema Designer

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rules script employed by this e*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that *feed* this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way *feeds* properly configured, and are they subscribing to the appropriate Events correctly?

In the e*Way Editor

- Check that all configuration options are set appropriately.
- Check that all settings you changed are set correctly.
- Check all required changes to ensure they have not been overlooked.
- Check the defaults to ensure they are acceptable for your installation.

On the e*Way's Participating Host

- Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e*Way's Collaborations publish.
- Check that your *path* environmental variable includes the location of the PeopleSoft HTTP dynamically-loaded libraries. The name of this variable on the different operating systems is:
 - ♦ PATH (Windows)
 - ♦ LD_LIBRARY_PATH (Solaris/Compaq)
 - ♦ LIBPATH (AIX)
- Check that your *classpath* environmental variable includes the location of the PeopleSoft HTTP e*Way Connection Java classes. The name of this variable can vary.

In the PeopleSoft Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.

5.5.2 System-related Problems

- Check that the connection between the external application and the e*Way is functioning appropriately.
- Once the e*Way is up and running properly, operational problems can be due to:
 - ♦ External influences (network or other connectivity problems).
 - ♦ Problems in the operating environment (low disk space or system errors)
 - ♦ Problems or changes in the data the e*Way is processing.
 - ♦ Corrections required to Collaboration Rules scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e*Way log file. See the *e*Gate Integrator Alert and Log File Reference Guide* for an extensive explanation of log files, debugging options, and using the e*Gate monitoring system to monitor operations and performance.

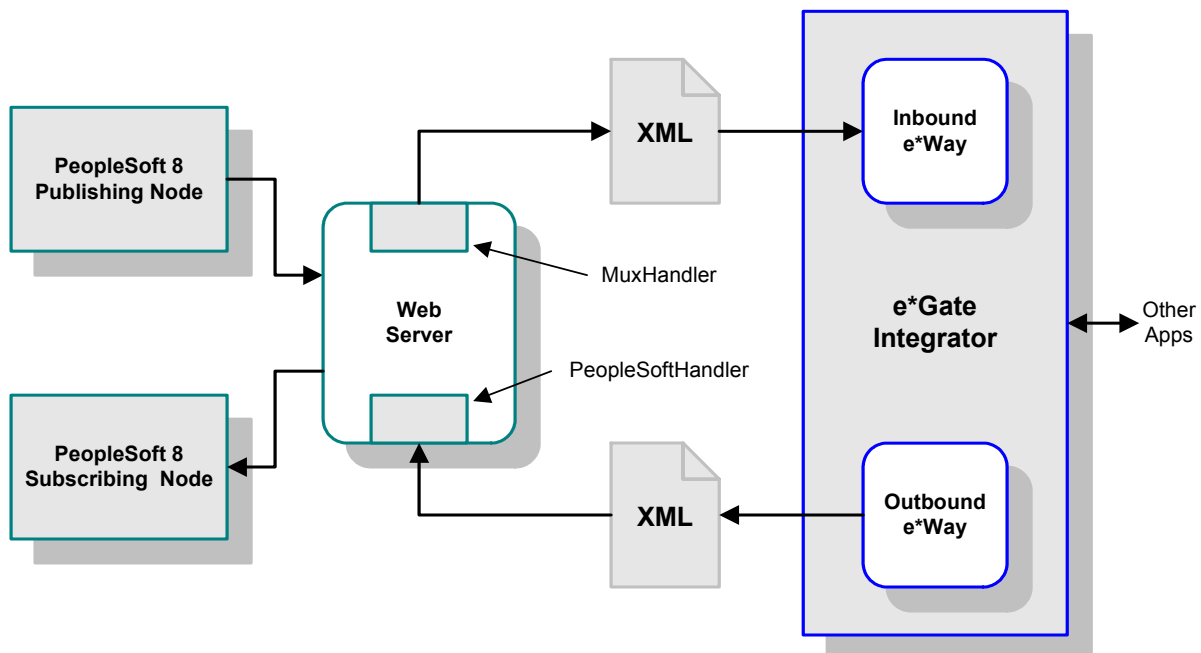
Operational Overview

This chapter contains an overview of the architecture and basic internal processes of the Java e*Way Intelligent Adapter for PeopleSoft (HTTP).

6.1 Introduction

The outbound e*Way uses Java methods to exchange data with the external system, package data as e*Gate *Events*, send those Events to Collaborations, and manage the connection between the e*Way and the external system. The inbound e*Way uses MUX components from the e*Gate API Kit for data exchange. These topics are covered in detail in the *e*Gate Integrator User's Guide*.

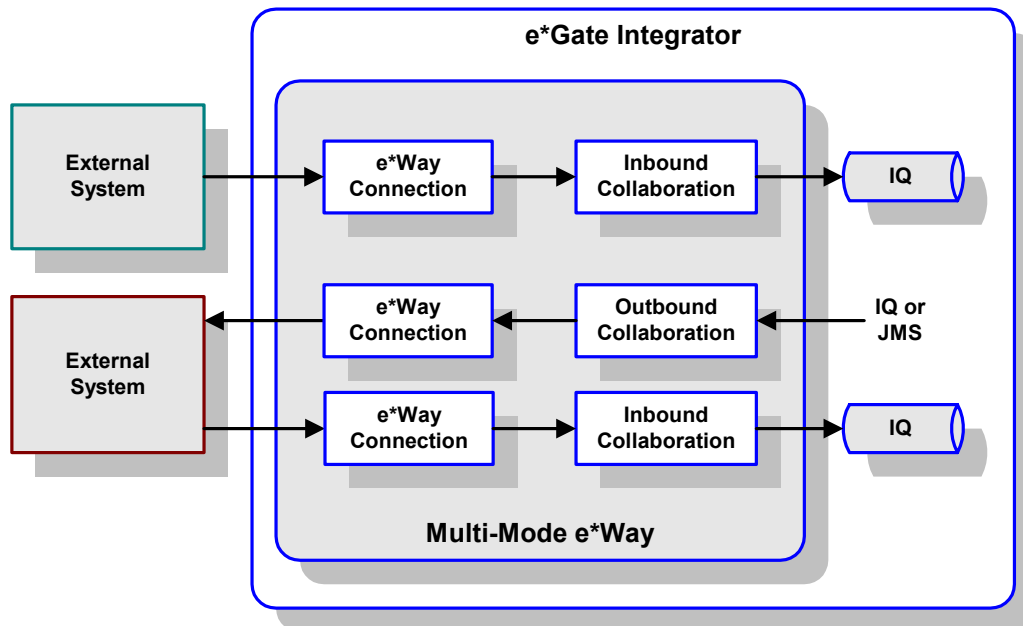
Figure 100 PeopleSoft 8 - e*Gate Integration



6.2 Multi-Mode e*Way

The PeopleSoft HTTP e*Way is based on the SeeBeyond Multi-Mode e*Way, which is a multi-threaded component forming an Intelligent Adapter for e*Gate Integrator to exchange information with multiple external systems. The e*Way connects to one or more external systems by means of *e*Way Connections*, each of which must be configured for the specific external system to which it connects (see Figure 101).

Figure 101 Multi-Mode e*Way



Each e*Way performs one or more *Collaborations* (see [Collaborations and Event Type Definitions](#) on page 108). Bidirectional data flow requires at least two Collaborations, one *Inbound* and one *Outbound*, as shown in Figure 101. Each Collaboration processes a stream of messages, or *Events*, containing data or other information.

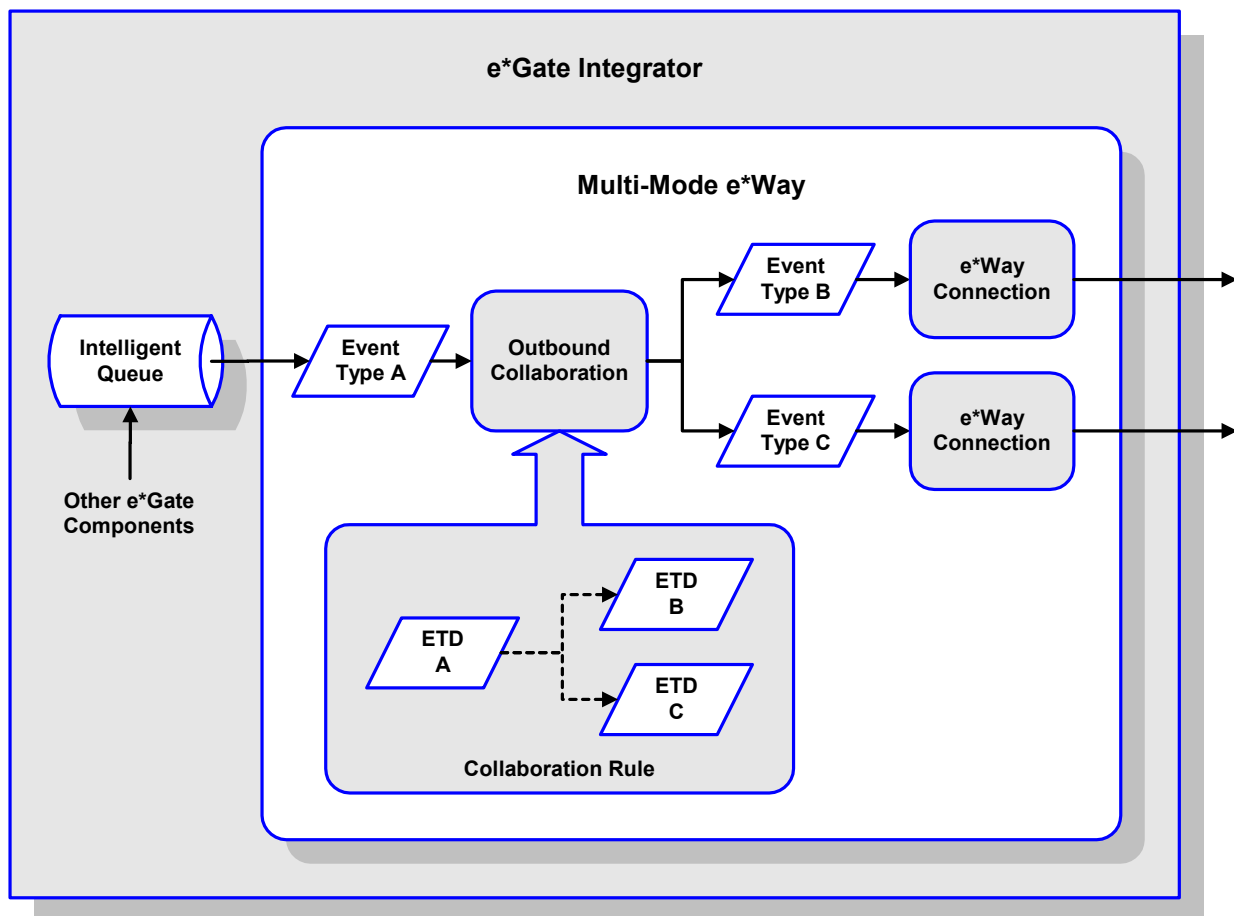
Each Collaboration that publishes its processed Events internally (within e*Gate Integrator) requires one or more *Intelligent Queues* (IQs) to receive the Events. Any Collaboration that publishes its processed Events only to an external system, which is the case with the PeopleSoft HTTP e*Way, *does not* require an IQ to receive Events.

6.3 Collaborations and Event Type Definitions

Collaborations execute the business logic that enable the e*Way to perform its intended task. Each Collaboration executes a specified *Collaboration Rule*, which contains the actual instructions to execute the business logic and specifies the applicable *Event Type Definitions* (ETDs). Events Types represent *instances* of their corresponding ETDs.

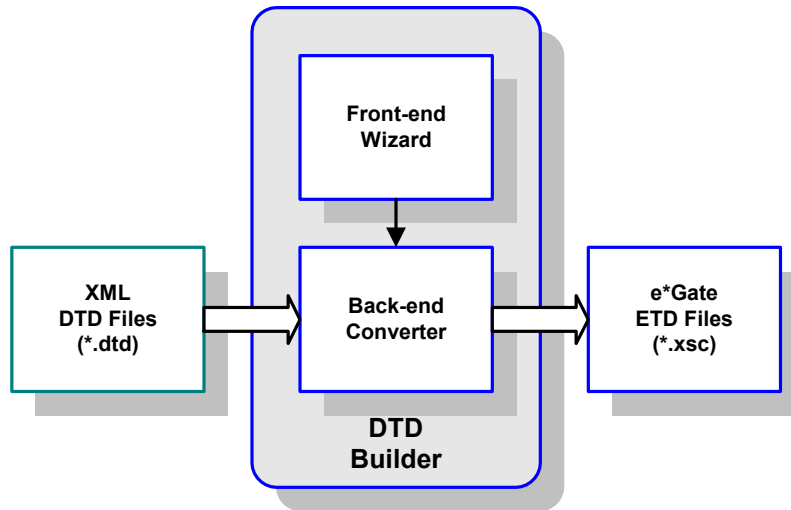
A look inside a typical outbound Collaboration is shown in Figure 102. In this diagram, two e*Way Connections are shown, feeding two external systems. More than two e*Way Connections can be accommodated in each e*Way and, as stated previously, multiple Collaborations as well.

Figure 102 Outbound Collaboration



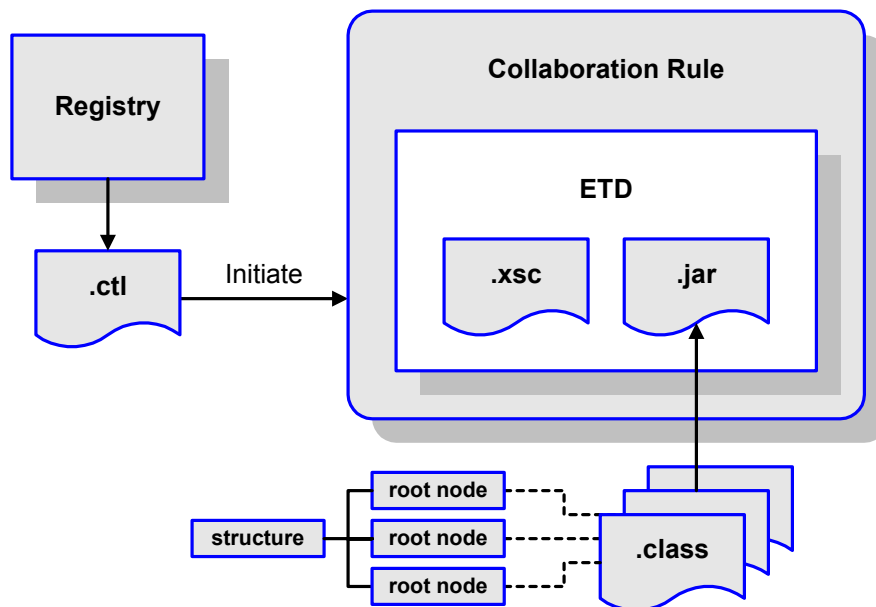
ETDs are representations of the data structure required by specific external systems, and transforming data from one format to another is a major part of the processing performed by the e*Way. Building an ETD obviously requires knowledge of the internal data structure of the specific application. This information often can be obtained by extracting metadata from the external application, which can be automated by using an *ETD Builder*. In the case of the PeopleSoft HTTP e*Way, these ETDs are built automatically by means of the Wizard-based XML DTD Builder (see Figure 103).

Figure 103 DTD Builder



Once compiled, an ETD has two components, an `.xsc` file and a `.jar` file, both having the same file name. The `.jar` file contains `.class` files whose names correspond to the root node names in the ETD. Ultimately, the ETD is used within a Collaboration Rule to define the structure of the corresponding Event. At run time, the Collaboration Rule is initiated according to information contained in a `.ctl` file contained in the e*Gate Registry (see Figure 104).

Figure 104 Event Type Definitions

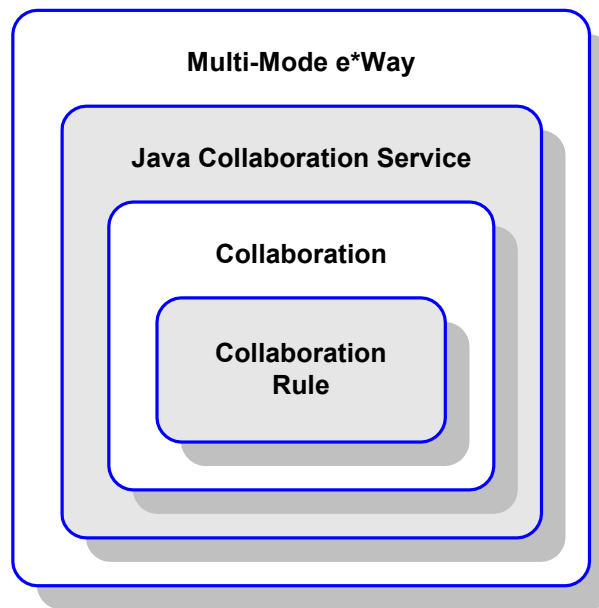


6.4 Java Collaboration Service

The Java Collaboration Service (JCS) provides an environment that allows you to use a Java class to implement the business logic that transforms Events as they move through e*Gate. When data passes through e*Gate using a Java Collaboration, a Java Virtual Machine (JVM) is instantiated and uses the associated Java Collaboration Rules class to accomplish the data transformation.

The relationships between the various Java e*Way components can be depicted as a nested structure, as shown in Figure 105.

Figure 105 Java Component Relationships



The Java Collaboration Service makes it possible to develop Collaboration Rules that execute e*Gate business logic using Java code. Using the Java Collaboration Editor, you create Java classes that utilize the `executeBusinessRules()`, `userInitialize()`, and `userTerminate()` methods.

To use the Java Collaboration Service, you create a Collaboration Rule and select Java as the service. Using Event Type instances of previously defined Event Type Definitions (ETDs), you then use the Java Collaboration Rules Editor to add the rules and logic between the Event Type instances. Compiling the Collaboration Rule creates a Java Collaboration Rules class and all required supporting files. This Java class implements the data transformation logic.

For more information on the Java Collaboration Service, see the *e*Gate Integrator Collaboration Services Reference Guide*.

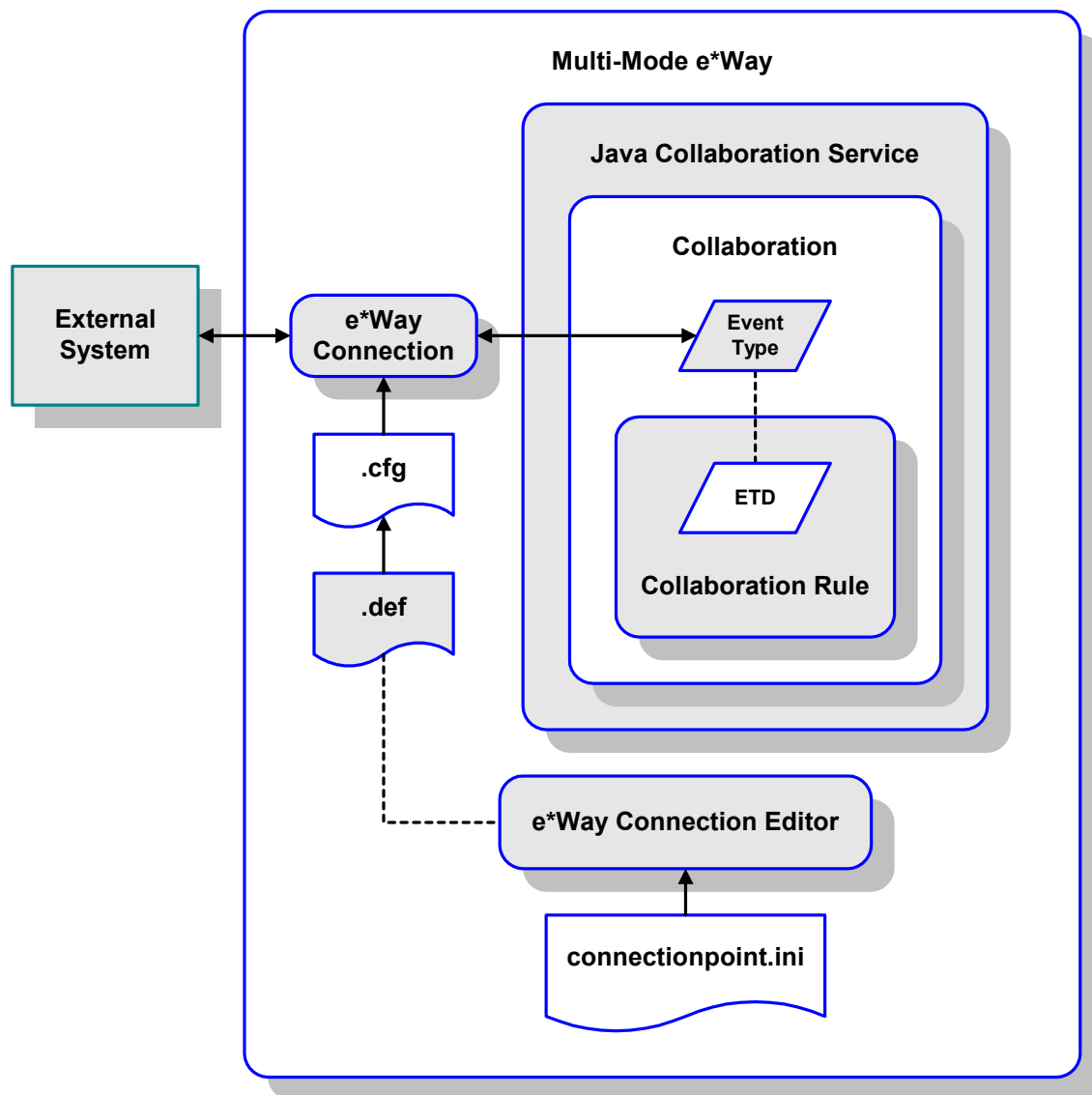
6.5 e*Way Connections

The e*Way Connections provide portals to external systems, allowing a single e*Way to adopt several configuration profiles simultaneously. Individual e*Way Connections can be configured using the e*Way Connection Editor to establish a particular kind of interaction with the external system.

6.5.1 Establishing Connections

An e*Way Connection to an external application is set up as depicted in Figure 106. The .def file supplied with the e*Way is configured for the specific application using the e*Way Connection Editor, and instantiated as a .cfg file for each e*Way Connection.

Figure 106 e*Way Connection Establishment



The e*Way Connection Editor enables you to modify all parameters of a Multi-Mode e*Way that control the way the e*Way communicates with an external application. Because each e*Way functions in a specific way to provide an interface to a specific external application or communications protocol, each e*Way Connection has a unique set of configuration parameters.

For more information on the Java ETD Editor and the Java Collaboration Editor, see the *e*Gate Integrator User's Guide*.

Configuration Parameters

This chapter describes the configuration parameters for the Java PeopleSoft HTTP e*Way.

7.1 Overview

7.1.1 e*Gate to PeopleSoft

Multi-Mode e*Way

The Outbound e*Way's inherent configuration parameters are set using the e*Way Configuration Editor; see [Configuring the Outbound e*Way](#) on page 93 for procedural information. The default configuration is provided in `sapeway.def`. The PeopleSoft HTTP e*Way's configuration parameters are organized into the following sections:

[JVM Settings](#) on page 115

[General Settings](#) on page 120

e*Way Connections

The Outbound e*Way Connection's configuration parameters also are set using the e*Way Configuration Editor; see [Creating e*Way Connections](#) on page 98 for procedural information. The default configuration is provided in `psoft8appmsg.def`. The PeopleSoft HTTP e*Way's configuration parameters are organized into the following sections:

[Connector](#) on page 121

[HTTP](#) on page 122

[Proxies](#) on page 124

[HTTP Authentication](#) on page 126

[SSL](#) on page 127

[ApplicationMessaging](#) on page 132

7.1.2 PeopleSoft to e*Gate

e*Gate API Kit

The Inbound e*Way's inherent configuration parameters are set using the e*Way Configuration Editor; see [Configuring the Inbound e*Way](#) on page 92 for procedural information. The default configuration is provided in `stcewipmp.def`. The Java PeopleSoft HTTP e*Way's configuration parameters are organized into the following sections:

[General Settings](#) on page 137

7.2 Multi-Mode e*Way

7.2.1 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 SDK* is located on the Participating Host.

Required Values

A valid pathname.

Note: This parameter is *required*, and must *not* be left blank.

Additional Information

The JNI DLL name varies for different operating systems:

Operating System	Java 2 JNI DLL Name
NT 4.0/ Windows 2000	jvm.dll
Solaris 2.6, 2.7, 2.8	libjvm.so
Linux 6	libjvm.so
HP-UX	libjvm.sl
AIX 4.3	libjvm.a

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of % symbols. For example:

```
%MY_JNIDLL%
```

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

To ensure that the JNI .dll file loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java SDK (or JDK) installation directory that contain shared libraries (UNIX) or .dll files (Windows).

CLASSPATH Prepend

Description

Specifies the paths to be prefixed to the CLASSPATH environment variable for the Java VM.

Required Values

An absolute path or an environmental variable.

Note: This parameter is optional and may be left blank.

Additional Information

If left un-set, no paths will be prefixed 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 Java VM. This parameter is optional. If left un-set, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set.

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

Required Values

An absolute path or an environment variable.

Note: This parameter is optional and may be left blank.

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 to attach the environment variable to the end of CLASSPATH.

Required Values

YES or NO. The default value is NO.

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 Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Note: This parameter is optional and may be left blank.

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 Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Note: This parameter is optional and may be left blank.

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 will be used.

Required Values

An integer between 0 and 2147483647.

Note: This parameter is optional and may be left blank.

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 Java VM will be used.

Required Values

An integer between 0 and 2147483647.

Note: This parameter is optional and may be left blank.

Class Garbage Collection

Description

Specifies whether the Class Garbage Collection will be done automatically by the Java VM. The selection affects performance issues.

Required Values

YES or NO.

Garbage Collection Activity Reporting

Description

Specifies whether garbage collection activity will be reported for debugging purposes.

Required Values

YES or NO.

Asynchronous Garbage Collection

Description

Specifies whether asynchronous garbage collection activity will be reported for debugging purposes.

Required Values

YES or NO.

Report JVM Info and all Class Loads

Description

Specifies whether the JVM information and all class loads will be reported for debugging purposes.

Required Values

YES or NO.

Disable JIT

Description

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

Required Values

YES or NO.

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

Remote debugging port number

Description

Specifies whether to allow remote debugging of the JVM.

Required Values

YES or NO.

Suspend option for debugging

Description

Specifies whether to suspend option for debugging on JVM startup.

Required Values

YES or NO.

7.2.2 General Settings

Rollback Wait Interval

Description

Specifies the time interval to wait before rolling back the transaction.

Required Values

A number within the range of 0 to 99999999, representing the time interval in milliseconds.

7.3 e*Way Connection

7.3.1 Connector

The parameters in this section furnish the basic e*Way Connection variables.

type

Description

Specifies the type of connection.

Required Values

The value always defaults to **PeopleSoft8 Application Messaging** for a PeopleSoft8 Application Messaging connection.

class

Description

Specifies the class name of the PeopleSoft8 Application Messaging connector object.

Required Values

The default value is `com.stc.eways.psoft8appmsg.PeopleSoft8AppMsgConnector`.

Property.Tag

Description

Specifies the data source.

Required Values

A valid data source package name.

7.3.2 HTTP

The parameters in this section furnish the required HTTP variables.

DefaultURL

Description

Specifies the destination URL (the PeopleSoft 8 Application Messaging Gateway). If HTTPS protocol is to be used, SSL also must be configured (see [SSL](#) on page 127).

Required Values

A full URL, for example:

For PeopleSoft 8.1 with BEA WebLogic,

```
http://gatewayserver/servlets/gateway
```

For PeopleSoft 8.4 with BEA WebLogic,

```
http://gatewayserver/PSIGW/PS81ListeningConnector
```

Where *gatewayserver* is the designation (computer name and port, host name, or IP address) of the Web server hosting the gateway.

AllowCookies

Description

Specifies whether or not cookies sent from servers is stored and sent on subsequent requests. If cookies are not allowed, sessions are not supported.

Required Values

Yes or No; the default is Yes.

ContentType

Description

Specifies the parameters for the Content Type request header.

Required Values

A string. The default value is `text/xml; charset=iso_8859-1`.

For publishing to PeopleSoft, the value is `xml,latin-1`.

AcceptType

Description

Specifies the parameters for the Accept Type request header.

Required Values

A string. The default value is `text/xml`.

7.3.3 Proxies

The parameters in this section furnish the required HTTP Proxy variables.

UseProxy

Description

Specifies whether or not a proxy is used, and whether it is HTTP or HTTPS.

Required Values

One of the following values; the default is **NO**.

- HTTP
- HTTPS
- NO

If **HTTP** is selected, then an HTTP proxy for non-secured connection is used and the HTTP-related parameters in this section applies.

If **HTTPS** is selected, then an HTTPS proxy for secured connection is used and the HTTPS-related parameters in this section applies, as do the parameters in the SSL section (see [SSL](#) on page 127).

If **NO** is selected, then no proxy is used.

HttpProxyHost

Description

Specifies the proxy host for non-secured HTTP connections.

Required Values

A valid host name. There is no default value.

HttpProxyPort

Description

Specifies the proxy port for non-secured HTTP connections.

Required Values

A valid port name. There is no default value.

HttpsProxyHost

Description

Specifies the proxy host for secured HTTPS connections.

Required Values

A valid host name. There is no default value.

HttpsProxyPort

Description

Specifies the proxy port for secured HTTPS connections.

Required Values

A valid port name. There is no default value.

UserName

Description

Specifies the user name for Proxy Authentication.

Required Values

A valid user name. There is no default value.

Password

Description

Specifies the user password for Proxy Authentication.

Required Values

A valid user password. There is no default value.

7.3.4 HTTP Authentication

The parameters in this section furnish the required HTTP Authentication variables.

UseHttpAuthentication

Description

Specifies whether or not standard HTTP Authentication is to be used (if required by the Web site).

Required Values

Yes or No; the default is No.

Additional Information

If this parameter is set to Yes, the parameters **UserName** and **PassWord** *must* be specified.

UserName

Description

Specifies the user name for standard HTTP Authentication.

Required Values

A valid user name. There is no default value.

PassWord

Description

Specifies the user password for standard HTTP Authentication.

Required Values

A valid user password. There is no default value.

7.3.5 SSL

The parameters in this section furnish the required Secure Sockets Layer (SSL) variables.

UseSSL

Description

Specifies whether or not to use SSL.

Required Values

Yes or No; the default is No.

If this parameter is set to **Yes**, the e*Way uses the parameter values in this section to configure to use the **HTTPS** protocol.

If this parameter is set to **No**, the e*Way ignores any certificate information.

Additional Information

If this parameter is set to **Yes**, the parameters **HttpsProtocolImpl** and **Provider** *must* be specified.

HttpsProtocolImpl

Description

Specifies the package that contains the **HTTPS** protocol implementation

Required Values

The default value is **com.sun.net.ssl.internal.www.protocol**.

Note: This parameter *must* be specified if **UseSSL** is set to **Yes**.

Additional Information

Specifying this parameter adds the **HTTPS URLStreamHandler** implementation by including the handler's implementation package name to the list of packages that are searched by the Java URL class. The default value specified is the package containing the Sun reference implementation of the **HTTPS URLStreamHandler**.

Provider

Description

Specifies the Cryptographic Service Provider.

Required Values

The default value is **com.sun.net.ssl.internal.ssl.Provider**.

Note: This parameter *must* be specified if **UseSSL** is set to **Yes**.

Additional Information

Specifying this parameter adds a JSSE provider implementation to the list of provider implementations. The default value specified is the Sun reference implementation of the Cryptographic Service Provider, **SunJSSE**.

X509CertificateImpl

Description

Specifies the implementation class of **X509Certificate**.

Required Values

This field should contain the concatenated values that represent the implementation class and package. For example, if the implementation class is called

```
MyX509CertificateImpl
```

and it appears in the package

```
com.radcrypto
```

then you should specify.

```
com.radcrypto.MyX509CertificateImpl.
```

Note: Specification of this parameter is optional.

SSLSocketFactoryImpl

Description

Specifies the implementation class of **SSL Socket Factory**.

Required Values

This field should contain the concatenated values that represent the implementation class and package. For example, if the implementation class is called

```
MySSLSocketFactoryImpl
```

and it appears in the package

```
com.radcrypto
```

then you should specify.

```
com.radcrypto.MySSLSocketFactoryImpl.
```

Note: Specification of this parameter is optional.

SSLServerSocketFactoryImpl

Description

Specifies the implementation class of **SSL Server Socket Factory**.

Required Values

This field should contain the concatenated values that represent the implementation class and package. For example, if the implementation class is called

```
MySSLServerSocketFactoryImpl
```

and it appears in the package

```
com.radcrypto
```

then you should specify.

```
com.radcrypto.MySSLServerSocketFactoryImpl.
```

Note: Specification of this parameter is optional.

KeyStore

Description

Specifies the default key store file for use by the Key Manager. If this parameter is not specified, then the key store managed by Key Manager is empty.

Required Values

A valid key store file name. There is no default value.

Note: Specification of this parameter is optional.

KeyStoreType

Description

Specifies the default key store type. If this parameter is not specified, then the system sets the default key store type to **jks**.

Required Values

A valid key store type. There is no default value.

Note: Specification of this parameter is optional.

KeyStorePassword

Description

Specifies the default key store password. If this parameter is not specified, then the default key store password is assumed to be a null string.

Required Values

A valid key store password. There is no default value.

Note: Specification of this parameter is optional.

TrustStore

Description

Specifies the default trust store name.

Required Values

A valid trust store name. There is no default value.

Note: Specification of this parameter is optional.

Additional Information

If this parameter is not specified, then the system searches for a default trust store. If a trust store named `<java-home>/lib/security/jssecacerts` is found, it is used. If not, then the system searches for a trust store named `<java-home>/lib/security/cacerts`. If it is found, it is used. If neither is found, then the trust store managed by the Trust Manager is a new, empty trust store.

TrustStoreType

Description

Specifies the default trust store type.

Required Values

A valid trust store type. There is no default value.

Note: Specification of this parameter is optional.

TrustStorePassword

Description

Specifies the default trust store password. If this parameter is not specified, then the default trust store password is assumed to be a null string.

Required Values

A valid trust store name. There is no default value.

Note: Specification of this parameter is optional.

KeyManagerAlgorithm

Description

Specifies the default Key Manager Algorithm name.

Required Values

The name of the key manager algorithm to use. For example, the default key manager algorithm used in the Sun reference implementation of JSSE is **SunX509**.

Note: Specification of this parameter is optional.

TrustManagerAlgorithm

Description

Specifies the default Trust Manager Algorithm name.

Required Values

The name of the trust manager algorithm to use. For example, the default trust manager algorithm used in the Sun reference implementation of JSSE is **SunX509**.

Note: Specification of this parameter is optional.

7.3.6 ApplicationMessaging

The parameters in this section help you set up the required information for Publishing XML messages to PeopleSoft's Application Messaging Gateway using the PeopleSoft Application Messaging protocol.

RequestVersion

Description

Specifies the version of PeopleTools for which the XML messages are valid.

Required Values

Installed version of PeopleTools; the default value is **8.13**.

ToNode

Description

Specifies the name of the local node on the receiving PeopleSoft system (the intended receiving node for messages sent to PeopleSoft 8).

Required Values

A valid node name; the default value is **PSFT_EP**.

Additional Information

This parameter is also referred to as the *node definition*, and corresponds to an entry in the node lookup table on the gateway servlet.

FromNode

Description

Specifies the node from which messages are sent to PeopleSoft 8. The node name must match the node definition for the third party system, as defined in the receiving PeopleSoft system

Required Values

The node name for the sending system; the default is **SeeBeyond**.

Additional Information

The parameters **FromNode**, **Channel**, and **PublicationID** uniquely identify the publication.

Password

Description

Specifies the user password for Application Messaging.

Required Values

A valid user password. There is no default value.

Channel

Description

Specifies the name of the message channel containing the message.

Required Value

A valid message channel name; for example: `ITEM_MAINTENANCE`. There is no default value.

Additional Information

The parameters **FromNode**, **Channel**, and **PublicationID** uniquely identify the publication.

Subject

Description

Specifies the name of the message, as defined in the PeopleSoft system.

Required Values

A valid message name; for example: `ITEM_SYNC`. There is no default value.

MessageVersion

Description

Specifies the message version. In publications containing multiple data versions, there is multiple occurrences of the data.

Required Values

A valid message version name; the default value is `VERSION_1`, which must be replaced with the actual value.

SubjectDetail

Description

Specifies a subtype of a message name, when this information is required by the receiving application.

Required Values

A valid message-name subtype.

Note: This parameter is not generally required.

PublicationID

Description

Specifies the identifier for the publication.

Required Values

A valid ID string. There is no default value.

Note: *Specification of this parameter is optional, since it can be system-generated (see below).*

Additional Information

The parameters **FromNode**, **Channel**, and **PublicationID** uniquely identify the publication. If the **FromNode** value is specified, but this parameter is not, the publication ID is set automatically to the next available publication ID on the specified channel within the subscribing PeopleSoft 8 database.

SubChannel

Description

Specifies the name of the subchannel containing the message.

Required Values

This field should contain the concatenated values that represent the subchannel. For example, if the subchannel is:

Business Unit\Journal ID

then the value of this field is:

M04123456789

where Business Unit = **M04** and Journal ID = **123456789**.

Note: *This parameter is conditional—see below.*

Additional Information

This parameter should be specified if a subchannel is defined in the subscribing PeopleSoft system; otherwise, it may be omitted.

Messages in the same channel but in different subchannels are assumed to refer to distinct objects—for example, different purchase orders or different employees. They are processed in parallel whenever possible.

OriginatingNode

Description

Specifies the name of the node that originally published the message. If not included in the XML file, the system sets it to the publishing node name.

Required Values

Name of the node from which the message originated. There is no default value.

Note: Specification of this parameter is optional.

Additional Information

The purpose of this parameter is to prevent circular publishing.

Publisher

Description

Specifies the operator ID (or class) that generated the message, if required by the receiving application.

Required Values

An application-defined operator ID or class. There is no default value.

Note: This parameter is conditional, and not required by the e*Way.

PublicationProcess

Description

Specifies the name of the program that generated the message, if required by the receiving application.

Required Values

An application-defined name of the program that generated the message. There is no default value.

Note: This parameter is conditional, and not required by the e*Way.

DefaultDataVersion

Description

Identifies the default message version for the sending system, if required by the receiving application.

Required Values

A string representing the default message version. There is no default value.

Note: This parameter is conditional, and not required by the e*Way.

Base64Deflate

Description

Enables or disables base64-encoded compression of XML messages. This compression is recommended to increase network throughput.

Required Values

Yes or No. The default is Yes.

Yes enables base64-encoded compression the <data> portion of the XML message sent from PeopleSoft 8. No disables compression, and the message is published as-is.

7.4 e*Gate API Kit (Multiplexer)

7.4.1 General Settings

The parameters in this section specify the name of the external client system and the IP port through which e*Gate and the client system communicates.

Request Reply IP Port

Description

Specifies the IP port that the e*Way will listen (bind) for client connections. This parameter is used for Request/Reply behavior.

Required Values

A valid TCP/IP port number between 1 and 65536. The default is 26051. Normally, you only need to change the default number if the specified TCP/IP port is in use, or you have other requirements for a specific port number.

Push IP Port

Description

Specifies the IP port through which this e*Way allows an external system to connect and receive unsolicited (without submitting a request) Events.

Required Values

A valid TCP/IP port number between 0 and 65536. The default is 0.

Additional Information

Any Event that this e*Way receives that has zero values for all fields in the 24 byte MUX header is sent to all callers of the **WaitForUnsolicited**. This parameter is optional. If set to zero, the e*Way will follow the Request/Reply scenario and not accept unsolicited Events.

Rollback if no Clients on Push Port

Description

Specifies whether the Event will continually roll back if there are no push clients connected.

Required Values

Yes or No. If set to Yes, the Event will continually roll back if there are no push clients connected.

Wait For IQ Ack

Description

Specifies whether the send client function does *not* return until the Event is committed to the IQ.

Required Values

Yes or No. If set to Yes, the send client function does *not* return until the Event is committed to the IQ.

Note: *This parameter should be set if the data must be committed to the IQ on every transaction before the API returns to the client. Setting this parameter to Yes will significantly impact performance. If normal request/reply type transactions are being sent/received, and the data can be recreated at the client, this parameter should not be set.*

Send Empty MSG When External Disconnect

Description

Specifies whether the e*Way sends an empty incoming message (containing only the multiplexer header) when an external client disconnects.

Required Values

Yes or No. If set to Yes, the e*Way sends an empty incoming message when an external client disconnects.

MUX Instance ID

Description

Specifies whether the specified 8 (eight) bytes is prefixed to the 24 (twenty-four) byte session ID of the request received from the external connection before sending to e*Gate.

Required Values

A string; the default is 0. If this value is other than "0", the 8 bytes are prefixed to the 24 byte session ID.

Note: *This is a string where "00" and "00000000" are valid MUX Instance IDs, while "0" is to turn this option off. Only the first 8 bytes are used.*

MUX Recovery ID

Description

Specifies whether the 8 bytes are prefixed to the reply and republish back to e*Gate provided the value is other than "0" and the multiplexer finds that the session related to the MUX ID in the return message has been dropped.

Required Values

A string; the default is 0. If this value is other than "0", the 8 bytes are prefixed to the 24 byte session ID.

Note: This is a string where "00" and "00000000" are valid MUX Recovery IDs, while "0" is to turn this option off. Only the first 8 bytes are used.

Java Classes and Methods

The PeopleSoft HTTP e*Way contains Java methods that are used to extend the functionality of the basic e*Way core.

8.1 Overview

This chapter contains descriptions of methods that are exposed in the user interface. Additional methods contained in the e*Way should only be accessed or modified by qualified SeeBeyond personnel. Unless otherwise noted, all classes and methods described in this chapter are **public**. Methods inherited from classes other than those described in this chapter are listed, but not described.

For ease of use, this chapter is organized into the following sections:

[Object Classes](#) on page 141

[Constructors](#) on page 149

[Methods](#) on page 154

8.2 Object Classes

The PeopleSoft HTTP e*Way makes use of the following object classes:

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

[PeopleSoft8AppMsgConnector Class](#) on page 144

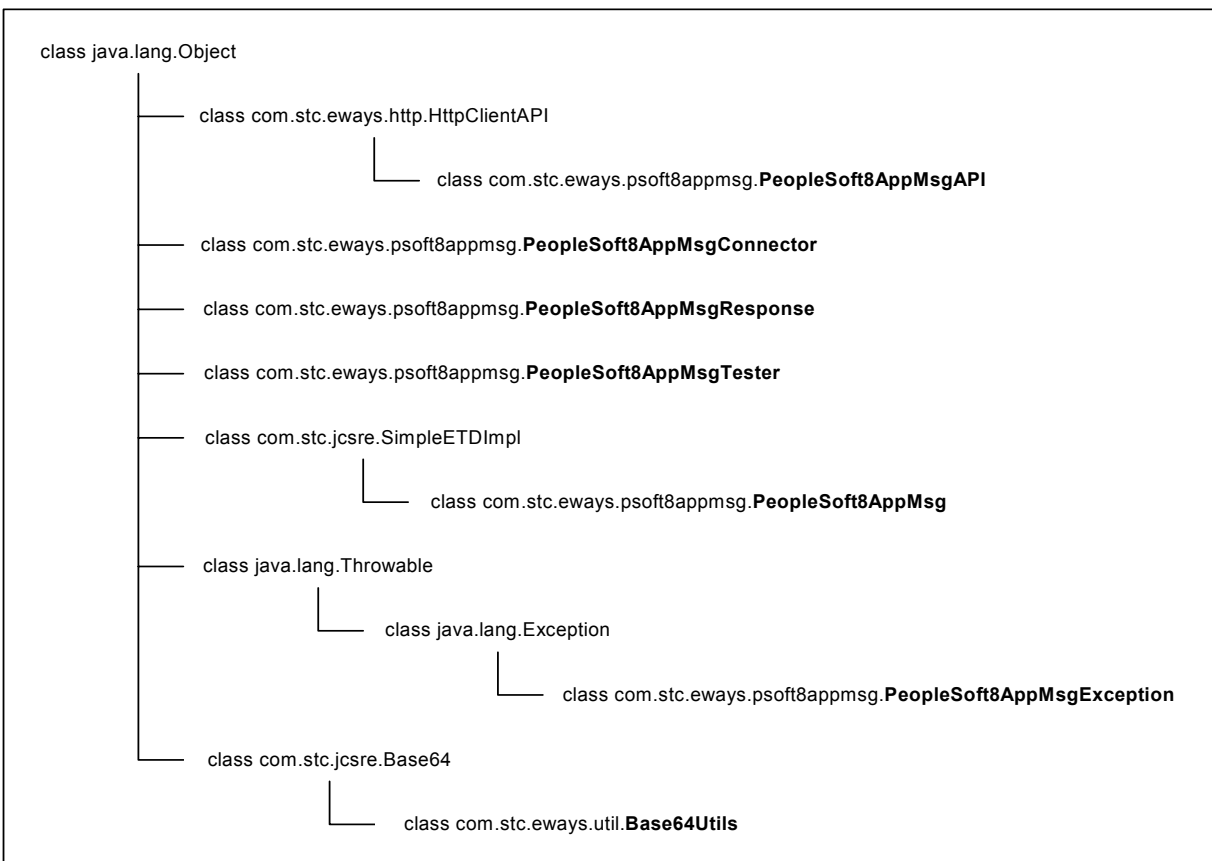
[PeopleSoft8AppMsgException](#) on page 151

[PeopleSoft8AppMsgResponse Class](#) on page 146

[PeopleSoft8AppMsgTester Class](#) on page 147

[Base64Utils Class](#) on page 148

Figure 107 Class Hierarchy



8.2.1 PeopleSoft8AppMsg Class

Description

This class extends the SeeBeyond **SimpleETDImpl** class.

Definition

`PeopleSoft8AppMsg`

Constructors

[PeopleSoft8AppMsg](#) on page 149

Methods

[initialize](#) on page 154

[reset](#) on page 157

[setToNode](#) on page 161

[getNode](#) on page 161

[setFromNode](#) on page 162

[getNode](#) on page 163

[setOriginatingNode](#) on page 163

[getNode](#) on page 164

[setPassWord](#) on page 164

[getPassWord](#) on page 165

[setChannel](#) on page 166

[getChannel](#) on page 166

[setSubChannel](#) on page 167

[getSubChannel](#) on page 167

[setSubject](#) on page 168

[getSubject](#) on page 169

[setSubjectDetail](#) on page 169

[getSubjectDetail](#) on page 170

[sendMessage](#) on page 177

[setPeopleSoft8AppMsgResponse](#) on page 177

[getPeopleSoft8AppMsgResponse](#) on page 178

[setDefaultDataVersion](#) on page 158

[getDefaultDataVersion](#) on page 158

[setMessageVersion](#) on page 159

[getMessageVersion](#) on page 159

[setRequestVersion](#) on page 160

[getRequestVersion](#) on page 160

[setPublicationID](#) on page 170

[getPublicationID](#) on page 171

[setPublicationProcess](#) on page 171

[getPublicationProcess](#) on page 172

[setPublisher](#) on page 174

[getPublisher](#) on page 174

[setXMLMessage](#) on page 176

[getXMLMessage](#) on page 176

[setBase64Deflate](#) on page 180

[getBase64Deflate](#) on page 180

Methods Inherited from `java.lang.Object` Class

`equals`

`notify`

`wait`

`getClass`

`notifyAll`

`wait`

`hashCode`

`toString`

`wait`

8.2.2 PeopleSoft8AppMsgAPI Class

Description

This class extends the SeeBeyond **HttpClientAPI** class. Please refer to the *HTTP e*Way Intelligent Adapter User's Guide* for information on the **HttpClientAPI** class.

Definition

PeopleSoft8AppMsgAPI

Constructors

[PeopleSoft8AppMsgAPI](#) on page 149

[PeopleSoft8AppMsgAPI](#) on page 149

Methods

[reset](#) on page 157

[setToNode](#) on page 161

[getToNode](#) on page 161

[setFromNode](#) on page 162

[getFromNode](#) on page 163

[setOriginatingNode](#) on page 163

[getOriginatingNode](#) on page 164

[setPassWord](#) on page 164

[getPassWord](#) on page 165

[setChannel](#) on page 166

[getChannel](#) on page 166

[setSubChannel](#) on page 167

[getSubChannel](#) on page 167

[setSubject](#) on page 168

[getSubject](#) on page 169

[setSubjectDetail](#) on page 169

[getSubjectDetail](#) on page 170

[setDefaultDataVersion](#) on page 158

[getDefaultDataVersion](#) on page 158

[setMessageVersion](#) on page 159

[getMessageVersion](#) on page 159

[setRequestVersion](#) on page 160

[getRequestVersion](#) on page 160

[setPublicationID](#) on page 170

[getPublicationID](#) on page 171

[setPublicationProcess](#) on page 171

[getPublicationProcess](#) on page 172

[setPublisher](#) on page 174

[getPublisher](#) on page 174

[setXMLMessage](#) on page 176

[getXMLMessage](#) on page 176

[setBase64Deflate](#) on page 180

[getBase64Deflate](#) on page 180

Methods Inherited from java.lang.Object Class

equals	notify	wait
getClass	notifyAll	wait
hashCode	toString	wait

8.2.3 PeopleSoft8AppMsgConnector Class

Description

This class extends the `java.lang.Object` class and implements the `com.stc.jcsre.EBobConnector` class.

Definition

```
PeopleSoft8AppMsgConnector
```

Constructors

[PeopleSoft8AppMsgConnector](#) on page 150

Methods

[open](#) on page 154

[close](#) on page 156

[isOpen](#) on page 155

[getProperties](#) on page 156

Methods Inherited from `java.lang.Object` Class

<code>equals</code>	<code>notify</code>	<code>wait</code>
<code>getClass</code>	<code>notifyAll</code>	<code>wait</code>
<code>hashCode</code>	<code>toString</code>	<code>wait</code>

8.2.4 PeopleSoft8AppMsgException Class

Description

This class extends the `java.lang.Exception` class.

Definition

```
PeopleSoft8AppMsgException
```

Constructors

[PeopleSoft8AppMsgException](#) on page 150

[PeopleSoft8AppMsgException](#) on page 151

[PeopleSoft8AppMsgException](#) on page 151

Methods

Methods Inherited from `java.lang.Object` Class

<code>equals</code>	<code>notify</code>	<code>wait</code>
<code>getClass</code>	<code>notifyAll</code>	<code>wait</code>
<code>hashCode</code>		<code>wait</code>

Methods Inherited from `java.lang.Throwable` Class

<code>fillInStackTrace</code>	<code>getMessage</code>
<code>printStackTrace</code>	<code>getLocalizedMessage</code>
<code>printStackTrace</code>	<code>toString</code>
<code>printStackTrace</code>	

8.2.5 PeopleSoft8AppMsgResponse Class

Description

This class extends the `java.lang.Object` class.

Definition

```
PeopleSoft8AppMsgResponse
```

Constructors

[PeopleSoft8AppMsgResponse](#) on page 151

Methods

[setChannel](#) on page 166

[getChannel](#) on page 166

[setPublicationID](#) on page 170

[getPublicationID](#) on page 171

[setPublishingNode](#) on page 175

[getPublishingNode](#) on page 175

[setPublishTimeStamp](#) on page 173

[getPublishTimeStamp](#) on page 173

[setResponseCode](#) on page 179

[getResponseCode](#) on page 179

[setExceptionDefaultMsg](#) on page 182

[getExceptionDefaultMsg](#) on page 183

[setExceptionMsgID](#) on page 183

[getExceptionMsgID](#) on page 184

[setExceptionMsgSet](#) on page 184

[getExceptionMsgSet](#) on page 185

[setExceptionTitle](#) on page 185

[getExceptionTitle](#) on page 186

[setIsExceptionResponse](#) on page 186

[getIsExceptionResponse](#) on page 187

Methods Inherited from `java.lang.Object` Class

<code>equals</code>	<code>notify</code>	<code>wait</code>
<code>getClass</code>	<code>notifyAll</code>	<code>wait</code>
<code>hashCode</code>	<code>toString</code>	<code>wait</code>

8.2.6 PeopleSoft8AppMsgTester Class

Description

This class extends the `java.lang.Object` class.

Definition

```
PeopleSoft8AppMsgTester
```

Constructors

[PeopleSoft8AppMsgTester](#) on page 152

Methods

[main](#) on page 157

Methods Inherited from `java.lang.Object` Class

<code>equals</code>	<code>notify</code>	<code>wait</code>
<code>getClass</code>	<code>notifyAll</code>	<code>wait</code>
<code>hashCode</code>	<code>toString</code>	<code>wait</code>

8.2.7 Base64Utils Class

Description

This class extends the **com.stc.jcsre.Base64** class.

Definition

```
Base64Utils
```

Constructors

[Base64Utils](#) on page 152

[Base64Utils](#) on page 153

Methods

[byteToBase64String](#) on page 181

[base64DecodeToByte](#) on page 181

Methods Inherited from com.stc.jcsre.Base64 Class

ENCODING

EOF

LINELIN

8.3 Constructors

PeopleSoft8AppMsg

Description

This constructor constructs a new **PeopleSoft8AppMsg**.

Signature

```
PeopleSoft8AppMsg ()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsg Class](#) on page 142

PeopleSoft8AppMsgAPI

Description

This constructor constructs a new **PeopleSoft8AppMsgAPI**.

Signature

```
PeopleSoft8AppMsgAPI ()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsgAPI Class](#) on page 143

PeopleSoft8AppMsgAPI

Description

This constructor constructs a new **PeopleSoft8AppMsgAPI**. It throws an exception when an exception is returned to it following an attempt to construct a **java.net.URL** object with the supplied URL string.

Signature

```
PeopleSoft8AppMsgAPI (psoft8GWServletURL)
```

Parameters

Name	Type	Description
<code>psoft8GWServletURL</code>	<code>urlString</code>	URL of the PeopleSoft 8 Gateway Servlet.

Throws

`java.net.MalformedURLException`

Used by

[PeopleSoft8AppMsgAPI Class](#) on page 143

PeopleSoft8AppMsgConnector

Description

This constructor constructs a new **HttpClientConnector**.

Signature

```
PeopleSoft8AppMsgConnector()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsgConnector Class](#) on page 144

PeopleSoft8AppMsgException

Description

This constructor creates a new **PeopleSoft8AppMsgException** without a detail message.

Signature

```
PeopleSoft8AppMsgException()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsgException Class](#) on page 145

PeopleSoft8AppMsgException

Description

This constructor constructs a new **PeopleSoft8AppMsgException** with the specified detail message.

Signature

```
PeopleSoft8AppMsgException(msg)
```

Parameters

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

Throws

None.

Used by

[PeopleSoft8AppMsgException Class](#) on page 145

PeopleSoft8AppMsgException

Description

This constructor constructs a new **PeopleSoft8AppMsgException** with the error code and specified detail message.

Signature

```
PeopleSoft8AppMsgException(errorCode, msg)
```

Parameters

Name	Type	Description
errorCode	integer	Error code.
msg	java.lang.String	Detail message.

Throws

None.

Used by

[PeopleSoft8AppMsgException Class](#) on page 145

PeopleSoft8AppMsgResponse

Description

This constructor constructs a new **PeopleSoft8HttpAPI**.

Signature

```
PeopleSoft8AppMsgResponse()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsgResponse Class](#) on page 146

PeopleSoft8AppMsgTester

Description

This constructor constructs a new **PeopleSoft8AppMsgTester**.

Signature

```
PeopleSoft8AppMsgTester()
```

Parameters

None.

Throws

None.

Used by

[PeopleSoft8AppMsgTester Class](#) on page 147

Base64Utils

Description

This constructor constructs new **Base64Utils**.

Signature

```
Base64Utils()
```

Parameters

None.

Throws

None.

Used by

[Base64Utils Class](#) on page 148

Base64Utils

Description

This constructor constructs new **Base64Utils**.

Signature

```
Base64Utils(fi, fo)
```

Parameters

Name	Type	Description
fi	java.io.InputStream	Input data.
fo	java.io.OutputStream	Output data.

Throws

None.

Used by

[Base64Utils Class](#) on page 148

8.4 Methods

initialize

Description

This method is called by external (collaboration service) to initialize object.

Signature

```
initialize(cntrCollab, key, mode)
```

Parameters

Name	Type	Description
cntrCollab	com.stc.common.collabService.JCollabController	The Java Collaboration Controller object.
key	java.lang.String	Key to one of the JMsgObjects.
mode	integer	Mode for ETD (IN_MODE, OUT_MODE, or IN_OUT_MODE)

Return Type

void

Overrides

initialize in class **com.stc.jcsre.SimpleETDImpl**

Throws

- com.stc.common.collabService.CollabConnException,
- com.stc.common.collabService.CollabDataException

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

open

Description

This method opens the connector for accessing the external system. It throws an exception when connection problems occur.

Signature

```
open(intoEgate)
```

Parameters

Name	Type	Description
intoEgate	boolean	Specify true if connector is to subscribe for events initially from an external and inbound to e*Gate, false if connector is to publish events outbound from e*Gate and to an external system.

Return Type

void

Overrides

None.

Throws

com.stc.jcsre.EBobConnectionException

Contained in

[PeopleSoft8AppMsgConnector Class](#) on page 144

isOpen

Description

This method verifies that the connector to the external system is still available. It returns true if the connector is still open and available, otherwise false. It throws an exception when connection problems occur.

Signature

```
isOpen()
```

Parameters

None.

Return Type

boolean

Overrides

None.

Throws

com.stc.jcsre.EBobConnectionException

Contained in

[PeopleSoft8AppMsgConnector Class](#) on page 144

getProperties

Description

This method retrieves the connection properties (as stored by the constructor) used by the connector to access the external.

Signature

```
getProperties()
```

Parameters

None.

Return Type

java.util.Properties

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgConnector Class](#) on page 144

close

Description

This method closes the connector to the external system and releases resources. It throws an exception when connection problems occur.

Signature

```
close()
```

Parameters

None.

Return Type

void

Overrides

None.

Throws

com.stc.jcsre.EBobConnectionException

Contained in

[PeopleSoft8AppMsgConnector Class](#) on page 144

main

Description

This method provides an entry point for a stand-alone test program.

Signature

```
main(args)
```

Parameters

Name	Type	Description
args	java.lang.String	Required arguments.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgTester Class](#) on page 147

reset

Description

This method clears all headers and request data from memory.

Signature

```
reset()
```

Parameters

None.

Return Type

boolean

Overrides

reset in class **com.stc.jcsre.SimpleETDImpl**

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setDefaultDataVersion

Description

This method sets the default message version for the sending system.

Signature

```
setDefaultDataVersion(defaultDataVersion)
```

Parameters

Name	Type	Description
defaultDataVersion	java.lang.String	The default message version.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getDefaultDataVersion

Description

This method gets the default message version that was previously set with the method **setDefaultDataVersion**, or null if the default message version was not previously set.

Signature

```
getDefaultDataVersion()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

setMessageVersion

Description

This method sets the name of the message version. For publications with multiple data versions, there are multiple occurrences of the data.

Signature

```
setMessageVersion(messageVersion)
```

Parameters

Name	Type	Description
messageVersion	java.lang.String	The message version of the messages sent.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getMessageVersion

Description

This method gets the name of the message version. It returns the message version previously set with the **setMessageVersion** method, or null if the message version was not previously set.

Signature

```
getMessageVersion()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setRequestVersion

Description

This method sets the PeopleTools version for which the XML message is valid.

Signature

```
setRequestVersion(psoft8Version)
```

Parameters

Name	Type	Description
psoft8Version	java.lang.String	The PeopleTools version.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getRequestVersion

Description

This method gets the PeopleTools version that was previously set with **setRequestVersion**, or null if the PeopleTools version was not previously set.

Signature

```
getRequestVersion()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setToNode

Description

This method sets the name of the node for which the messages is intended. This must correspond to an entry in the node lookup table on the gateway servlet, and the name of the local node (node definition) on the receiving PeopleSoft system.

Signature

```
setToNode ( toNode)
```

Parameters

Name	Type	Description
toNode	java.lang.String	The node to which the message is published.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getNode

Description

This method gets the name of the node for which the messages is intended. It returns the node name set with the **setToNode** method.

Signature

`getToNode()`

Parameters

None.

Return Type

`java.lang.String`

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setFromNode

Description

This method sets the name of the node from which the messages originate. The node name must match the node definition for the third party system as defined in the receiving PeopleSoft system.

Signature

`setFromNode(fromNode)`

Parameters

Name	Type	Description
<code>fromNode</code>	<code>java.lang.String</code>	The node from which the messages originate.

Return Type

`void`

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getFromNode

Description

This method gets the name of the node from which messages originate. It returns the source node name that was previously set with **setFromNode**, or null if the node was not previously set.

Signature

```
getFromNode()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setOriginatingNode

Description

This method sets the name of the node that originally published the message, and is used to prevent circular publishing. If not an XML file, the system sets it to the publishing node name.

Signature

```
setOriginatingNode(originatingNode)
```

Parameters

Name	Type	Description
originatingNode	java.lang.String	The name of the originating node.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getOriginatingNode

Description

This method gets the name of the originating node that was previously set with the method **setOriginatingNode**, or null if the originating node was not previously set.

Signature

```
getOriginatingNode()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setPassWord

Description

This method sets the password associated with the source node. This value is stored in the PeopleSoft database and must be communicated to the system administrators for the publishing system. If the node definition on the sending system has a node group defined, the password is present. If the node definition on the receiving system has a node group defined, the password must be present and must match the node group password.

Signature

```
setPassWord(passWord)
```

Parameters

Name	Type	Description
passWord	java.lang.String	The password associated with the source node.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getPassWord

Description

This method gets the password associated with the source node. Returns the password that was previously set with **setPassWord**, or null if the password was not previously set.

Signature

```
getPassWord()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setChannel

Description

This method sets the name of the message channel containing the message.

Signature

```
setChannel(channel)
```

Parameters

Name	Type	Description
channel	java.lang.String	The message channel to publish.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

[PeopleSoft8AppMsgResponse Class](#) on page 146

getChannel

Description

This method gets the name of the message channel containing the message. Returns the channel that was previously set with **setChannel**, or null if the channel was not previously set.

Signature

```
getChannel()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

[PeopleSoft8AppMsgResponse Class](#) on page 146

setSubChannel

Description

This method sets the name of subchannel which contains the message. Messages in the same channel but in different subchannels are assumed to refer to distinct objects—for example, different purchase orders or different employees. They are processed in parallel where possible.

This field should contain the concatenated values that represent the subchannel. For example, if the subchannel is **Business Unit \ Journal ID**, then the value of this field is **M04123456789** (where Business Unit = M04 and Journal ID = 123456789).

This field should be included if the subscribing PeopleSoft system has a subchannel defined. Otherwise, it may be omitted.

Signature

```
setSubChannel (subChannel)
```

Parameters

Name	Type	Description
subChannel	java.lang.String	The name of the sub-channel.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getSubChannel

Description

This method gets the name of subchannel that was previously set with **setSubChannel**, or null if the sub-channel was not previously set.

Signature

`getSubChannel ()`

Parameters

None.

Return Type

`java.lang.String`

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setSubject

Description

This method sets the name of the message as defined in the PeopleSoft system. This is the Message Definition.

Signature

`setSubject (subject)`

Parameters

Name	Type	Description
subject	<code>java.lang.String</code>	The name of the message being sent.

Return Type

`java.lang.String`

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getSubject

Description

This method gets the name of the message that was previously set with the **setSubject** method, or null if the name was not previously set.

Signature

```
getSubject()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setSubjectDetail

Description

This method sets the Application-defined subtype of message name.

Signature

```
setSubjectDetail(subjectDetail)
```

Parameters

Name	Type	Description
subjectDetail	java.lang.String	The message subtype defined.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142
[PeopleSoft8AppMsgAPI Class](#) on page 143

getSubjectDetail

Description

This method gets the name of the message version message subtype that was previously set with **setSubjectDetail**, or null if the message subtype was not previously set.

Signature

```
getSubjectDetail()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142
[PeopleSoft8AppMsgAPI Class](#) on page 143

setPublicationID

Description

This method sets the System-generated identifier for the publication. The fields, publishing node, channel, and publication ID uniquely identify the publication. If the From Node is specified and the Publication ID is omitted, the publication ID is set to the next available publication ID on that channel within the subscribing PeopleSoft database.

Signature

```
setPublicationID(publicationID)
```

Parameters

Name	Type	Description
publicationID	java.lang.String	The publication id for the message to be sent.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

[PeopleSoft8AppMsgResponse Class](#) on page 146

getPublicationID

Description

This method gets the System-generated identifier for the publication that was previously set with **setPublicationID**, or null if the publication ID was not previously set.

Signature

```
getPublicationID()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

[PeopleSoft8AppMsgResponse Class](#) on page 146

setPublicationProcess

Description

This method sets the Application-defined name of the program that generated the message (if required by the application).

Signature

```
setPublicationProcess(publicationProcess)
```

Parameters

Name	Type	Description
publicationProcess	java.lang.String	The operator ID/class that published the message.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getPublicationProcess

Description

This method gets the name of the program that was previously set with the method **setPublicationProcess**, or null if the originating node was not previously set.

Signature

```
getPublicationProcess()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setPublishTimeStamp

Description

This method sets the publication timestamp received from the PeopleSoft 8 Application Messaging response.

Signature

```
setPublishTimeStamp(publishTimeStamp)
```

Parameters

Name	Type	Description
publishTimeStamp	java.lang.String	The publication timestamp received from the PeopleSoft 8 Application Messaging response.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getPublishTimeStamp

Description

This method gets the publication timestamp received from the PeopleSoft 8 Application Messaging response, or null if no timestamp information was received.

Signature

```
getPublishTimeStamp()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setPublisher

Description

This method sets the Application-defined operator ID/class that published the message (if required by the application).

Signature

```
setPublisher(publisher)
```

Parameters

Name	Type	Description
publisher	java.lang.String	The operator ID/class that published the message.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getPublisher

Description

This method gets the operator ID/class that was previously set with the method **setPublisher**, or null if the originating node was not previously set.

Signature

```
getPublisher()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

setPublishingNode

Description

This method sets the publishing node received from the PeopleSoft 8 Application Messaging response.

Signature

```
setPublishingNode()
```

Parameters

Name	Type	Description
publishingNode	java.lang.String	The node from which the XML message was published.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getPublishingNode

Description

This method gets the publishing node received from the PeopleSoft 8 Application Messaging response, or null if no publishing node information was received.

Signature

```
getPublishingNode()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setXMLMessage

Description

This method sets the PeopleSoft 8 XML message to post.

Signature

```
setXMLMessage(psoft8XML)
```

Parameters

Name	Type	Description
psoft8XML	java.lang.String	The PeopleSoft 8 XML message to post.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getXMLMessage

Description

This method gets the PeopleSoft 8 XML message previously set with **setXMLMessage**, or null if the XML message was not previously set.

Signature

```
getXMLMessage ()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

sendMessage

Description

This method POSTs the PeopleSoft 8 XML message that was previously set with **setXMLMessage**.

Signature

```
sendMessage ()
```

Parameters

None.

Return Type

void

Overrides

None.

Throws

PeopleSoft8AppMsgException

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

setPeopleSoft8AppMsgResponse

Description

This method sets the **PeopleSoft8AppMsgResponse** object as the response from PeopleSoft 8 Application Messaging.

Signature

```
setPeopleSoft8AppMsgResponse ( response )
```

Parameters

Name	Type	Description
response	PeopleSoft8AppMsgResponse	The PeopleSoft8AppMsgResponse object.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

getPeopleSoft8AppMsgResponse

Description

This method gets the **PeopleSoft8AppMsgResponse** object holding the PeopleSoft 8 Application Messaging published response.

Signature

```
getPeopleSoft8AppMsgResponse ()
```

Parameters

None.

Return Type

object

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

setResponseCode

Description

This method sets the PeopleSoft 8 Application Messaging response code.

Signature

```
setResponseCode (responseCode)
```

Parameters

Name	Type	Description
responseCode	integer	The PeopleSoft 8 Application Messaging response code.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getResponseCode

Description

If the response from PeopleSoft 8 is not an exception, then this method gets the PeopleSoft 8 Application Messaging response code for the publication.

Signature

```
getResponseCode ()
```

Parameters

None.

Return Type

integer

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setBase64Deflate

Description

This method enables or disables base64-encoded compression (deflation) of XML messages. The default is to enable base64-encoding.

Signature

```
setBase64Deflate(base64Deflate)
```

Parameters

Name	Type	Description
base64Deflate	boolean	Set to true if desired to compress XML messages using base64 encoding; otherwise set to false.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

getBase64Deflate

Description

This method determine whether or not base64 encoding of compressed XML messages is used. This method returns what was set with `setBase64Deflate`. If `setBase64Deflate` was never called, the default is that base64 encoding of compressed messages is used.

Signature

```
getBase64Deflate()
```

Parameters

None.

Return Type

boolean

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsg Class](#) on page 142

[PeopleSoft8AppMsgAPI Class](#) on page 143

byteToBase64String

Description

This method converts byte-format data to base64-encoded data.

Signature

```
byteToBase64String(data)
```

Parameters

Name	Type	Description
data	byte[]	Data in byte format.

Return Type

java.lang.String

Overrides

None.

Throws

java.io.IOException

Contained in

[Base64Utils Class](#) on page 148

base64DecodeToByte

Description

This method converts base64-encoded data to byte-format data.

Signature

```
base64DecodeToByte(data)
```

Parameters

Name	Type	Description
data	java.lang.String	Data in base64 string format.

Return Type

byte[]

Overrides

None.

Throws

java.io.IOException

Contained in

[Base64Utils Class](#) on page 148

setExceptionDefaultMsg

Description

This method sets the default message for the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message.

Signature

```
setExceptionDefaultMsg(msg)
```

Parameters

Name	Type	Description
msg	java.lang.String	The default error message for the exception.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getExceptionDefaultMsg

Description

This method gets the default message for the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message.

Signature

```
getExceptionDefaultMsg()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setExceptionMsgID

Description

This method sets the message ID number of the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message.

Signature

```
setExceptionMsgID(msgID)
```

Parameters

Name	Type	Description
msgID	java.lang.String	The message ID number of the exception.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getExceptionMsgID

Description

This method gets the message ID number for the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message, or null if no message ID number was returned.

Signature

```
getExceptionMsgID()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setExceptionMsgSet

Description

This method sets the message set number for the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message.

Signature

```
setExceptionMsgSet(msgSet)
```

Parameters

Name	Type	Description
msgSet	java.lang.String	The message set number of the exception.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getExceptionMsgSet

Description

This method gets the message set number for the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message, or null if no message set number was returned.

Signature

```
getExceptionMsgSet()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setExceptionTitle

Description

This method sets the title of the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message.

Signature

```
setExceptionTitle(title)
```

Parameters

Name	Type	Description
title	java.lang.String	The title of the exception.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getExceptionTitle

Description

This method gets the title of the exception message as received from the PeopleSoft 8 Application Messaging exception XML response message, or null if no exception title information was returned.

Signature

```
getExceptionTitle()
```

Parameters

None.

Return Type

java.lang.String

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

setIsExceptionResponse

Description

This method sets the exception flag to indicate a PeopleSoft 8 Application Messaging exception response was received.

Signature

```
setIsExceptionResponse(isExceptionResponse)
```

Parameters

Name	Type	Description
isExceptionResponse	boolean	The flag to indicate an exception message was received. Set to true if an exception message was received; otherwise, set to false.

Return Type

void

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

getIsExceptionResponse

Description

This method gets the exception flag to see if a PeopleSoft 8 Application Messaging exception response was received. Returns true if an exception message was received or false if the message received was not an exception.

Signature

```
getIsExceptionResponse()
```

Parameters

None.

Return Type

boolean

Overrides

None.

Throws

None.

Contained in

[PeopleSoft8AppMsgResponse Class](#) on page 146

MUX Subscription Handler

A.1 Object Classes

The MUX subscription handler consists of ten java classes, which are contained in the `stcph.jar` file. This file is available when the e*Gate participating host is installed.

Note: The `MuxPublicationHandler` class depends on the MUX Java Client classes, which also are contained in the `stcph.jar` file.

A.1.1 Entry Class

Description

Holds the following information for an instance of a MUX subscription handler:

- Node Name
- MUX Host
- MUX Port
- MUX Expire
- MUX Timeout
- Uncompress?
- Base64-Decode?
- Include Headers?
- Log File

A.1.2 MuxHandlerConstants Class

Description

Contains constant values for the MUX handler package such as the name of the configuration file to store the parameters persistently.

A.1.3 MuxHandlerEntry Class

Description

Maintains a collection of Entry classes. This class Loads and parses the configuration file in order to load the data into memory. It also saves changes to the values for the Entry classes to the configuration file.

A.1.4 AdministerMuxHandler Class

Description

Displays the MUX administration root page for administering the MUX subscription handler.

A.1.5 AdministerMuxHandlerAddMode Class

Description

Displays the MUX administration page for adding a MUX subscription handler.

A.1.6 AdministerMuxHandlerDeleteMode Class

Description

Displays the MUX administration page for deleting a MUX subscription handler.

A.1.7 AdministerMuxHandlerEditMode Class

Description

Displays the MUX administration Page for editing the configuration values of a MUX subscription handler.

A.1.8 AdministerMuxHandlerError Class

Description

Displays the error page when an error occurs while administering the MUX subscription handler.

A.1.9 MuxPublicationHandler Class

Description

Handles the publishing of XML messages from PeopleSoft to the MUX e*Way using the MUX Java Client APIs.

A.1.10 MuxHandler Class

Description

Loaded by the Application Messaging Gateway and serves as an entry point to the MUX subscription handler. It loads any pre-existing MUX subscription handlers.

Index

A

AcceptType parameter 122
 AdministerMuxHandler class 189
 AdministerMuxHandlerAddMode class 189
 AdministerMuxHandlerDeleteMode class 189
 AdministerMuxHandlerEditMode class 189
 AdministerMuxHandlerError class 189
 AIX 15
 AllowCookies parameter 122
 Asynchronous Garbage Collection parameter 118
 Autorun 16

B

base64DecodeToByte method 181
 Base64Deflate parameter 136
 Base64Utils Class 148
 Base64Utils constructor 152, 153
 byteToBase64String method 181

C

Changing the User Name 94
 Channel parameter 133
 Class Garbage Collection parameter 118
 class parameter 121
 CLASSPATH Append From Environment Variable parameter 116
 CLASSPATH Override parameter 116
 CLASSPATH Prepend parameter 115
 close method 156
 Collaboration 77, 104, 106, 108
 Rules 104, 105
 Service 104
 configuration
 ApplicationMessaging 132–136
 Connector 121
 General Settings (Multiplexer) 137–139
 General Settings (Mutli-Mode) 120
 HTTP 122–123
 HTTP Authentication 126
 JVM Settings 115–119
 Proxies 124–125
 SSL 127–131

Configuration parameters
 Push IP Port 137
 Request Reply IP Port 137
 configuration parameters
 AcceptType 122
 AllowCookies 122
 Asynchronous Garbage Collection 118
 Base64Deflate 136
 Channel 133
 class 121
 Class Garbage Collection 118
 CLASSPATH Append From Environment Variable 116
 CLASSPATH Override 116
 CLASSPATH Prepend 115
 ContentType 122
 DefaultDataVersion 135
 DefaultURL 122
 Disable JIT 118
 FromNode 132
 Garbage Collection Activity Reporting 118
 HttpProxyHost 124
 HttpProxyPort 124
 HttpsProtocolImpl 127
 HttpsProxyHost 124
 HttpsProxyPort 125
 Initial Heap Size 117
 JNI DLL Absolute Pathname 115
 KeyManagerAlgorithm 130
 KeyStore 129
 KeyStorePassword 129
 KeyStoreType 129
 Maximum Heap Size 117
 Maximum Stack Size for JVM Threads 117
 Maximum Stack Size for Native Threads 117
 MessageVersion 133
 OriginatingNode 135
 PassWord (ApplicationMessaging) 132
 PassWord (HTTP Authentication) 126
 PassWord (Proxy Authentication) 125
 Property.Tag 121
 Provider 127
 PublicationID 134
 PublicationProcess 135
 Publisher 135
 Remote debugging port number 119
 Report JVM Info and all Class Loads 118
 RequestVersion 132
 Rollback Wait Interval 120
 SSLServerSocketFactoryImpl 128
 SSLSocketFactoryImpl 128
 SubChannel 134
 Subject 133
 SubjectDetail 133

- Suspend option for debugging 119
- ToNode 132
- TrustManagerAlgorithm 131
- TrustStore 130
- TrustStorePassword 130
- TrustStoreType 130
- type 121
- UseHttpAuthentication 126
- UseProxy 124
- UserName (HTTP Authentication) 126
- UserName (Proxies) 125
- UseSSL 127
- X509CertificateImpl 128
- configuration procedures
 - Inbound e*Way 92, 93
 - Outbound e*Way 98
- Constructors 149–153
- ContentType parameter 122
- conventions, writing in document 9
- Creating a Schema 53

D

- DefaultDataVersion parameter 135
- DefaultURL parameter 122
- Disable JIT parameter 118

E

- e*Way
 - creating 90
 - Installation 16
 - Properties 91
 - Schedules 94
 - Startup Options 94
 - troubleshooting 104
- Entry class 188
- Event Type Definition (ETD) 76

F

- FromNode parameter 132

G

- Garbage Collection Activity Reporting parameter 118
- General Settings configuration 120, 137–139
- getBase64Deflate method 180
- getChannel method 166
- getDefaultDataVersion method 158
- getExceptionDefaultMsg method 183
- getExceptionMsgID method 184

- getExceptionMsgSet method 185
- getExceptionTitle method 186
- getFromNode method 163
- getIsExceptionResponse method 187
- getMessageVersion method 159
- getOriginatingNode method 164
- getPassWord method 165
- getPeopleSoft8AppMsgResponse method 178
- getProperties method 156
- getPublicationID method 171
- getPublicationProcess method 172
- getPublisher method 174
- getPublishingNode method 175
- getPublishTimeStamp method 173
- getRequestVersion method 160
- getResponseCode method 179
- getSubChannel method 167
- getSubject method 169
- getSubjectDetail method 170
- getToNode method 161
- getXMLMessage method 176

H

- HttpProxyHost parameter 124
- HttpProxyPort parameter 124
- HttpsProtocolImpl parameter 127
- HttpsProxyHost parameter 124
- HttpsProxyPort parameter 125

I

- Initial Heap Size parameter 117
- initialize method 154
- Installation procedure
 - e*Way (UNIX) 19
 - e*Way (Windows) 16
 - MUX Handler (UNIX) 24
 - MUX Handler (Windows) 21
- InstallShield 16
- Intelligent Queue (IQ) 78, 104
- isOpen method 155

J

- Java constructors
 - Base64Utils 152, 153
 - PeopleSoft8AppMsg 149
 - PeopleSoft8AppMsgAP 149
 - PeopleSoft8AppMsgConnector 150
 - PeopleSoft8AppMsgException 150, 151
 - PeopleSoft8AppMsgResponse 151
 - PeopleSoft8AppMsgTester 152

Java methods

base64DecodeToByte 181
 byteToBase64String 181
 close 156
 getBase64Deflate 180
 getChannel 166
 getDefaultDataVersion 158
 getExceptionDefaultMsg 183
 getExceptionMsgID 184
 getExceptionMsgSet 185
 getExceptionTitle 186
 getFromNode 163
 getIsExceptionResponse 187
 getMessageVersion 159
 getOriginatingNode 164
 getPassWord 165
 getPeopleSoft8AppMsgResponse 178
 getProperties 156
 getPublicationID 171
 getPublicationProcess 172
 getPublisher 174
 getPublishingNode 175
 getPublishTimeStamp 173
 getRequestVersion 160
 getResponseCode 179
 getSubChannel 167
 getSubject 169
 getSubjectDetail 170
 getToNode 161
 getXMLMessage 176
 initialize 154
 isOpen 155
 main 157
 open 154
 reset 157
 sendMessage 177
 setBase64Deflate 180
 setChannel 166
 setDefaultDataVersion 158
 setExceptionDefaultMsg 182
 setExceptionMsgID 183
 setExceptionMsgSet 184
 setExceptionTitle 185
 setFromNode 162
 setIsExceptionResponse 186
 setMessageVersion 159
 setOriginatingNode 163
 setPassWord 164
 setPeopleSoft8AppMsgResponse 177
 setPublicationID 170
 setPublicationProcess 171
 setPublisher 174
 setPublishingNode 175
 setPublishTimeStamp 173

setRequestVersion 160
 setResponseCode 179
 setSubChannel 167
 setSubject 168
 setSubjectDetail 169
 setToNode 161
 setXMLMessage 176

Java Object Classes

AdministerMuxHandler 189
 AdministerMuxHandlerAddMode 189
 AdministerMuxHandlerDeleteMode 189
 AdministerMuxHandlerEditMode 189
 AdministerMuxHandlerError 189
 Base64Utils 148
 Entry 188
 MuxHandler 190
 MuxHandlerConstants 188
 MuxHandlerEntry 189
 MuxPublicationHandler 189
 PeopleSoft8AppMsg 142
 PeopleSoft8AppMsgAPI 143
 PeopleSoft8AppMsgConnector 144
 PeopleSoft8AppMsgException 145
 PeopleSoft8AppMsgResponse 146
 PeopleSoft8AppMsgTester 147

JNI DLL Absolute Pathname parameter 115

JVM Settings configuration 115–119

K

KeyManagerAlgorithm parameter 130
 KeyStore parameter 129
 KeyStorePassword parameter 129
 KeyStoreType parameter 129

L

logging options 96

M

main method 157
 Maximum Heap Size parameter 117
 Maximum Stack Size for JVM Threads parameter 117
 Maximum Stack Size for Native Threads parameter 117
 MessageVersion parameter 133
 Methods 154–187
 monitoring thresholds 97
 Multi-Mode e*Way configuration
 General Settings 120
 JVM Settings 115–119

Multiplexer e*Way configuration
 General Settings 137–139
 MUX Handler classes
 Descriptions 188–190
 UNIX installation 24
 Windows installation 21
 MUX Instance ID 138
 MUX Recovery ID 139
 MuxHandler class 190
 MuxHandlerConstants class 188
 MuxHandlerEntry class 189
 MuxPublicationHandler class 189

O

Object Classes 141–148
 open method 154
 Oracle 15
 OriginatingNode parameter 135

P

Participating Host 104
 PassWord parameter (ApplicationMessaging) 132
 PassWord parameter (HTTP Authentication) 126
 PassWord parameter (Proxy Authentication) 125
 PeopleSoft8AppMsg Class 142
 PeopleSoft8AppMsg constructor 149
 PeopleSoft8AppMsgAP constructor 149
 PeopleSoft8AppMsgAPI Class 143
 PeopleSoft8AppMsgConnector Class 144
 PeopleSoft8AppMsgConnector constructor 150
 PeopleSoft8AppMsgException Class 145
 PeopleSoft8AppMsgException constructor 150, 151
 PeopleSoft8AppMsgResponse Class 146
 PeopleSoft8AppMsgResponse constructor 151
 PeopleSoft8AppMsgTester Class 147
 PeopleSoft8AppMsgTester constructor 152
 procedures
 configuration, inbound 92, 93
 configuration, outbound 98
 installation 16
 Properties, e*Way 91
 Property.Tag parameter 121
 Provider parameter 127
 psoft8appmsg.def 113
 psoft8appmsg.def. 93
 PublicationID parameter 134
 PublicationProcess parameter 135
 Publisher parameter 135
 Push IP Port 137

Q

Queues 78

R

Remote debugging port number parameter 119
 Report JVM Info and all Class Loads parameter 118
 Request Reply IP Port 137
 RequestVersion parameter 132
 reset method 157
 Rollback if no Clients on Push Port 137
 Rollback Wait Interval parameter 120

S

sapeway.def. 113
 Schedules 94
 Schema, creating 53
 Send Empty MSG When External Disconnect 138
 sendMessage method 177
 setBase64Deflate method 180
 setChannel method 166
 setDefaultDataVersion method 158
 setExceptionDefaultMsg method 182
 setExceptionMsgID method 183
 setExceptionMsgSet method 184
 setExceptionTitle method 185
 setFromNode method 162
 setIsExceptionResponse method 186
 setMessageVersion method 159
 setOriginatingNode method 163
 setPassWord method 164
 setPeopleSoft8AppMsgResponse method 177
 setPublicationID method 170
 setPublicationProcess method 171
 setPublisher method 174
 setPublishingNode method 175
 setPublishTimeStamp method 173
 setRequestVersion method 160
 setResponseCode method 179
 setSubChannel method 167
 setSubject method 168
 setSubjectDetail method 169
 Setting Startup Options or Schedules 94
 setToNode method 161
 setXMLMessage method 176
 SSLServerSocketFactoryImpl parameter 128
 SSLSocketFactoryImpl parameter 128
 Startup Options 94
 stcewipmp.def 92, 114
 SubChannel parameter 134
 Subject parameter 133
 SubjectDetail parameter 133

Suspend option for debugging parameter 119

T

ToNode parameter 132
troubleshooting 104
TrustManagerAlgorithm parameter 131
TrustStore parameter 130
TrustStorePassword parameter 130
TrustStoreType parameter 130
type parameter 121

U

UNIX installation
 e*Way 19
 MUX Handler 24
UseHttpAuthentication parameter 126
UseProxy parameter 124
User name 94
UserName parameter (HTTP Authentication) 126
UserName parameter (Proxies) 125
UseSSL parameter 127

W

Wait For IQ Ack 138
Windows installation
 e*Way 16
 MUX Handler 21

X

X509CertificateImpl parameter 128