e*Way Intelligent Adapter for SAP (BAPI) User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Version 20100713160509.

Contents

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 e*Gate™ Integrator system, and have a working knowledge of:

- Operation and administration of the appropriate operating systems (see **Supported Operating Systems** on page 18)
- Windows-style GUI operations
- SAP BAPI concepts and operations
- Integrating SAP R/3 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 exposed Java methods. 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 SAP BAPI is frequently referred to as the SAP BAPI 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.

```
stcregutil -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 Guide.*
- For more information on the Multi-Mode and File e*Ways, see the *Standard e*Way Intelligent Adapter User's Guide*.

Comprehensive information on SAP R/3 can be found at the SAP Help Portal:

http://help.sap.com

Once you have selected the appropriate SAP R/3 version and language, you are presented with the SAP Library, which contains an index and a search facility (**Find**). Under *Business Framework Architecture*, you can locate information such as:

- BAPI User Guide
- BAPI Programming Guide
- BAPI Enhancements and Modifications

If you have a SAP customer or partner SAPNet user ID, you also can access the following links for additional information on:

- The SAP Java Connector:

  https://www013.sap-ag.de/connectors

- Working with and developing BAPI solutions:

  https://www013.sap-ag.de/BAPI

Chapter 1

Introduction

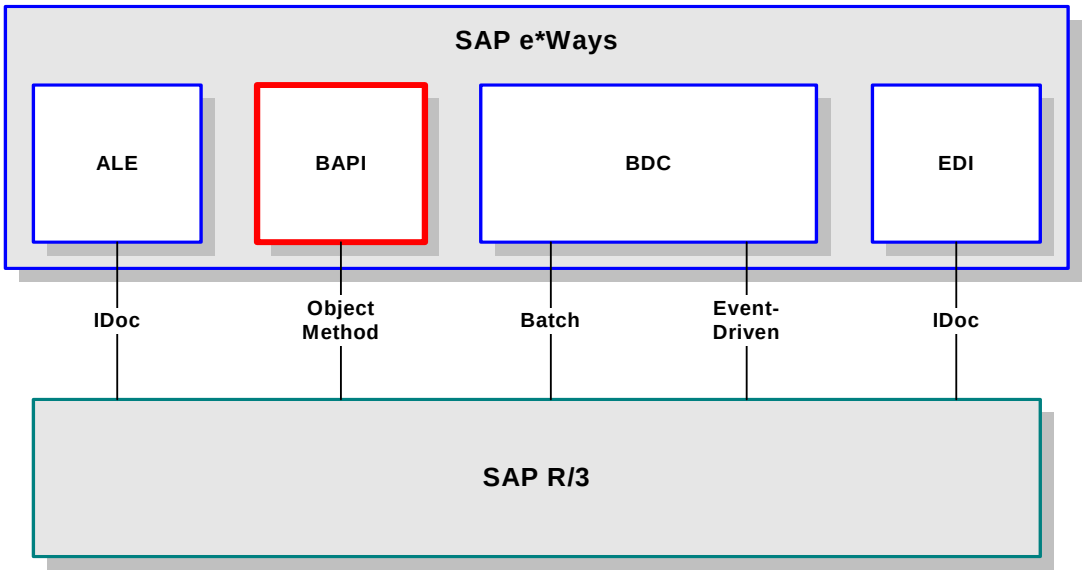
1.1 Overview

The e*Way Intelligent Adapter for SAP BAPI has been designed specifically to connect e*Gate to SAP enterprise-management software within a network of diverse hardware and software systems. Using one or more SAP e*Ways, e*Gate can act as a bus, linking SAP applications and other software systems, or differently-configured SAP systems. This e*Way allows bidirectional data exchange between e*Gate and an SAP system via SAP's Business Application Programming Interface (BAPI).

1.2 SAP Interface Options

SAP offers several interface options, including Application Link Enabling (ALE), Business Application Programming Interface (BAPI), and Electronic Data Interchange (EDI). Batch Data Communication (BDC) actually is a user-emulation method that can be either batch or event-driven.

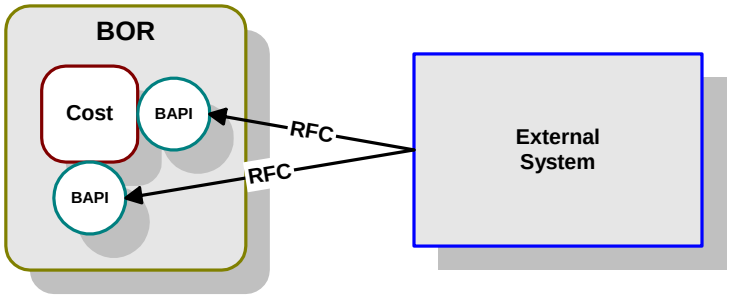
Figure 1 SAP Interface Options



1.2.1 BAPI Methods

BAPI is an acronym for **B**usiness **A**pplication **P**rogramming **I**nterface and is SAP's way of providing precise access to processes and data residing in their system. More specifically, BAPIs are *methods* of SAP Business Objects stored in the Business Object Repository (BOR) and are implemented by function modules programmed in ABAP/4. Also, these functions are Remote Function Call (RFC) enabled, and thus can be called by an external process such as the SAP BAPI e*Way.

Figure 2 BAPI Example



1.2.2 BAPI vs. ALE-IDoc

SAP's ALE-IDoc mechanism also provides access to processes and data residing in the SAP system, but with one major difference: that access is asynchronous. In other words, data retrieved from the SAP system is not guaranteed to be the most current; there could have been several seconds lapse between when data was captured into an IDoc and when it was sent out via ALE to the external system. Also, if data in an IDoc is sent into SAP, it may not be posted immediately into the database tables. Another difference is data communicated through IDocs tend to be overly comprehensive and lengthy.

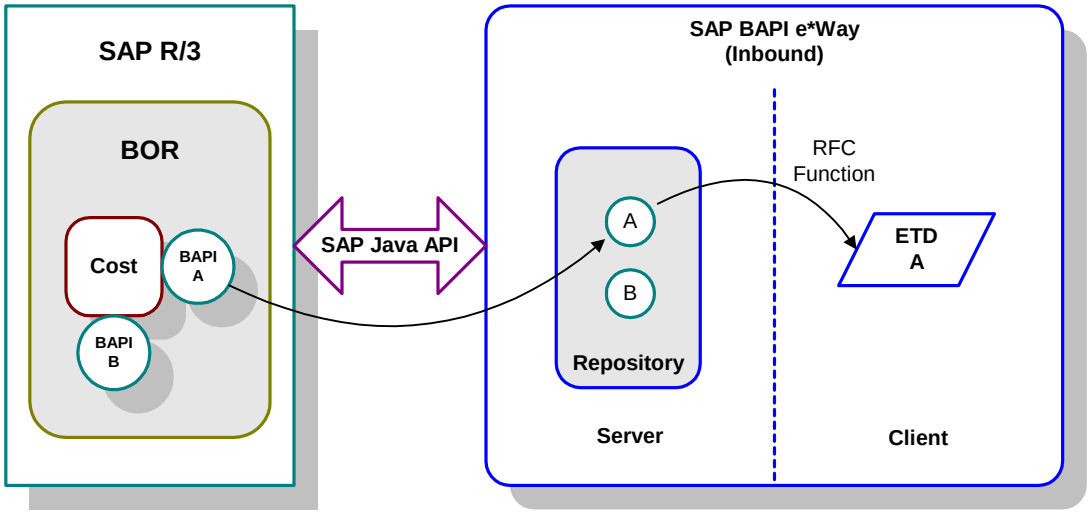
In contrast, BAPIs provide synchronous access to SAP. That is, a request by an external process for data to be retrieved from or posted into the SAP system is executed immediately and control is returned to the caller only when the transaction has completed (or failed). Moreover, the data exchanged is brief as compared with IDocs.

Topic	ALE-IDoc	BAPI
Communication	Asynchronous	Synchronous
Data from SAP	Sending may lag capture into IDoc	Immediate sending
Data to SAP	Posting may lag receipt of data	Immediate posting
Data content	Lengthy and overly comprehensive	Brief and selective

1.3 The SAP BAPI e*Way

The SAP BAPI e*Way serves as the External System of Figure 2. It provides an interface with the SAP R/3 system by means of the SAP Java API set contained in the file **jCO.jar**. Applicable BAPI methods are held in the e*Gate server's repository, and are invoked by an RFC call from the R/3 system. When invoked, they are passed as an RFC Function into an ETD.

Figure 3 BAPI e*Way



The functionality of the SAP BAPI e*Way simplifies the whole process of determining the requisite **IMPORT**, **EXPORT** and **TABLE** parameters—marshalling all the necessary data using the correct type and format, calling the ABAP/4 function module that represents the BAPI, and then extracting and parsing data from the **EXPORT** and/or **TABLE** parameters.

Note: *If you develop custom BAPIs, follow the instructions in the BAPI Programmers Reference to ensure your BAPI has been properly created and released to be visible in the BAPI Browser.*

1.3.1 IDoc Transport

The SAP BAPI e*Way contains a feature that allows you to send and receive IDocs using the IDOC_INBOUND_ASYNCHRONOUS Remote Function Module. This essentially provides a limited emulation of the ALE IDoc process described in the preceding section.

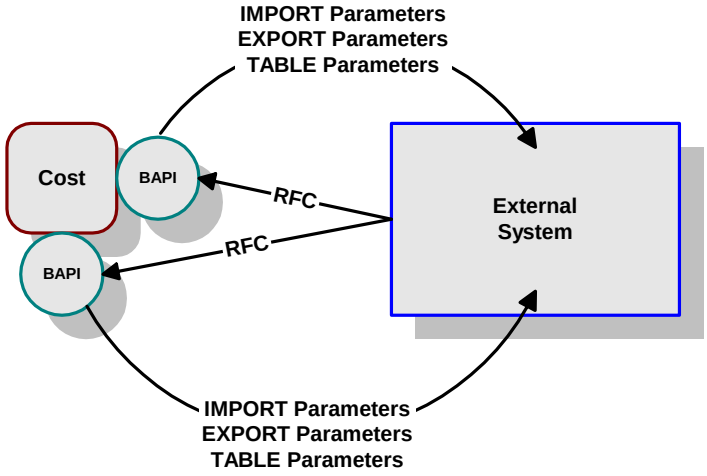
1.3.2 Invoking BAPI Methods

Before it can be invoked, a BAPI (or any RFC-enabled function module on SAP for that matter) requires the following:

- **IMPORT** parameters - data which must be provided to the BAPI
- **EXPORT** parameters - data which is returned by the BAPI
- **TABLE** parameters - data that may be provided to and/or returned by the BAPI

The detailed metadata for these parameters such as descriptions of their value types and mandatory or optional nature, can be found under SAP transaction **SE37**. These metadata, however, can prove tedious to implement without an automated solution such as the SAP BAPI e*Way.

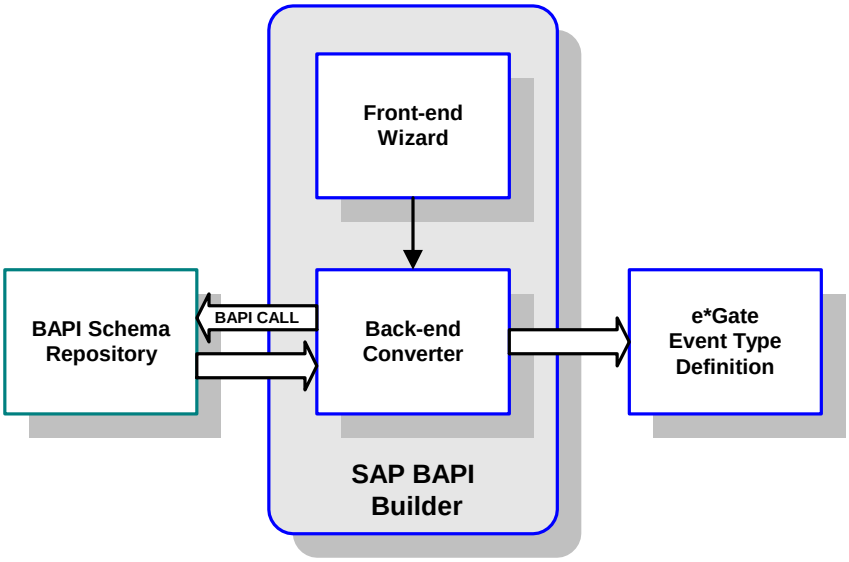
Figure 4 Required Information



1.3.3 Event Type Definitions

Event Type Definitions (ETDs) are templates that embody the business rules related to the associated external application. In the case of the SAP BAPI e*Way, these ETDs contain methods that communicate with corresponding BAPI methods in SAP R/3. These ETDs are built automatically by means of the BAPI Builder, which is invoked from within the ETD Editor (see Figure 5). A front-end Wizard provides the user interface.

Figure 5 BAPI Builder Operation



The functionality of the SAP BAPI e*Way simplifies the whole process of determining the requisite **IMPORT**, **EXPORT** and **TABLE** parameters—marshalling all the necessary data using the correct type and format, calling the ABAP/4 function module that represents the BAPI, and then extracting and parsing data from the **EXPORT** and/or **TABLE** parameters.

1.3.4 Components

The SAP BAPI e*Way uses the Multi-Mode e*Way executable, **stceway.exe**, which is installed with e*Gate Integrator. It also makes use of two ETD builders, the **BAPI Wizard** and the **IDoc Wizard**, also installed with e*Gate Integrator. The e*Way installation furnishes:

- An e*Way Connection Definition template, **BapiConnector.def**
- An e*Way Configuration ETD, **BapiConfiguration.xsc**
- Java Archive (**.jar**) files containing essential Java object methods
- Associated control files and library files

Sample schemas are also provided for demonstration and system testing purposes, and can serve as templates for custom schemas (see **Sample Schemas** on page 97).

1.3.5 Supported Operating Systems

The e*Way Intelligent Adapter for SAP BAPI supports the following releases of SAP R/3: 4.0B, 4.5B, 4.6B, 4.6C, and 4.7. For information about supported operating systems, see the **readme.txt** file provided on the installation CD. The SAP BAPI e*Way is not supported on Solaris 10 (AMD and Intel), and supports Japanese and Korean versions of the operating systems only in combination with SAP R/3 releases 4.6C and 4.7.

Chapter 2

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\ewsapbapi directory on the installation CD-ROM for important information regarding this installation.*

2.1 System Requirements

To use theSAP BAPI e*Way, you need the following:

1. An e*Gate Participating Host.
2. A TCP/IP network connection.
3. Sufficient free disk space (about 1.5 MB) to accommodate e*Way files (not including sample schemas).

   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. A minimum of 500 MB of free space is suggested for running a schema; more, if logging is enabled.

2.2 External System Requirements

- SAP R/3 release 4.0B, 4.5B, 4.6B, 4.6C, and 4.7 (English)
- SAP Java Connector (JCo) version 2.1.5. The eWay supports connectivity to SAP using SAP JCo 2.1.5.

Note: *SAP BAPI eWays can run on a 64-bit JVM, but only after the correct 64-bit JCo files have been applied. SAP customers who use 64-bit JVM must download the JCo files from the Web site for SAP Service Marketplace.*

2.2.1 Environment Configuration

You must install the SAP Java Connector (JCo) appropriate to your operating system. This contains the APIs used to connect to SAP R/3, and must be obtained directly from SAP. See **Installing SAP Java Connector Version 2.1.5** on page 25.

2.2.2 Security Issues

To provide an acceptable and secure business-to-business environment, SAP recommends that the e*Gate user be of a special user class to limit interaction with the system. This class is either:

- **System** (in recent SAP R/3 versions)
- **CPI-C** (in earlier SAP R/3 versions)

These user classes are limited to external applications, since neither can log on to a SAPGUI session.

2.3 Installing the e*Way

2.3.1 Windows Systems

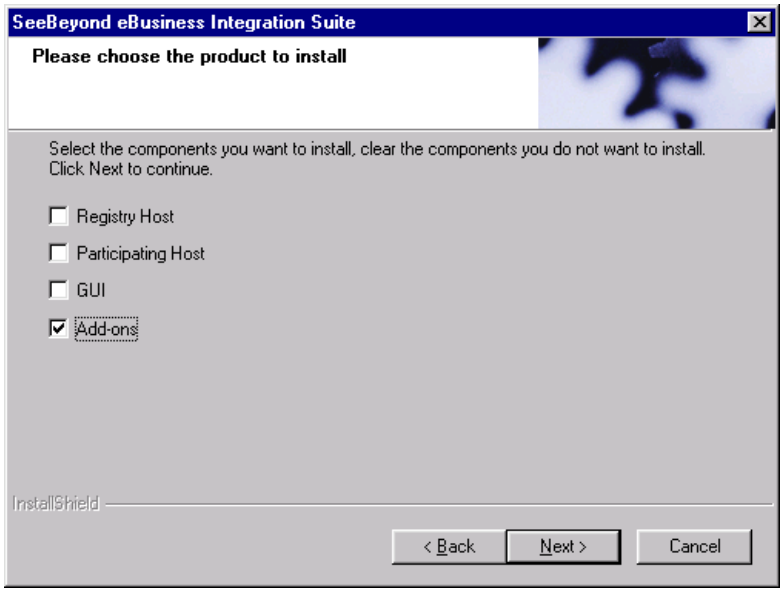
Installation Procedure

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

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 6). Check **Add-ons**, then click **Next**.

Figure 6 Choose Product Dialog Box

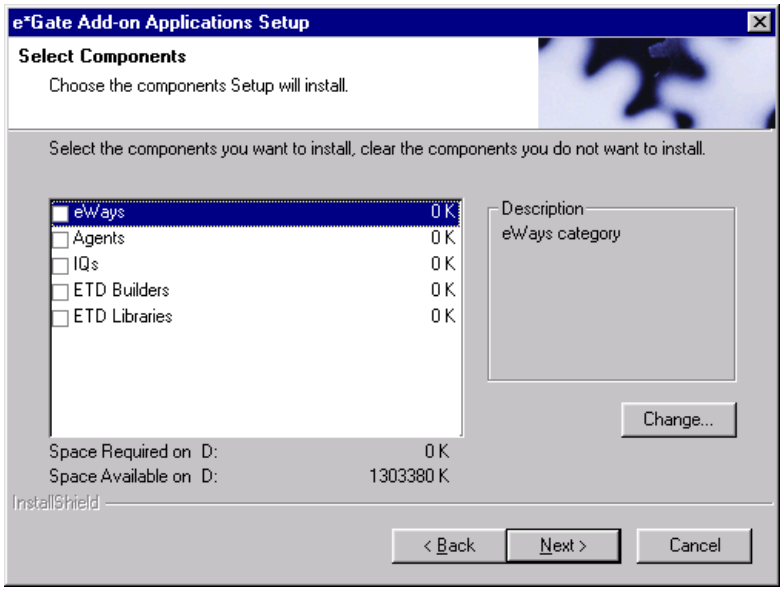


   - B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (bypassing the **Choose Product** dialog):

```
setup\addons\setup.exe
```

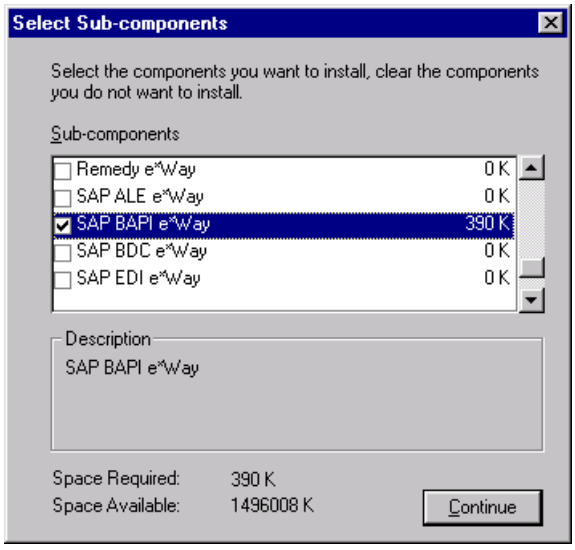
5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 7). Highlight—*but do not check*—**eWays** and then click **Change**.

Figure 7 Select Components Dialog Box



6 When the **Select Sub-components** dialog box appears (see Figure 8), check the **SAP BAPI e*Way**.

Figure 8 Select e*Way Dialog



7 Click **Continue**, and the **Select Components** dialog box reappears.

8 Click **Next** and continue with the installation.

Subdirectories and Files

Note: *Installing the e*Way Intelligent Adapter for SAP BAPI 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 1 Participating Host & Registry Host

Subdirectories	Files
\bin\	librfc32.dll
\classes\	stcewcommoneway.jar
\configs\sapbapi\	BapiConnector.def
\etd\	bapi.ctl bapiwizard.ctl
\etd\sapbapi\	BapiConfiguration.xsc
\ThirdParty\sun\	jta.jar

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

Table 2 Registry Host Only

Subdirectories	Files
\	stcewsapbapi.ctl

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.*

To Install the e*Way on a UNIX System

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 Oracle. Note also that* ***no spaces*** *should appear in the installation path name.*

Subdirectories and Files

Note: *Installing the e*Way Intelligent Adapter for SAP BAPI 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 3 Participating Host & Registry Host

Subdirectories	Files
/bin/	librfccm.sl (HP-UX only)
/classes/	stcewcommoneway.jar
/configs/sapbapi/	BapiConnector.def
/etd/	bapi.ctl bapiwizard.ctl
/etd/sapbapi/	BapiConfiguration.xsc
/ThirdParty/sun/	jta.jar

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

Table 4 Registry Host Only

Subdirectories	Files
/	stcewsapbapi.ctl

2.4 Installing SAP Java Connector Version 2.1.5

This sections contain directions specific to installing SAP Java Connector version 2.1.5 to your Windows or Unix platform. Once you have installed the e*Way, install the SAP Java Connector (JCo). A JCo archive file must be obtained directly from SAP.

- If you have a secure ISDN line, you can download from the FTP site (**sapserv3** or **sapserv4**). Consult your SAP Account Representative regarding access.
- If you have a SAPNet user ID, you can access the following link. Select **SAP Java Connector**, then **Download**.

  https://websmp206.sap-ag.de/connectors

Add the required DLL and JAR files to your system

1. Copy the following SAP DLL files to your shared library path:

   For **Windows** (add the required DLL files to *WINNT\system32*):

   - librfc32.dll
   - sapjcorfc.dll

   For **UNIX**:

   - librfccm.*
   - libsapjcorfc.*

   Use the following UNIX file extensions: *.so for Solaris, *.sl for HP-UX, and *.0 for AIX.

2. Download the following DLL files. These are available, free of charge, from various sources on the Internet:

   - MSVCP71.DLL
   - msvcr71.dll

   Copy these DLL files to your ***WINNT\system32*** directory.

3. Locate and copy the SAP JAR file, **sapjco.jar**, to the following directorys:

   ```
   <eGate>\client\ThirdParty\SAP
   ```

   and

   ```
   <eGate>\Server\registry\repository\default\ThirdParty\SAP
   ```

   where <eGate> is the folder where you installed eGate Integrator.

Note: *When using SAP Java Connector 2.1.5, all Java Collaborations must have the new SAP JCo files added to the Java Collaboration classpath. For more information see* **Adding Third-party JAR Files to the Collaboration Classpath** *on page 79.*

2.5 Optional Example Files

The installation CD contains two sample schema files located in the **samples\ewsapbapi** directory. Both schemas operate in Client and Server modes.

- **BapiJava.zip**

  Standard BAPI sample

- **TrfcAleBapiJava.zip**

  tRFC IDoc sample, using the IDOC_INBOUND_ASYNCHRONOUS BAPI

To use these schemas, you must load them onto your system using the following procedure. See **Sample Schemas** on page 97.

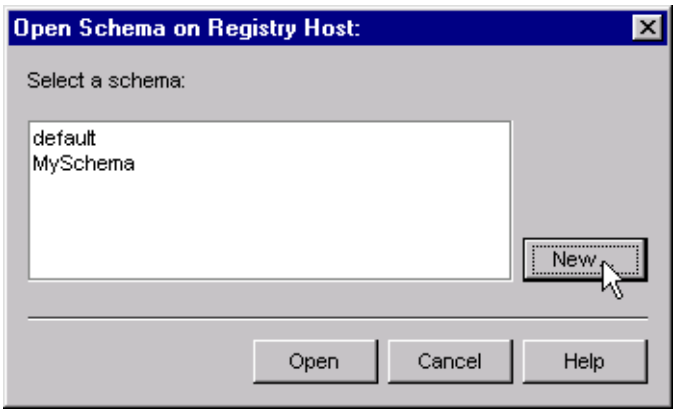
Note: *The SAP BAPI e*Way must be properly installed on your system before you can run the sample schema.*

2.5.1 Installation Procedure

To load a sample schema

1 Invoke the **Open Schema** dialog box and select **New** (see Figure 9).

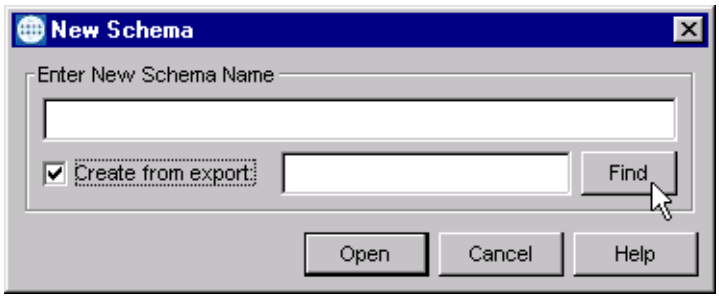
Figure 9 Open Schema Dialog



2 Type the name you want to give to the schema (for example, **BapiJavaSample**)

3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 10).

Figure 10 New Schema Dialog



4 Select the desired archive file (*.**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* ***.zip** *to a local directory and extract the files. Using a text editor, edit the file* ***.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. The **\.lock** subdirectories are not shown.

Table 5 Subdirectories and Files - BAPI Sample (BapiJava)

Subdirectories	Files
\runtime\	BapiJava.ctl
\runtime\configs\messageservice\	cpJMSServer.cfg cpJMSServer.sc
\runtime\configs\sapbapi\	cpC4XClient.cfg cpC4XClient.sc cpC4XServer.cfg cpC4XServer.sc
\runtime\configs\stcewfile\	ewCostCenterClientEater.cfg ewCostCenterClientEater.sc ewCostCenterClientFeeder.cfg ewCostCenterClientFeeder.sc ewCostCenterServerFeeder.cfg ewCostCenterServerFeeder.sc
\runtime\data\	clientdata.fin serverdata.fin

Table 5 Subdirectories and Files - BAPI Sample (BapiJava)

Subdirectories	Files
\sandbox\Administrator\collaboration_rules\	crCostCenterClient.class crCostCenterClient.ctl crCostCenterClient.java crCostCenterClient.xpr crCostCenterClient.xts crCostCenterClientBase.class crCostCenterServer.class crCostCenterServer.ctl crCostCenterServer.java crCostCenterServer.xpr crCostCenterServer.xts crCostCenterServerBase.class crFeeder.class crFeeder.ctl crFeeder.java crFeeder.xpr crFeeder.xts crFeederBase.class
\sandbox\configs\messageservice\	JMSServer_eWc.cfg JMSServer_eWc.sc
\sandbox\etd\	ClientOutput.jar ClientOutput.ssc ClientOutput.xsc
\sandbox\etd\sapbapi\	BapiConfiguration.xsc ClientData.jar ClientData.ssc ClientData.xsc CostCenter.jar CostCenter.xsc ServerData.jar ServerData.ssc ServerData.xsc

Table 6 Subdirectories and Files - IDoc Sample (TrfcAleBapiJava)

Subdirectories	Files
\runtime\	BAPIIDOC.ctl
\runtime\collaboration_rules\bapitest\	crALEBapiClient.class crALEBapiClient.ctl crALEBapiClient.java crALEBapiClient.xpr crALEBapiClient.xts crALEBapiClientBase.class crALEBapiServer.class crALEBapiServer.ctl crALEBapiServer.java crALEBapiServer.xpr crALEBapiServer.xts crALEBapiServerBase.class
\runtime\configs\messageservice\	cpJMSServer.cfg cpJMSServer.sc
\runtime\configs\sapbapi\	cpC4XClient.cfg cpC4XClient.sc cpC4XServer.cfg cpC4XServer.sc cpS4XServer.cfg cpS4XServer.sc
\runtime\configs\stcewfile\	ewALEBapiClient.cfg ewALEBapiClient.sc ewEater.cfg ewEater.sc
\runtime\data\	CREMAS03.dat
\sandbox\Administrator\etd\idoc\	IDOC_CREMAS03_46C.jar IDOC_CREMAS03_46C.xsc IDOC_CREMAS03_46C_451.jar IDOC_CREMAS03_46C_451.xsc IDOC_CREMAS03_46C_452.jar IDOC_CREMAS03_46C_452.xsc IDOC_INBOUND_ASYNCHRONOUS.jar IDOC_INBOUND_ASYNCHRONOUS.xsc
\sandbox\Administrator\etd\sapbapi\	BapiConfiguration.xsc

Note: *In this sample schema,* IDOC_CREMAS03_46C.xsc *is the default ETD file. If you run this schema under e*Gate 4.5.2 or 4.5.1, you must replace this file with* IDOC_CREMAS03_46C_452.xsc *or* IDOC_CREMAS03_46C_451.xsc*, respectively.*

Chapter 3

e*Way Configuration

This chapter describes the configuration parameters for the Java SAP BAPI e*Way Connections.

3.1 Overview

3.1.1 Multi-Mode e*Way

The e*Way's inherent configuration parameters are set using the e*Way Configuration Editor; see **Configuring the e*Way** on page 146 for procedural information. The default configuration is provided in **sapeway.def**. The SAP BAPI e*Way's configuration parameters are organized into the following sections:

3.1.2 e*Way Connections

An e*Way Connection's configuration parameters also are set using the e*Way Configuration Editor; see **Creating e*Way Connections** on page 154 for procedural information. The default configuration is provided in **BapiConnector.def**. The SAP BAPI e*Way's configuration parameters are organized into the following sections:

3.2 Multi-Mode e*Way

3.2.1 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
Windows	jvm.dll
Solaris	libjvm.so
Linux	libjvm.so
HP-UX	libjvm.sl
AIX	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 not 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**.

3.2.2 General Settings

These parameters are used only with e*Gate 4.5.3 and later, and have not been tested with this e*Way.

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.

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

Standard IQ FIFO

Description

Specifies whether or not to receive messages from STC_Standard IQs in first-in, first-out (FIFO) order.

Required Values

Yes or **No**; the default value is **No**.

3.3 e*Way Connections

3.3.1 Server

The parameters in this section apply to RFC Server-mode operation.

Gateway Hostname

Description

Specifies the host name of the Gateway for the SAP R/3 system.

Required Values

A valid host name. Do *not* include any Router string.

Router String

Description

This parameter allows access to an SAP system that is located behind a firewall. The string is composed of the hostnames or IP addresses of all SAP Routers that are between this e*Way and the SAP Gateway Host.

For example, if there are two routers, saprouter1, and saprouter2 (in order) from the e*Way to the SAP Gateway Host, the Router String is determined as follows:

```
saprouter1:       204.79.199.5
saprouter2:       207.105.30.146
Router String:    /H/204.79.199.5/H/207.105.30.146/H/
```

Note: *You must include the* **/H/** *tokens to separate the routers, as well as at the beginning and ending of the string (as shown above).*

Required Values

A valid router string, as shown in the example above.

Note: *This parameter is conditional and needs to be specified* ***only*** *to gain access to an SAP system that is behind a firewall.*

Gateway Service

Description

Specifies the Gateway Service of the R/3 Target System that is sending transactions. This is equivalent to a TCP/IP Port Number and can be referenced in the following file on the SAP R/3 System:

- **Unix:**

```
/etc/services
```

- **Windows:**

```
C:\Winnt\system32\drivers\etc\Services
```

Required Values

The SAP-recommended value is the string **"sapgw"** concatenated with the SAP System Number.

For example, with a System Number of **"01"**, the Gateway Service would be **"sapgw01"**

Additional Information

If the SAP GUI is not installed, the Service Name/Port Number correlation may not be accessible. In this case, the most likely correlation is as follows, where **XX = 00** through **99**:

Service	Port
sapgwXX	33XX/tcp

Program ID

Description

Specifies the Program ID for the SAP service.

Required Values

A valid Program ID

Note: *This value is case-sensitive, and must match **exactly** the value shown in SAP transaction* **SM59**.

Wait for Request Interval

Description

Specifies the amount of time to wait for a request from the caller of a RFC function installed on this e*Way. If a request is received within this interval, the Java handler for the called RFC function is dispatched immediately. Otherwise, the next **Wait for Request Interval** will occur after the Event Type **get** interval (as configured in the e*Way Connection Properties for this e*Way Connection) has elapsed.

Required Values

A number from **1** through **2147483647**; the default value is **3000**.

Units

Milliseconds, **Seconds**, or **Minutes**; the default unit is **Milliseconds**.

Enable RFC Trace

Description

Activates or deactivates the SAP RFC Trace feature.

Required Values

Off or **On**; the default is **Off**.

Transactional Mode

Description

Specifies the use of either internal single-phase transactional behavior or Transactional RFC (tRFC). Transactional RFC (tRFC) employs unique Transaction IDs (TIDs) to avoid message repeats by tracking communications with the SAP R/3 system.

Required Values

Internal or **Transactional RFC** (**tRFC**); the default is **Internal**.

TID Manager Class

Description

Specifies the type of Transactional ID manager.

Required Values

- **File Based Database**

  The default value is **com.stc.jcsre.bapi.FileTIDManagerImpl**.

Transaction ID Verification Database

Description

Specifies the pathname to the database file in which the disposition of all transactions outgoing from this e*Way is recorded. Transactions are recorded as being:

- **C** (Committed)
- **U** (Unprocessed or rolled back)
- **R** (Reserved or pending)

This parameter is ignored if tRFC is not enabled.

Required Values

A valid path name. If you provide a filename that is *not* an absolute path, then the value of **SystemData** from **.egate.store** is prefixed to the value of the parameter.

For example, if SystemData = **home****eGate**, then:

a value such as **data\SapTRFC.TIDdb**

becomes **\home\eGate\data\SapTRFC.TIDdb**

Additional Information

With the sample TID management module delivered with the SAP BAPI e*Way, if Transactional RFC is enabled, then each instance of the BAPI e*Way must have its own Transaction ID Verification Database file.

3.3.2 Client

The parameters in this section apply to RFC Client-mode operation.

Use Load Balancing

Description

Beginning with SAP R/3 release 3.0C, load balancing can be used to automatically route requests to the Server (within a group of Application Servers) that currently has the best response time, as determined by an SAP Message Server. This parameter selects or deselects the load balancing feature.

Required Values

Yes or **No**; the default is **No**.

Application Server Hostname

Description

Specifies the host name of the R/3 target system.

Required Values

A valid host name. Do *not* include any Router string.

- If Load Balancing is used, this value represents the **SAP Message Server**
- If Load Balancing is ***not*** used, this value represents the **SAP Application Server**

Router String

Description

This parameter allows access to an SAP system that is located behind a firewall. The string is composed of the host names or IP addresses of all SAP Routers that are between this e*Way and the SAP Gateway Host.

For example, if there are two routers, saprouter1, and saprouter2 (in order) from the e*Way to the SAP Gateway Host, the Router String is determined a follows:

```
saprouter1:       204.79.199.5
saprouter2:       207.105.30.146
Router String:    /H/204.79.199.5/H/207.105.30.146/H/
```

Note: *You must include the* **/H/** *tokens to separate the routers, as well as at the beginning and ending of the string (as shown above).*

Required Values

A valid router string, as shown in the example above.

Note: *This parameter is conditional and needs to be specified* ***only*** *to gain access to an SAP system that is behind a firewall.*

System Number

Description

Specifies the system number of the R/3 Application Server.

Required Values

A valid system number.

Note: *This parameter is conditional, and should be configured only when* ***not*** *using SAP Load Balancing.*

Client

Description

Specifies the SAP Client Number used to access the SAP system.

Required Values

An alphanumeric string, including any leading zeros.

User

Description

Specifies the SAP Client User Name.

Required Values

A valid user name.

Note: *This parameter must be defined* ***before*** *specifying the Password.*

Password

Description

Specifies the password associated with the specified user name.

Required Values

A valid password.

Note: *JCo 2.1.5 does not support mixed case and the user must enter passwords in upper case for all design time and runtime SAP connection configurations.*

Language

Description

Specifies the language used in the SAP system.

Required Values

DE (German), **EN** (English) or **JA** (Japanese); the default is **EN**.

Enable RFC Trace

Description

Activates or deactivates the SAP RFC Trace feature. The trace files are created in the directory where the control broker is started, and have the format **rfc<number>.trc**.

Required Values

Yes or **No**; the default is **No**.

Enable ABAP4 Debug Window

Description

Enabling this option launches an SAP ABAP/4 Debugging window on the Participating Host. The window relates to whatever RFC-enabled ABAP/4 Program is currently being called on the R/3 system.

Required Values

Yes or **No**; the default is **No**.

Note: *The SAP Front-end GUI software must be installed on the e*Way's Participating Host for this feature to be used.*

Gateway Hostname

Description

Specifies an Optional Gateway Host Name for the R/3 Target System.

Required Values

A string specifying the host name—do *not* include any Router string.

Note: *This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.*

Gateway Service

Description

Specifies the Optional Gateway Service for the R/3 Target System. This is equivalent to a TCP/IP Port Number and can be referenced on the SAP R/3 System in the file:

- **Unix:**

  `/etc/services`

- **Windows:**

  `C:\Winnt\system32\drivers\etc\Services`

Required Values

The SAP-recommended value is the string **"sapgw"** concatenated with the SAP System Number.

For example, with a System Number of **"01"**, the Gateway Service would be **"sapgw01"**

Note: *This parameter is conditional, and should be configured only when **not** using SAP Load Balancing.*

System ID

Description

Specifies the ID of the SAP R/3 System. Typically, this is the Oracle back-end Database name, and can be found using SAP transaction **S000** (main menu) and following the menu path **System** > **Status** > **Database Data** > **Name**.

Required Values

A valid ID string.

Note: *This parameter is conditional, and should be configured **only** when using SAP Load Balancing.*

Message Server Hostname

Description

Specifies the host name of the R/3 Message Server, in a load-balancing scenario.

Required Values

A valid server host name. Do *not* include any Router string.

Note: *This parameter is conditional, and should be configured **only** when using SAP Load Balancing.*

Application Server Group

Description

Specifies the name of the group of SAP Application Servers that shares the workload.

Required Values

A valid server group name string. Do *not* include any Router string.

Note: *This parameter is conditional, and should be configured* ***only*** *when using SAP Load Balancing.*

Transactional Mode

Description

Specifies the type of mode.

- **Non-Transactional**

  Actions performed by BAPI calls are committed immediately and automatically by SAP (autocommit).

- **Transactional RFC (tRFC)**

  The e*Way communicates with the SAP R/3 system using unique Transaction IDs (TID), to avoid message repeats. This setting is also used for Queued Transactional RFC (qRFC), which consists of ordered tRFC calls.

- **Via COMMIT/ROLLBACK BAPI**

  Performs a single-phase commit, wherein actions performed by BAPI calls are committed or rolled back by calling the BAPI_TRANSACTION_COMMIT or BAPI_TRANSACTION_ROLLBACK BAPIs, respectively.

- **XA-compliant**

  Performs a two-phase commit by emulating an XA Resource Manager.

Required Values

One of the above modes; the default is **Non-Transactional**.

Additional Information

When setting these parameters manually in the Collaboration (under **$bapiConfiguration**), you must enter them *exactly* as they appear above (and in the GUI). For example, when specifying **Transactional RFC** you must also include the parenthetical clause (**tRFC**), as it appears above.

TID Manager Class

Description

If using Transactional RFC mode, specifies the type of Transactional ID manager.

Required Values

- File Based Database:

  The default value is **com.stc.jcsre.bapi.FileTIDManagerImpl**.

Note: *This parameter should be selected only if using Transactional RFC (tRFC).*

Transaction ID Verification Database

Description

If using Transactional RFC mode, specifies the pathname to the database file in which the disposition of all transactions outgoing from this e*Way is recorded. Transactions are recorded as being:

- **C** (Committed)
- **U** (Unprocessed or rolled back)
- **R** (Reserved or pending)

This parameter is ignored if tRFC is not enabled.

Required Values

A valid pathname. If you provide a filename that is *not* an absolute path, then the value of **SystemData** from **.egate.store** is prefixed to the value of the parameter.

For example, if SystemData = **\home\eGate**, then:

a value such as **data\SapTRFC.TIDdb**

becomes **\home\eGate\data\SapTRFC.TIDdb**

Additional Information

The Transaction ID management module, which includes the TID management Monk functions, is a customer-replaceable module. With the sample TID management module delivered with the SAP BAPI e*Way, if Transactional RFC is enforced it is required that each instance of the BAPI e*Way needs to have its own Transaction ID Verification Database file. The result is undefined if two or more e*Ways share a TID database file.

Note: *This parameter should be selected only if using Transactional RFC (tRFC).*

Maximum TID Database Rows

Description

If using Transactional RFC (tRFC) mode, specifies the maximum number of rows in the outbound TID Database that will be kept before the oldest rows are purged and their corresponding TIDs confirmed on SAP R/3. Confirmation allows SAP R/3 to remove those TIDs from its TID tracking database and reduce resource consumption.

Required Values

It is recommended that one row per Collaboration thread be used in calculating this limit. Since tRFC is not inherently XA-compliant, repeats of the last Event (sent from e*Gate) can occur. However, storing at least the last EID (Event ID) sent in the TID database allows the same TID to be sent to SAP R/3 during the execution call and thus SAP will know the message is a duplicate and ignore it.

The default is **200**.

3.3.3 connector

The parameters in this section apply to the SAP connector.

type

Description

Specifies the connector type.

Required Values

A valid connector type; the default value is **bapi**.

class

Description

Specifies the implementing class for the specified type.

Required Values

A valid class designation; the default value is **com.stc.jcsre.bapi.BapiConnector.**

Share Connector Within Collaboration

Description

Specifies whether or not the ETDs within a Collaboration share a single e*Way Connection.

Note: *Sharing this connection is confined to the containing collaboration only. Other connections based off the same configuration file in other collaborations will be themselves different instances.*

Required Values

Yes or **No**; the default value is **No**. Select **Yes** only if *all* the ETDs within the Collaboration are to share the same BAPI e*Way Connection.

Connection Establishment Mode

Description

Specifies how the connection with the database server is established and closed.

- **Automatic** indicates that the connection is automatically established when the collaboration is started, and maintains the connection as needed.
- **OnDemand** indicates that the connection is established on demand as business rules requiring a connection to the external system are performed. The connection is closed once the methods are complete.

- **Manual** indicates that the user will explicitly call the connection open and close methods in the collaboration as business rules.

Required Values

Automatic, **OnDemand**, or **Manual**; the default value is **Automatic**.

Note: *This option is ignored in e*Gate release 4.5.1 and earlier.*

Additional Information

When in **Manual** mode, and setting parameters manually in the Collaboration (under **$bapiConfiguration**), you must enter them *exactly* as they appear in **Transactional Mode** on page 45 (and in the GUI). For example, when specifying **Transactional RFC** you must also include the parenthetical clause (**tRFC**).

Connection Inactivity Timeout

Description

Specifies timeout in milliseconds for the **Automatic** connection establishment mode. If it is not set, or set to zero, the continuous connection will not timeout due to inactivity. However if the connection goes down, it will automatically attempt to reestablish the connection. If a nonzero value is specified, the connection manager monitors for any inactivity and stops the connection if it reaches the specified value.

Required Values

An integer in the range of **0** to **864000**.

Note: *This option is ignored in e*Gate release 4.5.1 and earlier.*

Connection Verification Interval

Description

Specifies the minimum period of time in milliseconds between checks for connection status to the LDAP server. If the connection to the server is detected to be down during verification, the collaboration's **onConnectionDown** method is called. If the connection comes from a previous connection error, the collaboration's **onConnectionUp** method is called.

Required Values

An integer in the range of **0** to **864000**; the default is **60000**.

Note: *This option is ignored in e*Gate release 4.5.1 and earlier.*

Chapter 4

System Implementation

In this chapter we summarize the procedures required for implementing a working system incorporating the Java e*Way Intelligent Adapter for SAP BAPI. Please see the *e*Gate Integrator User's Guide* for additional information.

4.1 eWay Considerations

JCo 2.1.5 does not support mixed case and the user must enter passwords in upper case for all design time and runtime SAP connection configurations.

4.2 Overview

This e*Way provides a specialized transport component for incorporation into an operational Schema. The schema also contains 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.

4.2.1 Pre-Implementation Tasks

Installation of eBI Software

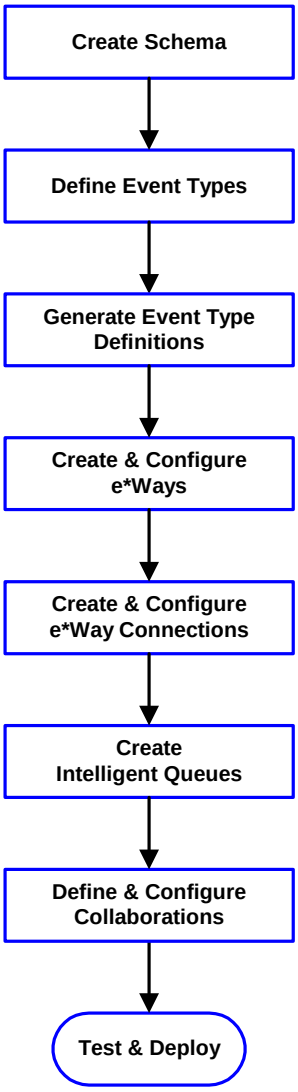
The first task is to install the eBI software as described in **Chapter 2**.

Importation of 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.*

4.2.2 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 52).

2 The second step is to define the Event Types you are transporting and processing within the Schema (see **Creating Event Types** on page 53).

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 **Creating Event Type Definitions** on page 54).

4 The fourth step is to create and configure the required e*Ways (see **Creating and Configuring e*Ways** on page 144).

5 The fifth step is to configure the e*Way Connections (see **Creating e*Way Connections** on page 154).

6 Now you need to create Intelligent Queues to hold published Events (see **Creating Intelligent Queues** on page 66

7 Next you need to define and configure the Collaborations between Event Types (see **Defining Collaborations** on page 67).

8 Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

4.2.3 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.

4.3 Creating a Schema

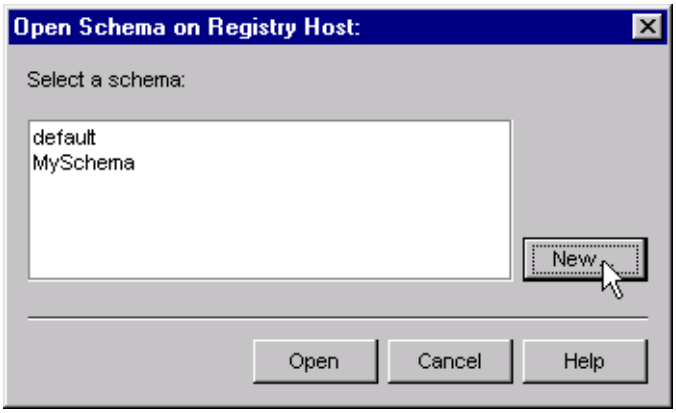
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

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

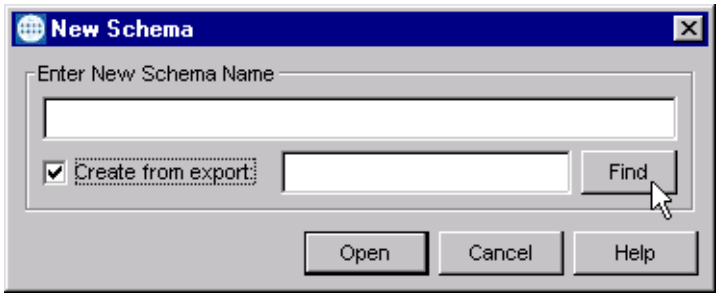
1 Invoke the **Open Schema** dialog box (Figure 11) and **Open** an existing schema, or click **New** to create a new schema.

Figure 11 Open Schema Dialog



2 Clicking **New** invokes the **New Schema** dialog box (Figure 12).

Figure 12 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 **Java**.

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.

4.4 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**.

4.5 Creating Event Type Definitions

Event Type Definitions (ETDs) are templates that embody the business rules related to the associated external application, and define 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. Java ETDs are given the **.xsc** file name extension.

4.5.1 BAPI ETDs

BAPI ETDs contain methods that communicate with corresponding BAPI methods in SAP R/3. These ETDs are created automatically using the BAPI Wizard, as described in **Using the BAPI Wizard** on page 55.

Note: *The BAPI ETD serves as a front end to accessing APIs that communicate with SAP, either to call and receive data from SAP ABAP/4 functions or be called by and subsequently return data to ABAP/4 functions. Hence, the ETD is* ***not*** *marshalable.*

4.5.2 IDoc ETDs

The SAP BAPI e*Way allows you to send and receive IDocs using the SAP RFM IDOC_INBOUND_ASYNCHRONOUS. To send or receive IDocs using this method, you must generate an ETD for IDOC_INBOUND_ASYNCHRONOUS using the BAPI Wizard, and an ETD for your IDoc using the IDoc Wizard (see **Using the IDoc Wizard** on page 59).

4.5.3 Custom ETDs

In most schemas, including the sample schemas included with this e*Way, you will also make use of the File e*Way. ETDs for the File e*Way are built using the Custom ETD Wizard, as described in **Using the Custom ETD Wizard** on page 63. For additional information on the File e*Way, see the *Standard e*Way Intelligent Adapter User's Guide*.

4.5.4 Using the BAPI Wizard

The BAPI Wizard takes a BAPI or RFC object and converts it to a **.xsc** file.

Note: *Package names and root node names in the message structure must be in English (DBCS is not currently supported).*

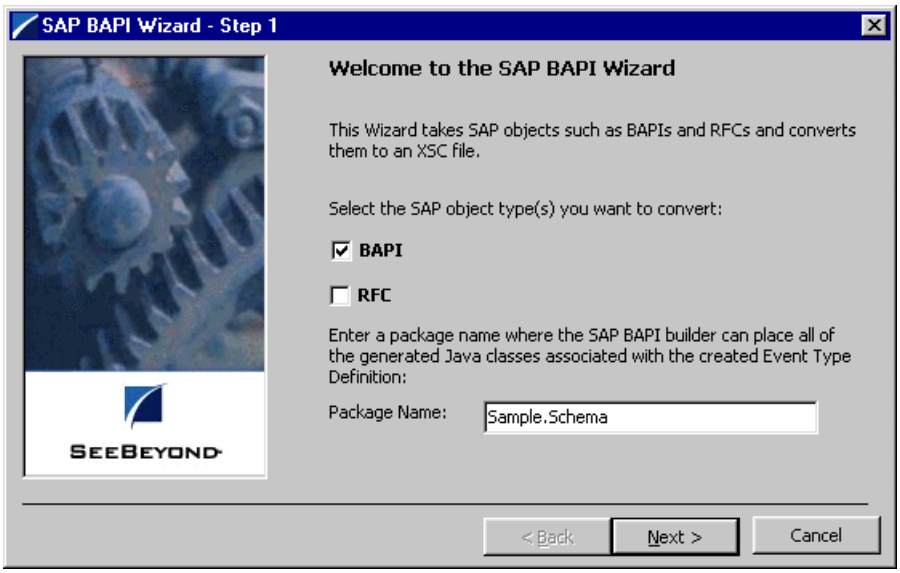
To prepare the BAPI Wizard to generate an ETD

1. Start the e*Gate Schema Designer, and open the schema for which you want to create a BAPI ETD.
2. From the **Options** menu, select **Default Editor** > **Java**.
3. Launch the ETD Editor.
4. Select **New** on the Java ETD Editor's toolbar. The New Event Type Definitions window appears, displaying all installed ETD Wizards.
5. Invoke the BAPI Wizard by selecting its icon (see Figure 13).

Figure 13 BAPI Wizard Icon

The initial Wizard dialog window now appears (see Figure 14).

Figure 14 SAP BAPI Wizard (1)



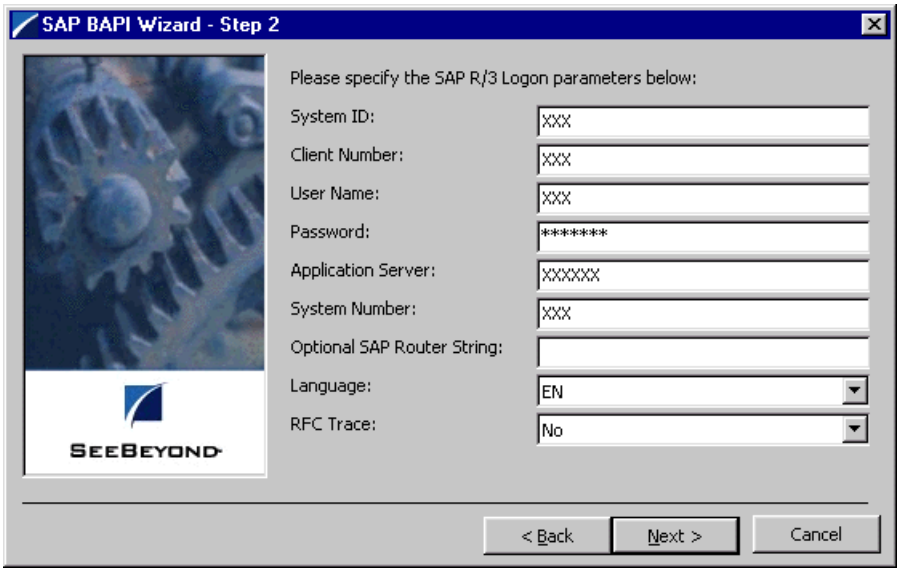
6 Select the type of SAP object to be converted.

- **BAPI**
- **RFC**

7 Enter a Package Name for the container in which the Wizard places the generated Java classes.

8 Click **Next** to view the next Wizard dialog box.

Figure 15 SAP BAPI Wizard (2)

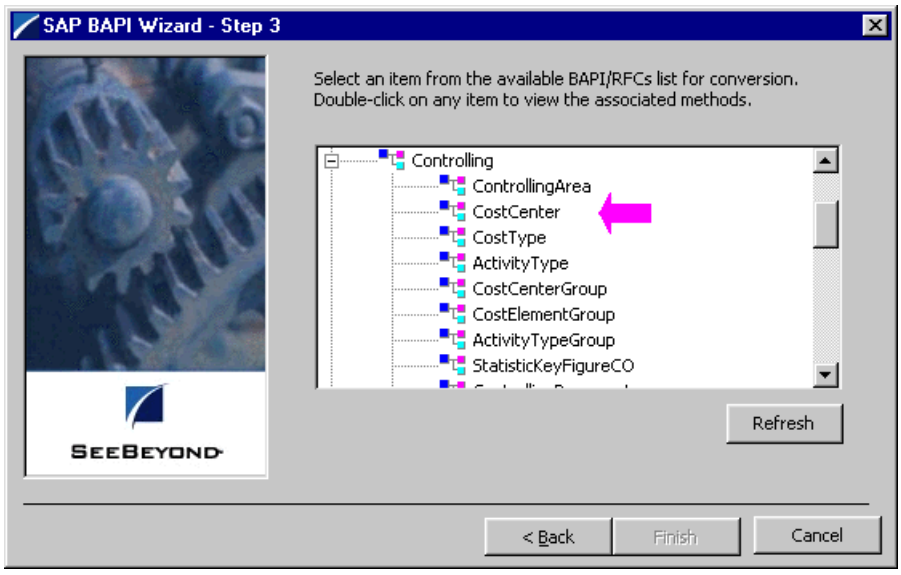


9 Specify the SAP R/3 logon parameters for your system:

- System ID
- Client Number
- User Name
- Password
- Application Server
- System Number
- Optional SAP Router String
- Language - Default: **EN** (English)
- RFC Trace - Default: **No**

10 Click **Next** to view the next Wizard dialog box.

Figure 16 SAP BAPI Wizard (3)



11 This Wizard dialog box lists all of the available BAPI and RFC Business Objects. Expand the tree as necessary and select the desired BAPI or RFC; for example, **Controlling/CostCenter**.

Note: *There is a limitation on the number of RFCs that can be displayed. Currently, we only display a maximum of 32767 RFCs in this list. An error message is generated when this number is exceeded. In addition, displaying more than 20,000 RFCs can result in diminished GUI performance, but this is only a design time constraint; runtime performance is not affected.*

Workaround

The following steps outline a workaround to overcome the diminished GUI performance described above.

A Create file in the eGate installation directory, <drive>:\egate\client\bin, that matches the login credentials required to connect to the SAP server. This file name will include credentials similar to the following example:

Language = EN
AppServer = sapdev5.stc.com
System Number = 00
System ID = C3X

Concatenate the filename credentials with "." (a period). Pre-append the filename with "RFCMAP" and post-append the file name with ".phx"

The syntax for this filename is:

RFCMAP.Language.AppServer.SysNumber.SysID.phx

B Using a text editor, add the names of the RFCs to this file. Do not include more than 32767 entries in this list. The first line in this file is the total entry count. The

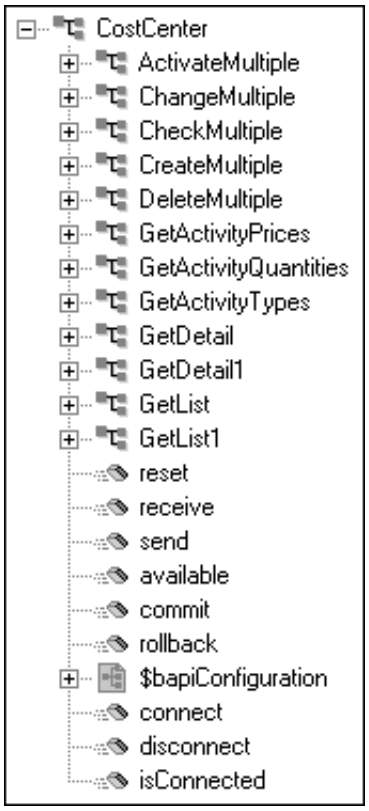
contents of the file would appear as follows. In this example, there are six RFCs to choose.

```
C:\eGate\client\bin>type RFCMAP.EN.sapdev2.00.C4X.phx
0000006
RFC>>/BKC/RFC_CHECK_TARGET_PERNR|/BKC/RFC_CHECK_TARGET_PERNR
RFC>>/BKC/RFC_CHECK_TARGET_SYSTEM|/BKC/RFC_CHECK_TARGET_SYSTEM
RFC>>/BKC/RFC_CLEAR_TARGET_AREA|/BKC/RFC_CLEAR_TARGET_AREA
RFC>>/BKC/RFC_GET_ASSOB_NUMBERS|/BKC/RFC_GET_ASSOB_NUMBERS
RFC>>/BKC/RFC_INSERT_ASSOB_DATA|/BKC/RFC_INSERT_ASSOB_DATA
RFC>>/BKC/RFC_INSERT_BAV_DATA|/BKC/RFC_INSERT_BAV_DATA
```

Reading the list from this file fixes the performance issue. It can be used in a Schema once the RFC is selected and a subsequent ETD is created from it.

12 Click **Finish** to view the ETD.

Figure 17 Event Type Definition - BAPI Example (CostCenter)



4.5.5 Using the IDoc Wizard

The Java ETD Editor contains an IDoc Wizard, which takes an IDoc and converts it to a **.xsc** file.

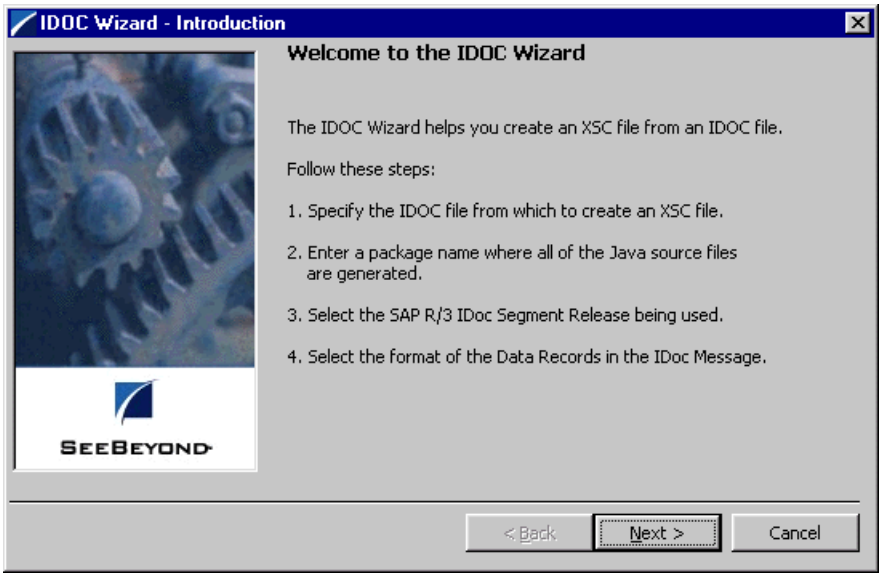
To create an Event Type Definition using the IDoc Wizard

1. Start the e*Gate Schema Designer, and open the schema for which you want to create an IDoc ETD.
2. From the **Options** menu, select **Default Editor** > **Java**.
3. Launch the ETD Editor.
4. Select **New** on the Java ETD Editor's toolbar. The New Event Type Definitions window appears, displaying all installed ETD Wizards.
5. Invoke the IDoc Wizard by selecting its icon (Figure 18).

Figure 18 IDoc Wizard Icon

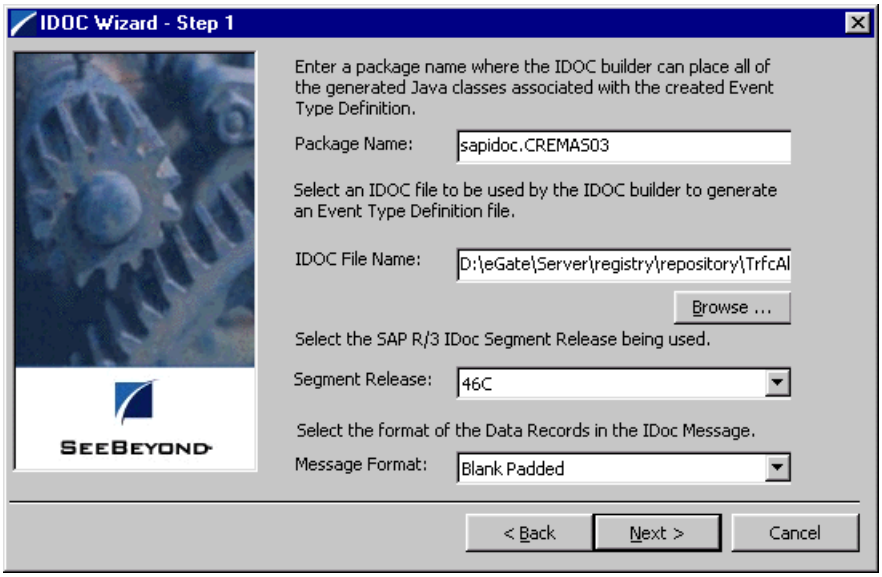
The initial Wizard dialog window now appears (see Figure 19).

Figure 19 IDoc Wizard - Introduction



6. Select **Next**.

Figure 20 IDoc Wizard - Step 1



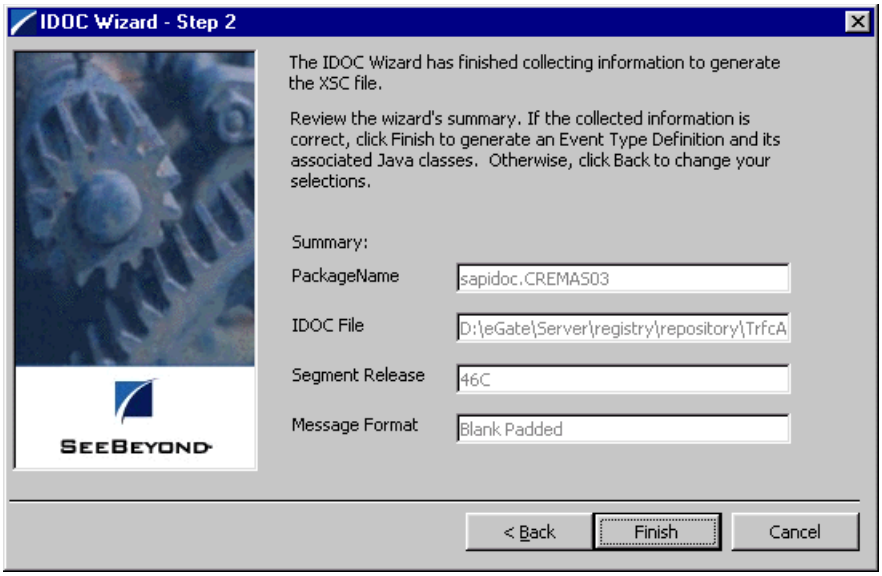
7 Enter a Package Name for the container in which the Wizard places the generated Java classes.

8 Enter an IDoc File Name for the IDoc description file to be used to create this IDoc.

9 Select the SAP segment release version to be used.

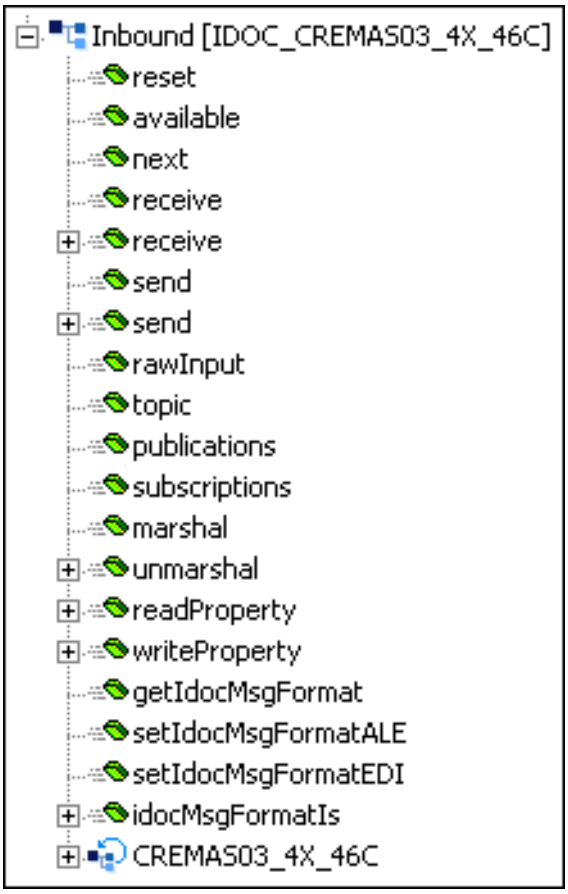
10 Click **Next** to view the next Wizard dialog box.

Figure 21 IDoc Wizard - Step 2



11 Click **Finish** to view the ETD.

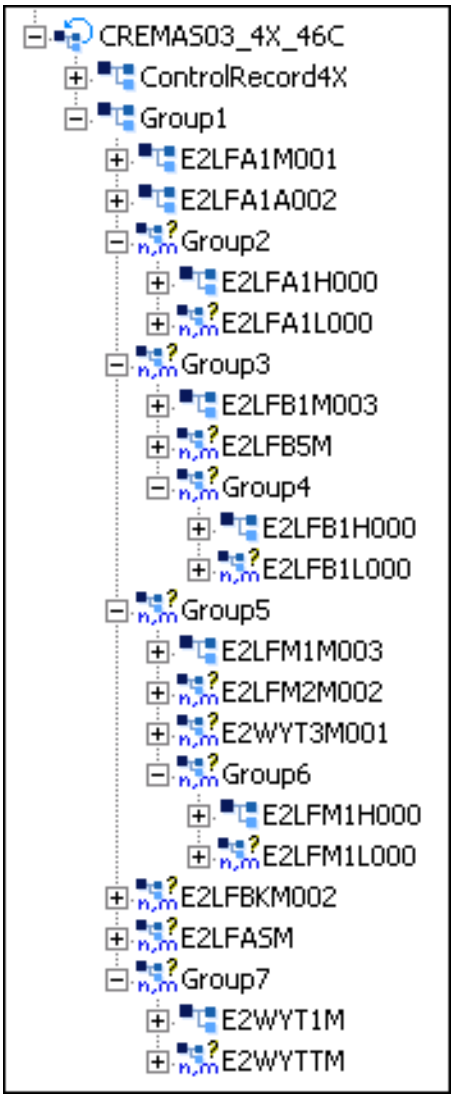
Figure 22 Event Type Definition - IDoc Client Example (CREMAS03)



The message format for the IDoc is stored in the resulting Java file, shown in this example as **IDOC_CREMAS03_4X_46C**, in the field **idocMsgFormat**. This field determines how data is marshaled or unmarshaled. If you want to change the type of data that is being processed in the collaboration, you can use one of the **set** methods; however, you must make sure these **set** methods are called *before* calling the **marshal** and **unmarshal** methods.

The repeating node, **CREMAS03_4X_46C**, contains a large number of sub-nodes (in several levels), each containing numerous IDoc fields. This structure is shown in Figure 23, not including the IDoc fields.

Figure 23 Repeating Node - CREMAS03_4X_46C



4.5.6 Using the Custom ETD Wizard

To prepare the Custom ETD Wizard to generate an ETD

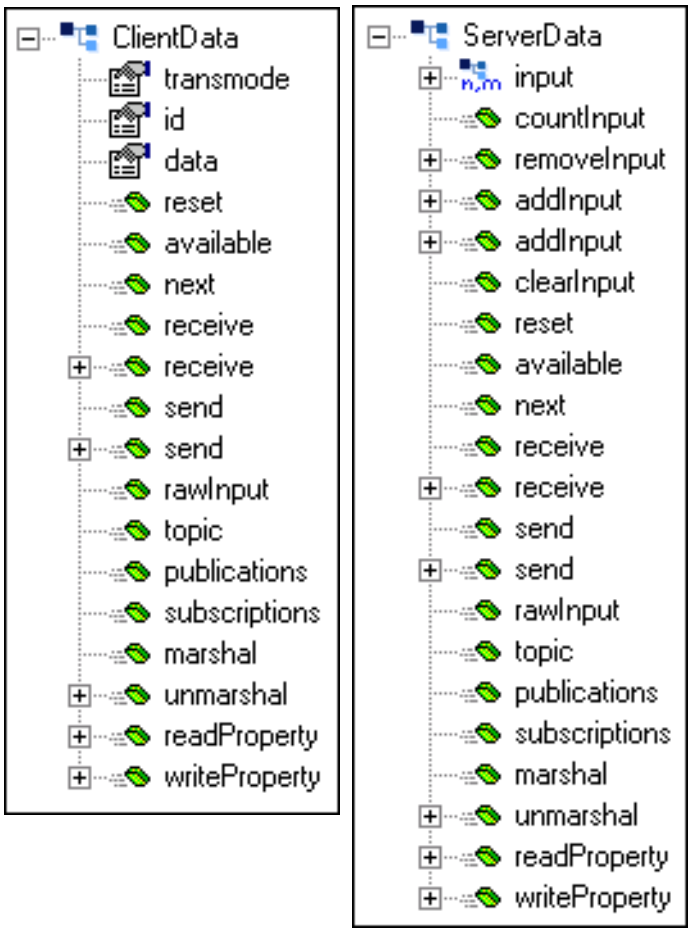
1. Start the e*Gate Schema Designer, and open the schema for which you want to create an IDoc ETD.
2. From the **Options** menu, select **Default Editor** > **Java**.
3. Launch the ETD Editor.
4. Select **New** on the Java ETD Editor's toolbar. The New Event Type Definitions window appears, displaying all installed ETD Wizards.
5. Invoke the Custom ETD Wizard by selecting its icon (see Figure 24).

Figure 24 Custom ETD Wizard Icon

6. When the Custom ETD Wizard dialog window opens, enter:
   - **Root Node Name**
   - **Package Name**
7. Click **Next** and **Finish** to close the Custom ETD Wizard.
8. In the Event Type Definition pane of the ETD Editor, create the required fields.
9. Rename the fields as appropriate (they are created as Field1, Field2, and so on).
10. From the **File** menu, select **Compile And Save**.

    If the file fails to compile, errors are displayed in the **Compile** message box in the Editor. If the file compiles without error, the title bar displays the name of the **.xsc** file and **(Sandbox)**.
11. To move the file from the *Sandbox* to the runtime environment, select **Promote to Run Time** from the Editor's **File** menu.

Figure 25 Custom ETD Examples - ClientData/ServerData



4.6 Assigning ETDs to Event Types

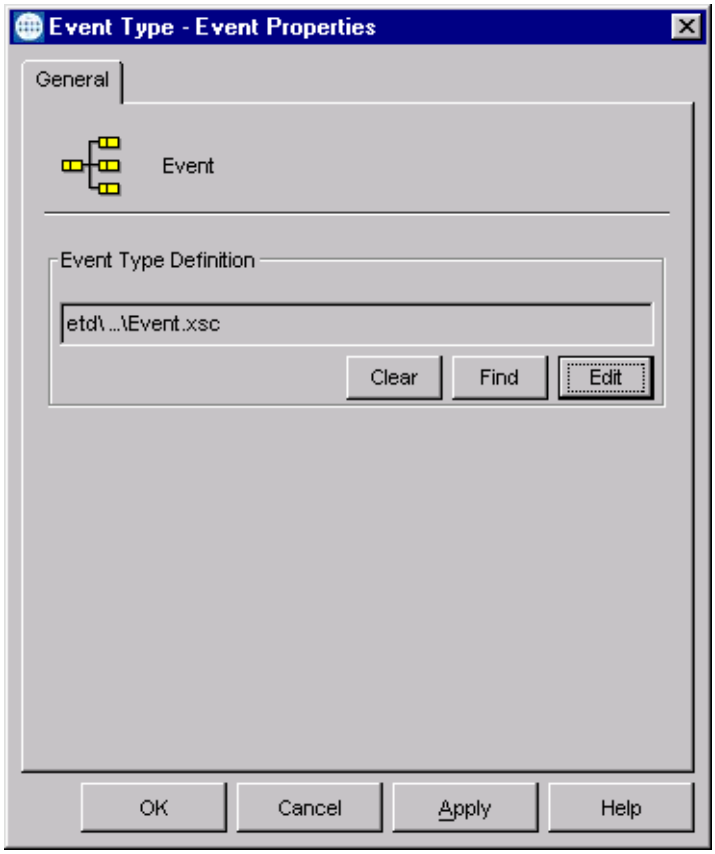
After you have created the e*Gate system's ETD files, you can assign them to Event Types you have already created.

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 26.

Figure 26 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.
5. Open the **etd** folder, then select the desired file name (**.xsc**). Note that there may be an intervening sub-folder.
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.

4.7 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 buffer for Events. They actively record information about the current *state* of Events.

To set up the IQ Manager for the BAPI e*Way

1 Open the IQ Manager Properties.

2 Select **SeeBeyond JMS** as the IQ Manager Type.

3 Select **Use Default Configuration**.

4 Click **Apply** and **OK** to apply the properties and close the dialog box.

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

4.8 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. Additional information regarding Collaborations can be found in **Collaborations** on page 179 and in the *e*Gate Integrator User's Guide*.

4.8.1 Creating Collaboration Rules

As shown in **Collaborations** on page 179, a Collaboration is governed by a Collaboration Rule, which specifies the source and destination ETDs to be used and contains the business rules that map information between the ETDs. The pass-through Collaboration Rule **crServerFeeder** from the sample schema **BapiJava** is used as an uncomplicated example in the following procedure.

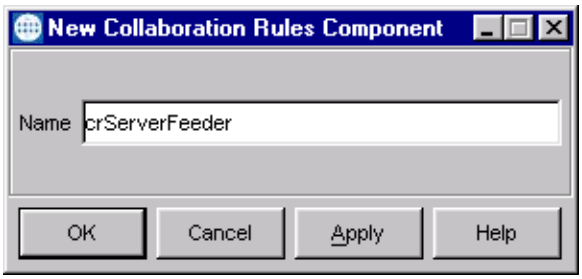
To create a new Collaboration Rule

1 Click the **Create New Collaboration Rules** button in the Schema Designer.

Figure 27 Create New Collaboration Rules Button

2 Type a name for the new Collaboration Rule into the text box on the **New Collaboration Rules Component** dialog box (see Figure 28).

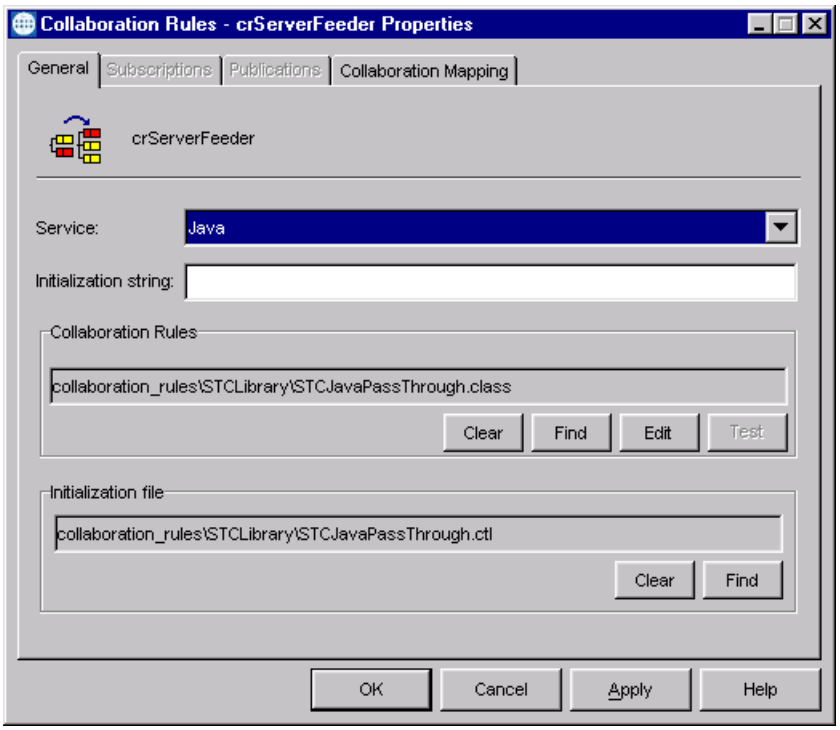
Figure 28 New Collaboration Rules Component Dialog Box



3 Click **OK** and the new Collaboration Rule appears in the Components view of the Schema Designer.

4 Right-click on the Rule and select **Properties** from the pop-up menu. The **Collaboration Rules Properties** dialog box appears, opened to the **General** tab (see Figure 29).

Figure 29 Collaboration Rules Properties - General Tab



5 Select the following parameters from the provided menu or file browser:

A Service (for example, **Java**)

B Initialization string (if required)

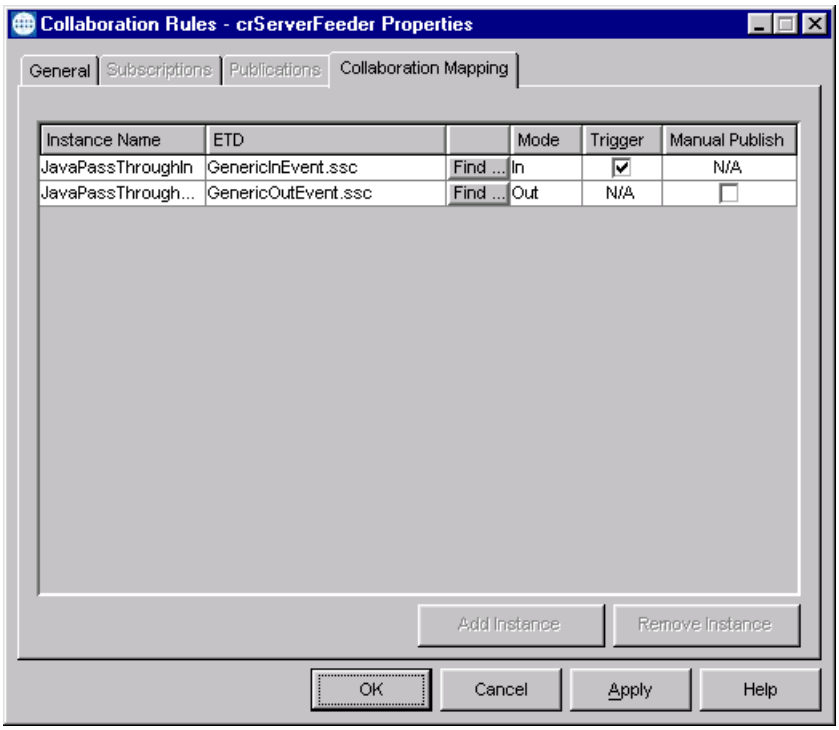
C Collaboration Rules (for example, **STCJavaPassThrough.class**)

D Initialization file (for example, **STCJavaPassThrough.ctl**)

6 Click the **Collaboration Mapping** tab to view its contents (see Figure 30).

7 Use the **Find** browser to select the source and destination ETDs. In this example, they are **GenericInEvent.ssc** and **GenericOutEvent.ssc**.

Figure 30 Collaboration Rules Properties - Collaboration Mapping Tab



8 Clicking the **Edit** button on the **General** tab opens the Collaboration Rules Editor, where you can define the business rules for the Collaboration.

4.8.2 Defining Business Rules

Collaboration Business Rules are defined using the 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**.

Note: *The information given in this section is necessarily brief; see the* ***e*Gate Integrator User's Guide*** *for detailed descriptions of the Java Collaboration Rules Editor and its use.*

General Procedure

1. Access Collaboration Rules Editor by clicking the **New/Edit** button under the **Collaboration Rules** field in the Collaboration Rules *Properties* dialog box.
2. When the Collaboration Rules Editor opens, maximize the window and expand the command nodes in the *Source Events* and *Destination Events* panes to display the available nodes and methods.
3. Each rule is created by one of the following actions:
   - Clicking the **rule** button on the *Business Rules* toolbar
   - Dragging an object from the *Source Events* pane and dropping it onto an object in the *Destination Events* pane
   - Dragging an object from the *Source Events* pane or *Destination Events* pane and dropping it into the **Rule** field of the *Rule Properties* pane

   Descriptions are added by typing the desired information into the **Description** field of the *Properties* pane.
4. When the Business Rules for your Collaboration Rule are complete, **Save** and **Compile**.

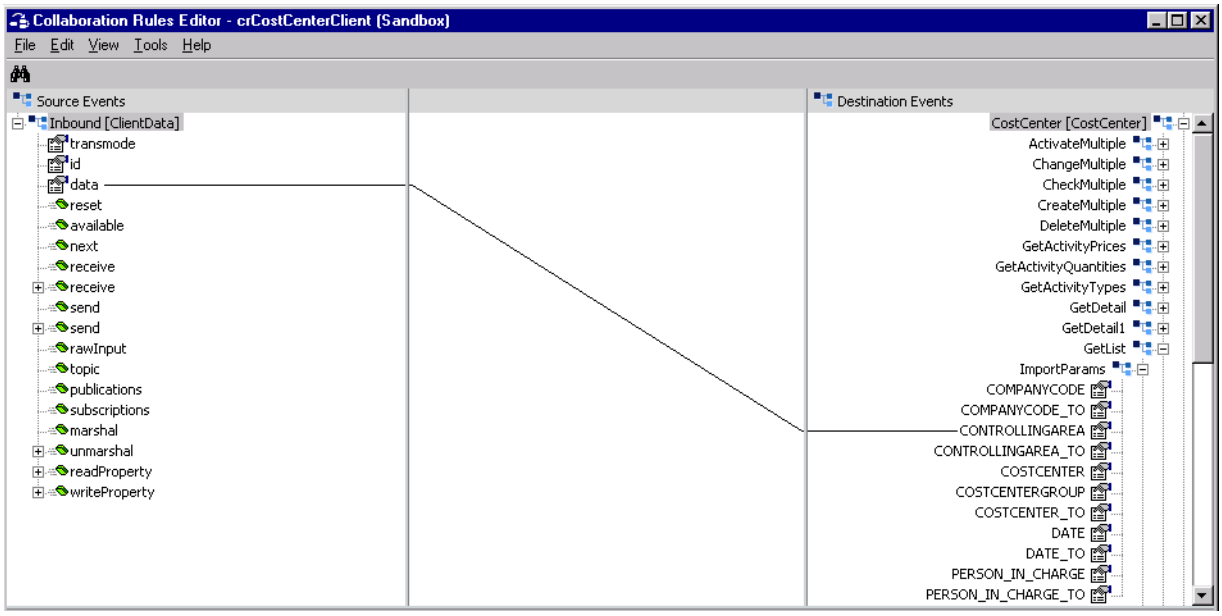
Note: *When data field conversion is required as part of the Collaboration Rule (in the RFC layer), you must perform it in the rule manually. You must remember to pad data to full field length with leading zeros, and format date and time fields correctly.*

Example: BAPI Client

In the following procedure, the Collaboration Rule **crCostCenterClient** from the sample schema **BapiJava** is used as a BAPI example. Additional information about the sample schema can be found in **BAPI Sample - Client Mode** on page 98, and detailed, comprehensive information on using the Collaboration Rules Editor can be found in the *e*Gate Integrator User's Guide*.

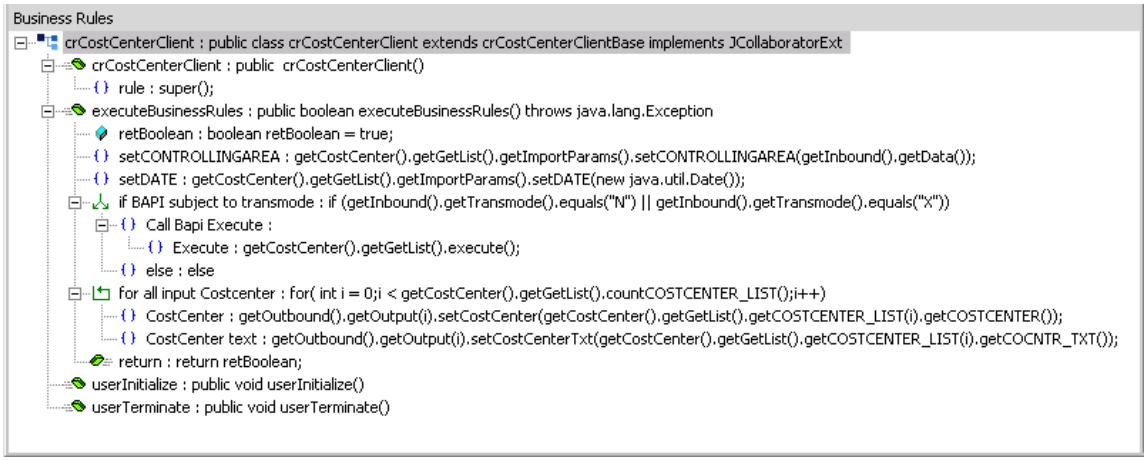
The mapping between the ETDs for **crCostCenterClient**, as displayed in the Collaboration Rules Editor, appear in Figure 31.

Figure 31 Collaboration Rule Map - crCostCenterClient



The Business Rules for **crCostCenterClient**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 32.

Figure 32 Business Rules - crCostCenterClient



The procedure that was followed in defining the business rules contained in the **crCostCenterClient** Collaboration Rule is given below.

1 The rule **Copy data to CONTROLLINGAREA** was created by dragging:

Source Events/Inbound/data

and dropping onto:

Destination Events/CostCenter/GetList/ImportParams/**CONTROLLINGAREA**

2 The rule **set STARTTIME to current time** was created by dragging:

Destination Events/CostCenter/GetList/ImportParams/**DATE**

and dropping into the **Rule** field of the *Rule Properties* pane to create the following code:

```
getCostCenter().getGetList().getImportParams().setDATE()
```

The argument **new java.Date()** was then typed into the **setDATE()** method's argument field, resulting in the following code:

```
getCostCenter().getGetList().getImportParams().setDATE(new
    java.util.Date())
```

3 The **if** expression, **call BAPI subject to Transmode**, was created by clicking the **if** button on the *Business Rules* toolbar. Then:

A The field:

Source Events/Inbound/**transmode**

was dragged and dropped into the **Rule** field of the *Rule Properties* pane to create the following code:

```
if (getInbound().getTransmode())
```

B The expression **.equals("N") ||** was typed in to create the following:

```
if (getInbound().getTransmode().equals(''N'') ||
```

C Again, the field:

Source Events/Inbound/**transmode**

was dragged and dropped into the **Rule** field of the *Rule Properties* pane to create the following code:

```
if (getInbound().getTransmode().equals(''N'') ||
      (getInbound().getTransmode())
```

D Finally, the expression **.equals("X")** was typed in to create the following:

```
getInbound().getTransmode().equals("N") ||
      getInbound().getTransmode().equals("X")
```

4 The associated **then** expression was selected and the command **Call BAPI Execute** was typed into the **Description** field of the *Then Properties* pane.

5 A new rule was then added under the **then** expression. This rule was populated by dragging the method:

Destination Events/CostCenter/GetList/**execute()**

was dragged and dropped into the **Rule** field of the *Rule Properties* pane to create the following code:

```
getCostCenter().getGetList().execute()
```

6 The **for** loop was created by selecting the **if** expression **call BAPI subject to Transmode**, and clicking the **for** button. The following values were entered in the *For Properties* pane:

- Counter Initialization: **int i = 0**
- Condition: **i <**
- Counter Update: **i++**

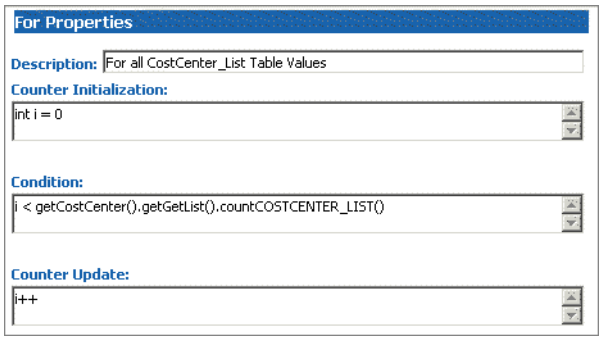
The method:

Destination Events/Cost Center/GetList/**countCOSTCENTER_LIST()**

was then dragged and dropped into the **Condition** field of the *Rule Properties* pane to create the following code:

```
i < getCostCenter().getGetList().countCOSTCENTER_LIST()
```

The values appear in the *For Properties* pane as follows:



7 The rule **print CostCenter_List Values** is a trace statement created by typing the following in the **Rule Properties/Rule** field:

```
EGate.ewayTrace(" COCNTR_TEXT  - " +
    getCostCenter().getGetList().getCOSTCENTER_LIST(i).getCOCNTR_
    TXT())
```

8 The next rule, **get Costcenter**, was created by:

A The field:

Destination Events/outbound [BapiResponse]/BapiResponseData/**CostCenter**

was dragged and dropped into the **Rule** field of the *Rule Properties* pane.

B In the **Select Repetition Instance** dialog box, the expression **i** was typed in as the value for **BapiResponseData**.

C Clicking **OK** created the following code:

```
getoutbound().getBapiResponseData(i).setCostCenter()
```

D The node

Destination Events/CostCenter/GetList/COSTCENTER_LIST/**COSTCENTER**

was dragged and dropped into the **setCostCenter()** method's argument field.

E In the **Select Repetition Instance** dialog box, the expression **i** was typed in as the value for **COSTCENTER_LIST**.

F Clicking **OK** created the following code:

```
getoutbound().getBapiResponseData(i).setCostCenter(getCostCente
      r().getGetList().getCOSTCENTER_LIST(i).getCOSTCENTER())
```

9 The last rule, **get CostCenter Text**, was created by:

A The field:

Destination Events/outbound [BapiResponse]/BapiResponseData/ CostCenterTxt

was dragged and dropped into the **Rule** field of the *Rule Properties* pane.

B In the **Select Repetition Instance** dialog box, the expression **i** was typed in as the value for **BapiResponseData** to create the following code:

```
getoutbound().getBapiResponseData(i).setCostCenterTxt()
```

C The field:

Destination Events/CostCenter/GetList/COSTCENTER_LIST/**COCNTR_TXT**

was dragged and dropped into the **setCostCenterTxt()** method's argument field.

D In the **Select Repetition Instance** dialog box, the expression **i** was typed in as the value for **COSTCENTER_LIST**.

E Clicking **OK** created the following code:

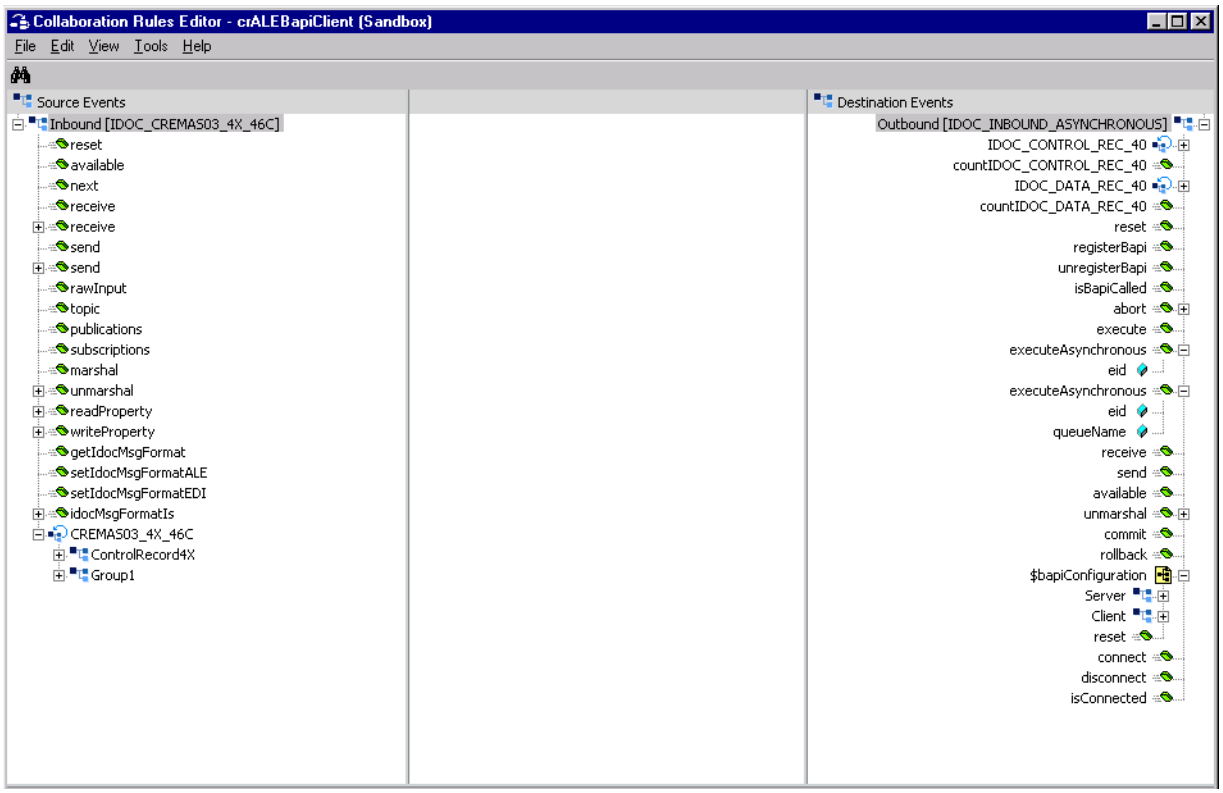
```
getoutbound().getBapiResponseData(i).setCostCenterTxt(getCostCe
      nter().getGetList().getCOSTCENTER_LIST(i).getCOCNTR_TXT()
      )
```

Example: IDoc Client

In the following procedure, the Collaboration Rule **crALEBapiClient** from the sample schema **TrfcAleBapiJava** is used as an IDoc example. Additional information about the sample schema can be found in **IDoc Sample - Client Mode** on page 123, and detailed, comprehensive information on using the Collaboration Rules Editor can be found in the *e*Gate Integrator User's Guide*.

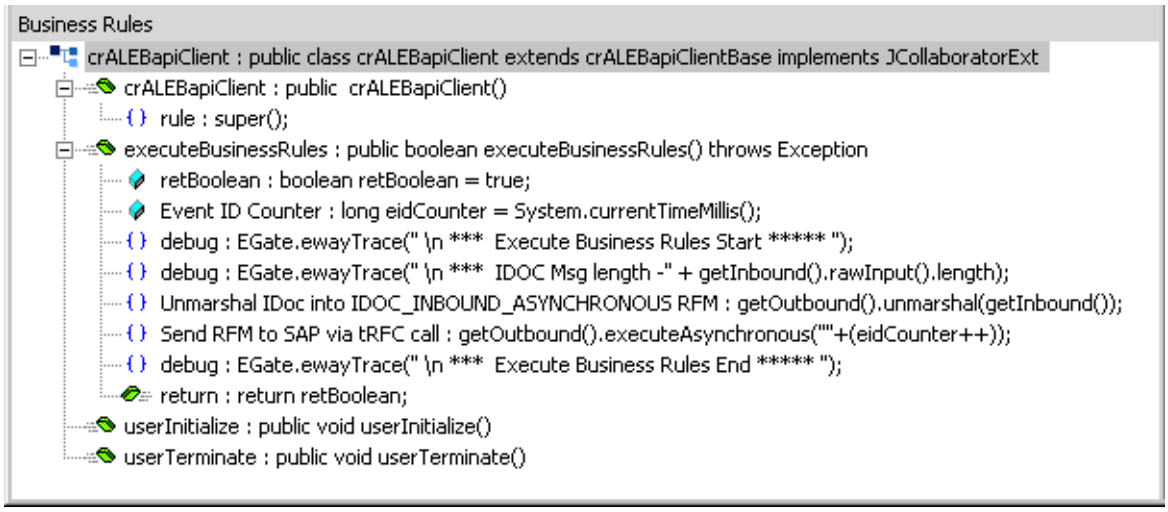
The ETDs for **crALEBapiClient**, as displayed in the Collaboration Rules Editor, appear in Figure 33.

Figure 33 Collaboration Rule Map - crALEBapiClient



The Business Rules for **crServerFeeder**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 34.

Figure 34 Business Rules - crALEBapiClient



The **crALEBapiClient** Collaboration Rules were created as follows:

1 The variable **Event ID Counter** was created by:

A Clicking on the **var** button to insert a variable.

B Typing the following values into the *Variable Properties* window:

- Description: **Event ID Counter**
- Name: **eidCounter**
- Type: **long**
- Array: (not selected)
- Initial Value: **System.currentTimeMillis()**
- Access Modifiers: **none**

2 The first **debug** rule is the first part of a trace statement created by typing the following in the **Rule** field of the *Rule Properties* pane:

```
EGate.ewayTrace(" \n ***  Execute Business Rules Start ***** ")
```

3 The second **debug** rule was created by typing the following in the **Rule** field of the *Rule Properties* pane:

```
EGate.ewayTrace(" \n *** IDOC Msg length -" +
    getInbound().rawInput().length)
```

4 The rule **Unmarshal IDoc into IDOC_INBOUND_ASYNCHRONOUS RFM** was created by:

A Dragging the method

Destination Events/Outbound [IDOC_INBOUND_ASYNCHRONOUS]/ **unmarshal**

into the **Rule** field of the *Rule Properties* pane.

B Entering **getInbound()** as the IDoc value in the **unmarshal()** *Parameters* dialog box, producing the following code:

```
getOutbound().unmarshal(getInbound())
```

5 The rule **Send RFM to SAP via tRFC call** was created by:

A Dragging the method

Destination Events/Outbound /**executeAsynchronous**

into the **Rule** field of the *Rule Properties* pane.

B The **executeAsynchronous** Parameters dialog box opens. Enter ""+(eidCounter++) as the eid value. This produces the following code:

C Entering **""+(eidCounter++)** as the eid value in the **executeAsynchronous** *Parameters* dialog box, producing the following code:

```
getOutbound().executeAsynchronous(""+(eidCounter++))
```

6 The last **debug** rule is created by typing the following into the **Rule** field of the *Rule Properties* pane:

```
EGate.ewayTrace(" \n *** Execute Business Rules End ***** ")
```

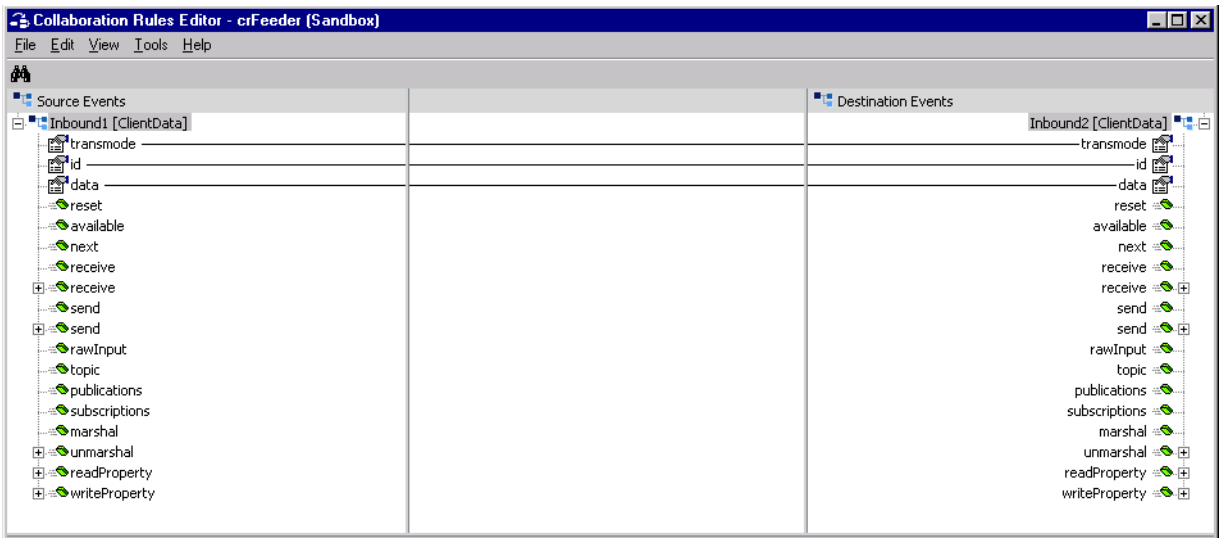
Example: Pass-Through

In the following procedure, the Collaboration Rule **crFeeder** from the sample schema **BapiJava** is used as a typical pass-through example for situations in which the Java Pass-Through Collaboration Service, with its preset Business Rules, is inappropriate.

Additional information about the sample schema can be found in **BAPI Sample - Client Mode** on page 98, and detailed, comprehensive information on using the Collaboration Rules Editor can be found in the *e*Gate Integrator User's Guide*.

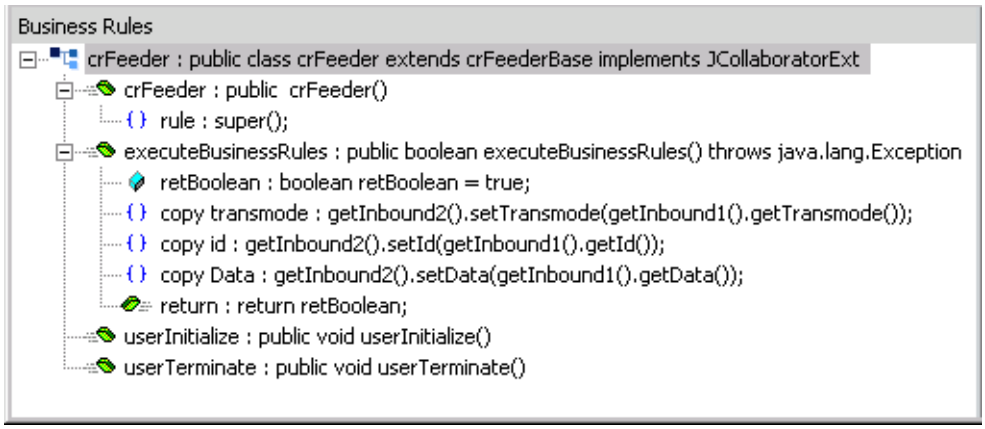
The mapping between the ETDs for **crFeeder**, as displayed in the Collaboration Rules Editor, appear in Figure 35.

Figure 35 Collaboration Rule Map - crFeeder



The Business Rules for **crFeeder**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 36.

Figure 36 Business Rules - crFeeder



The following procedure is used to create the **crFeeder** Collaboration Business Rules.

1 The field:

Source Events/Inbound [Clientdata]/**transmode**

was dragged and dropped onto the field:

Destination Events/Outbound [Clientdata]/**transmode**

A line appears on the mapping plane to indicate a node relationship. This creates the following code, as displayed in the Rule Properties, **Rule** field:

```
getOutbound().setTransmode(getInbound().getTransmode())
```

2 The field:

Source Events/Inbound [Clientdata]/**id**

was dragged and dropped onto the field:

Destination Events/Outbound [Clientdata]/**id**

3 The field:

Source Events/Inbound [Clientdata]/**data**

was dragged and dropped onto the field:

Destination Events/Outbound [Clientdata]/**data**

4 Clicking the **rule** button inserts a rule after the **Copy id to id** rule.

5 The method:

Destination Events/Outbound [Clientdata]/**send()**

was dragged and dropped onto the **Rule** field of the *Rule Properties* pane (note that this was the first **send()** method on the command node).

4.8.3 Adding Third-party JAR Files to the Collaboration Classpath

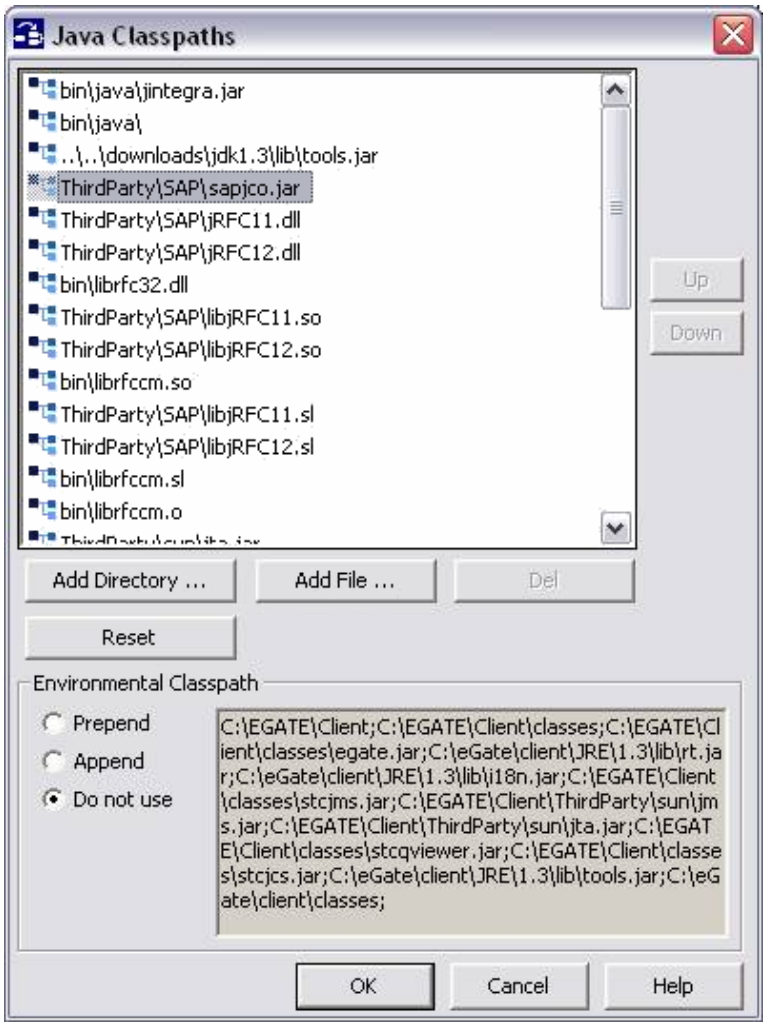
When using SAP Java Connector Version 2.1.5 with a Java Collaboration, you must add the SAP JAR file, **sapjco.jar**, to your Collaboration Classpath.

To add **sapjco.jar** to the Collaboration Classpath do the following:

1 Open the Collaboration in the Collaboration Rules Editor. From the Collaboration Rules Editor toolbar, select **Tools**, and select **Options**. The Java Classpaths dialog box appears.

2 Click Add File, and select the **sapjco.jar** (see Figure 37).

Figure 37 Adding files to the Collaboration Classpath



3 Once you have added the JAR file, click OK to close the Java Classpath dialog box. Compile and promote the Collaboration.

For more information on required DLL and JAR files for JCo 2.1.5, see **"Installing SAP Java Connector Version 2.1.5" on page 25**.

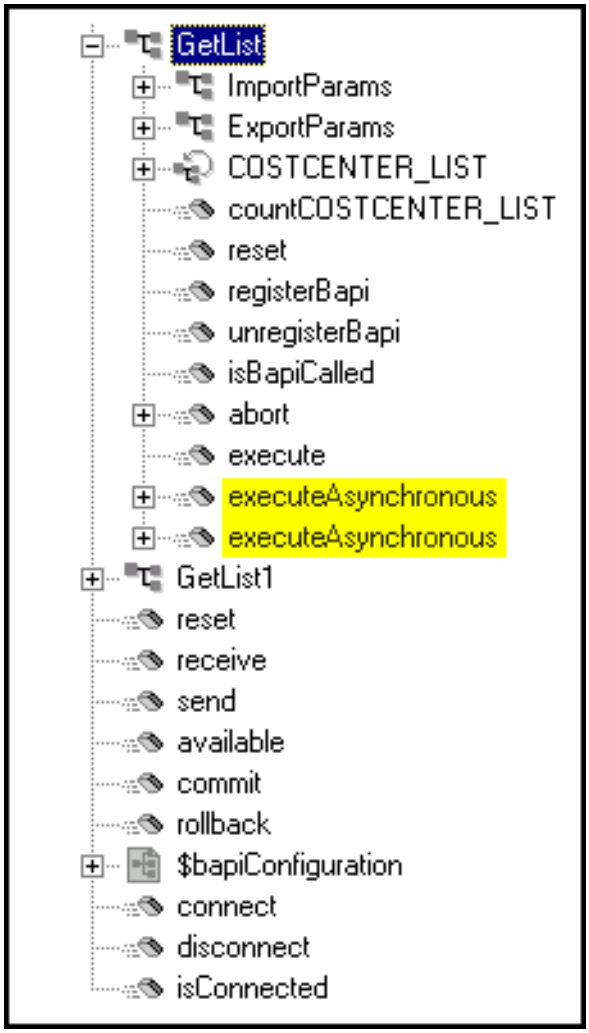
4.9 Establishing Connections from an ETD

Transactional RFC (tRFC) connections can be set up using the ETD Editor as described in this section.

4.9.1 Client Mode

Use the BAPI ETD (see **Figure 17 on page 58**) to set up the tRFC connection for the BAPI/RFC Client. An expanded view of the relevant nodes is shown in Figure 38.

Figure 38 tRFC Connections (Client)



To set up tRFC for Client mode operation

1. Populate the appropriate BAPI/RFC **Import** and/or **Table** parameter nodes with data from an Inbound Event.
2. Call the BAPI ETD's **executeAsynchronous()** method, supplying it with the Event ID (EID) of the Inbound Event.

Note: *Only messages received from JMS e*Way Connections give access to such IDs, by calling the Inbound ETD's* **readProperty(JMSMessageID)** *method.*

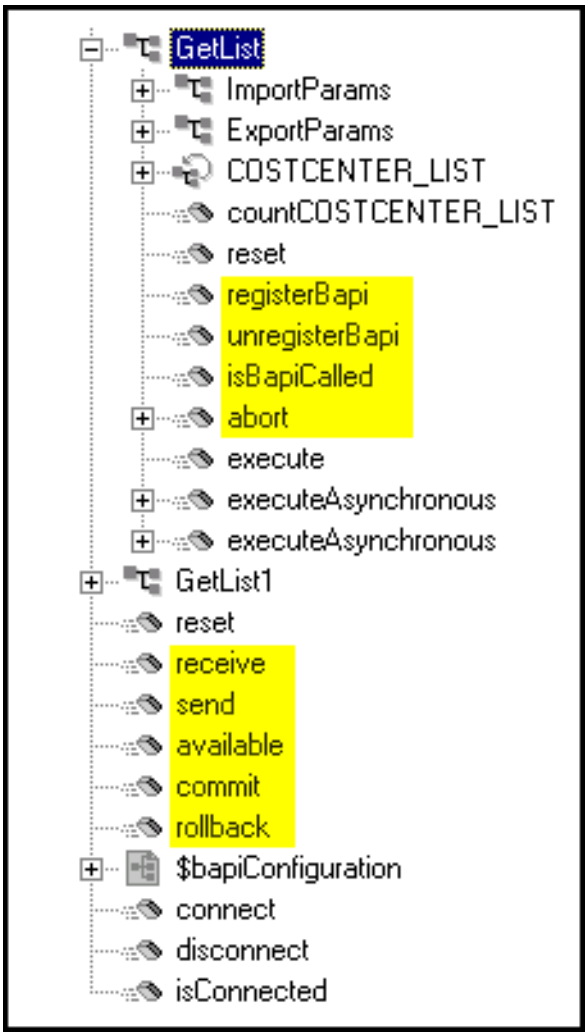
3 Return successfully from **executeBusinessRules()** method.

When the Collaboration has successfully committed all read messages from the JMS Queue, the BAPI ETD's **commit()** method will be called automatically. This, in turn, calls the **confirmTID()** method of the SAP TID Manager which informs the SAP system that the processed TIDs will not be reused.

4.9.2 Server Mode

Use the BAPI ETD (see **Figure 17 on page 58**) to set up the tRFC connection for the BAPI/RFC Server. An expanded view of the relevant nodes is shown in Figure 39.

Figure 39 tRFC Connections (Server)



To set up either Automatic or On-Demand Connection Establishment Modes

1. Register BAPI/RFC as a server.

   In **userInitialize()**, call the **registerBapi()** method for the appropriate BAPI/RFC from the BAPI ETD. This instructs the BAPI connector to start listening for calls to the respective BAPI/RFC.

2. Poll for BAPI/RFC calls.

   In **executeBusinessRules()**, call the **receive()** method from the BAPI ETD as the first rule.

3. Determine if any calls are made.

   Call the **available()** method from the BAPI ETD from an **if** rule as the condition.

4. Test if specific BAPI/RFC is called.

   In the **then** section, call the **isBapiCalled()** method for the appropriate BAPI/RFC from an **if** rule as the condition.

5. Deploy business logic.

   In this **then** section, drag and drop to or from the data nodes (**Import**, **Export** and **Table** parameters) for the appropriate BAPI/RFC and send data to e*Gate via a message-based ETD, if necessary.

6. Acknowledge the SAP caller.

   A. Positive acknowledgment, Internal Transaction mode:

      If no data response to the caller is required, simply return successfully out of **executeBusinessRules()** method and when the messages sent to e*Gate has been successfully posted, the SAP caller will automatically be acknowledged.

   B. Positive acknowledgment, tRFC mode (or if response required):

      Before returning out of **executeBusinessRules()** method, call the **send()** method for the appropriate BAPI/RFC to send the populated **Export** and/or **Table** parameters back to the SAP caller.

   C. Negative acknowledgment:

      When data received cannot yet be processed, the only way to apprise the SAP caller is by sending a explanation message before aborting the connection. To do so, call the **abort()** method in the BAPI ETD.

To set up Manual Connection Establishment Mode

1. Establish the server connection by calling the **connect()** method for the BAPI ETD (you can first verify the connection by calling the **isConnected()** method).
2. Register BAPI/RFC as a server.

   In **executeBusinessRules()**, call the **registerBapi()** method for the appropriate BAPI/RFC from the BAPI ETD. This instructs the BAPI connector to start listening for calls to the respective BAPI/RFC. The Connection profile for the Server is obtained dynamically from a message-based ETD.
3. Poll for BAPI/RFC calls.

   In **executeBusinessRules()**, call the **receive()** method from the BAPI ETD as the first rule.
4. Determine if any calls are made.

   Call the **available()** method from the BAPI ETD from an **if** rule as the condition.
5. Test if specific BAPI/RFC is called.

   In the **then** section, call the **isBapiCalled()** method for the appropriate BAPI/RFC from an **if** rule as the condition.
6. Deploy business logic.

   In this **then** section, drag and drop to or from the data nodes (**Import**, **Export** and **Table** parameters) for the appropriate BAPI/RFC and send data to e*Gate via a message-based ETD, if necessary.
7. Acknowledge the SAP caller.

   A. Positive acknowledgment, Internal Transaction mode:

      If no data response to the caller is required, simply return successfully out of **executeBusinessRules()** method and when the messages sent to e*Gate has been successfully posted, the SAP caller will automatically be acknowledged.

   B. Positive acknowledgment, tRFC mode (or if response required):

      Before returning out of **executeBusinessRules()** method, call the **send()** method for the appropriate BAPI/RFC to send the populated **Export** and/or **Table** parameters back to the SAP caller.

   C. Negative acknowledgment:

      When data received cannot yet be processed, the only way to apprise the SAP caller is by sending a explanation message before aborting the connection. To do so, call the **abort()** method in the BAPI ETD.

Note: *In Internal Transaction mode, the BAPI ETD's* **disconnect()** *method cannot be called until the SAP caller has been automatically acknowledged by means of the* **commit()** *method.*

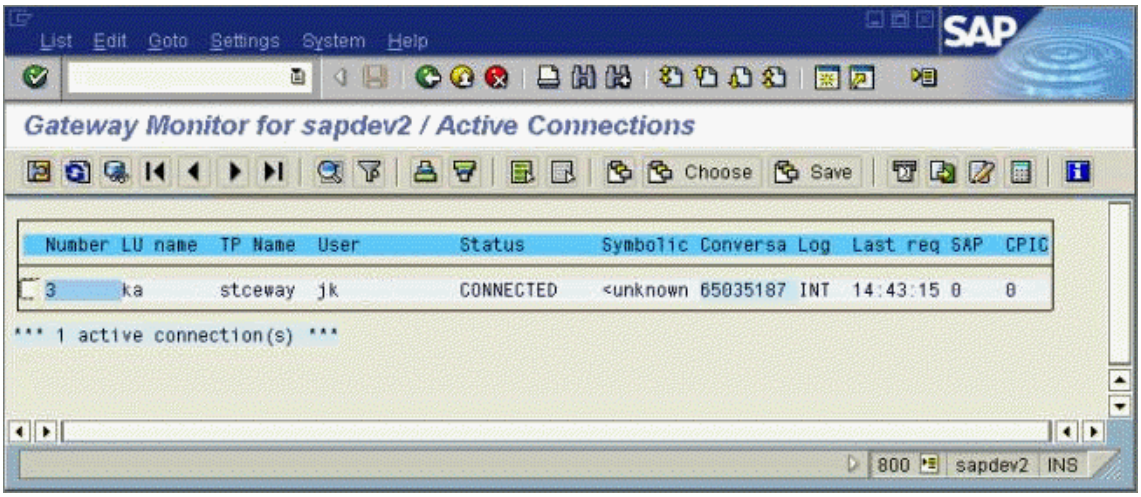
4.10 SAP R/3 Procedures

4.10.1 Connections

Viewing Connection Status

The status of an RFC connection can be viewed in SAP transaction **SMGW**, *Gateway Monitor*. A typical active connection is shown in Figure 40.

Figure 40 Gateway Monitor - Active Connections

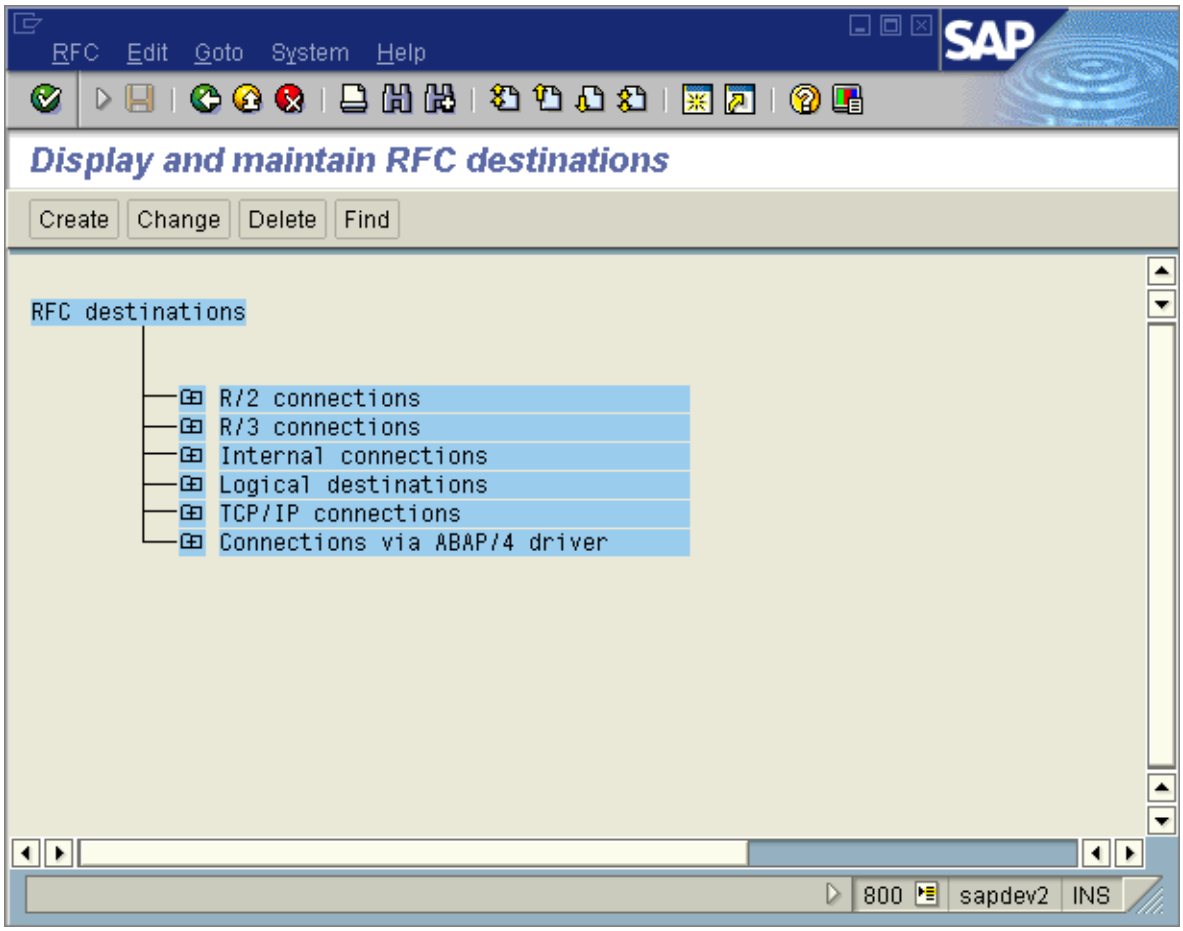


Testing an RFC Connection

To test an RFC connection, go to SAP transaction **SM59**, *Display and Maintain RFC Destinations*, and perform the following procedure. This example uses the R&D Test connection, ZSBYNRADTEST, to demonstrate the procedure.

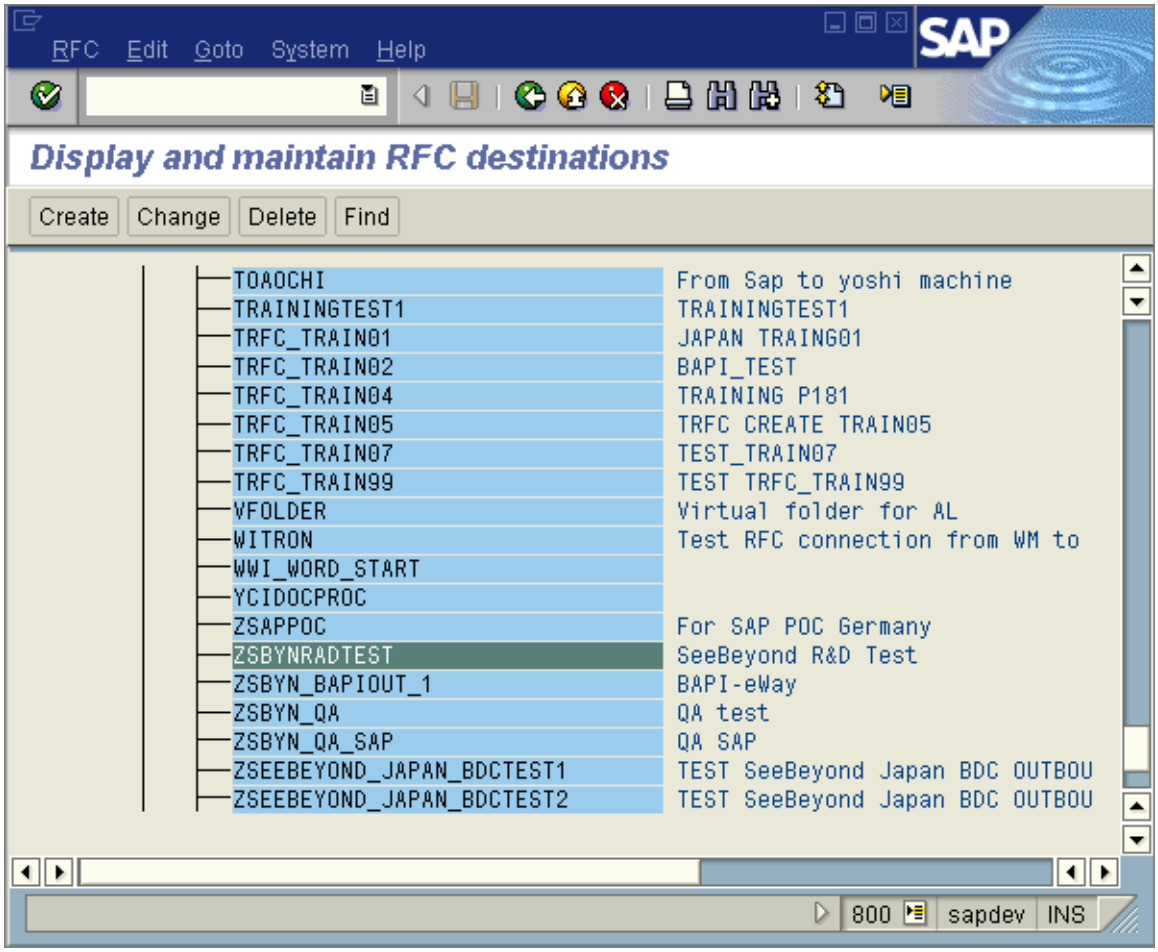
1 From the SAP GUI, select **TCP/IP Connections** (see Figure 41).

Figure 41 SAP - Display and Maintain RFCD Destinations



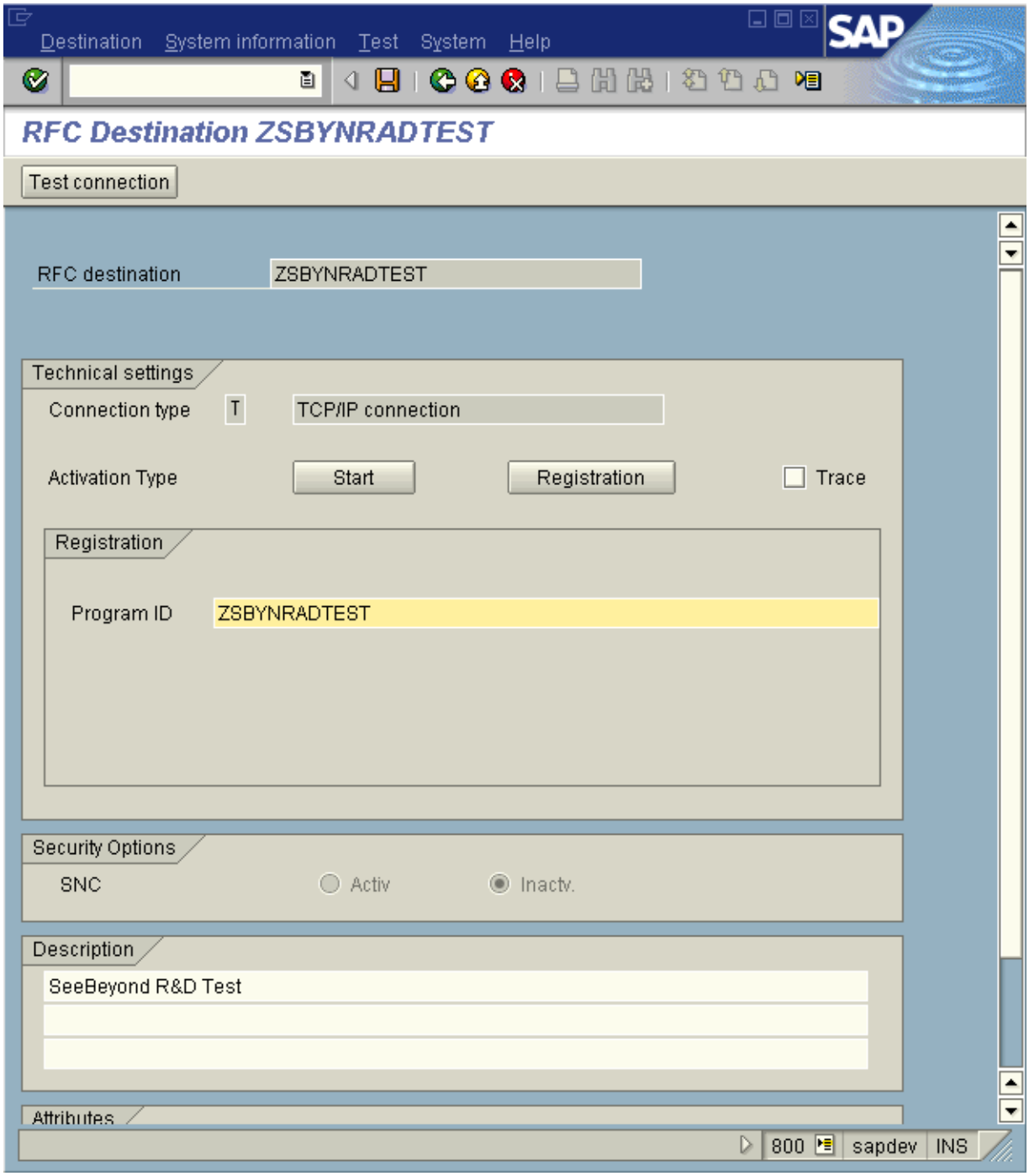
2 Select the specific connection (see Figure 42).

Figure 42 SAP - Display and Maintain RFCD Destinations - Connections



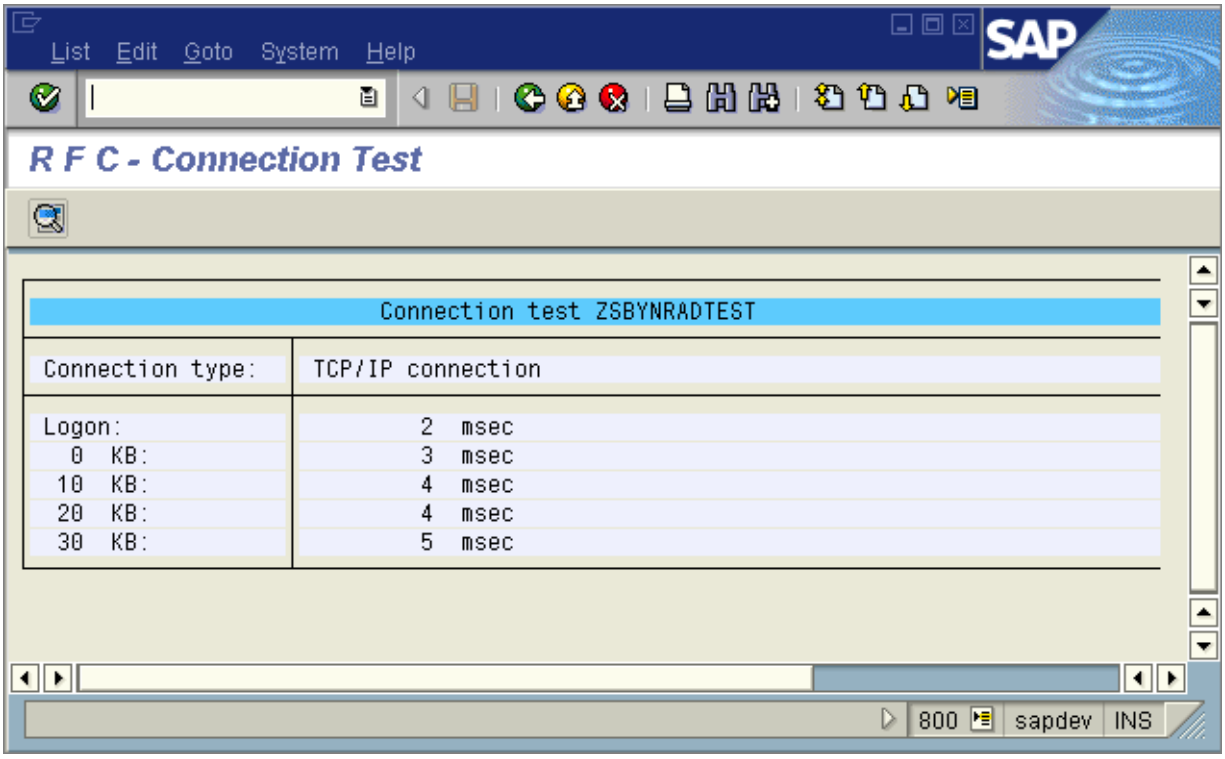
3 Select **Test Connection** (see Figure 43).

Figure 43 RFC Destination ZSBYNRADTEST - Test Connections



4 If the connection is working, you will receive a report similar to that shown in Figure 44.

Figure 44 tRFC Connection Test



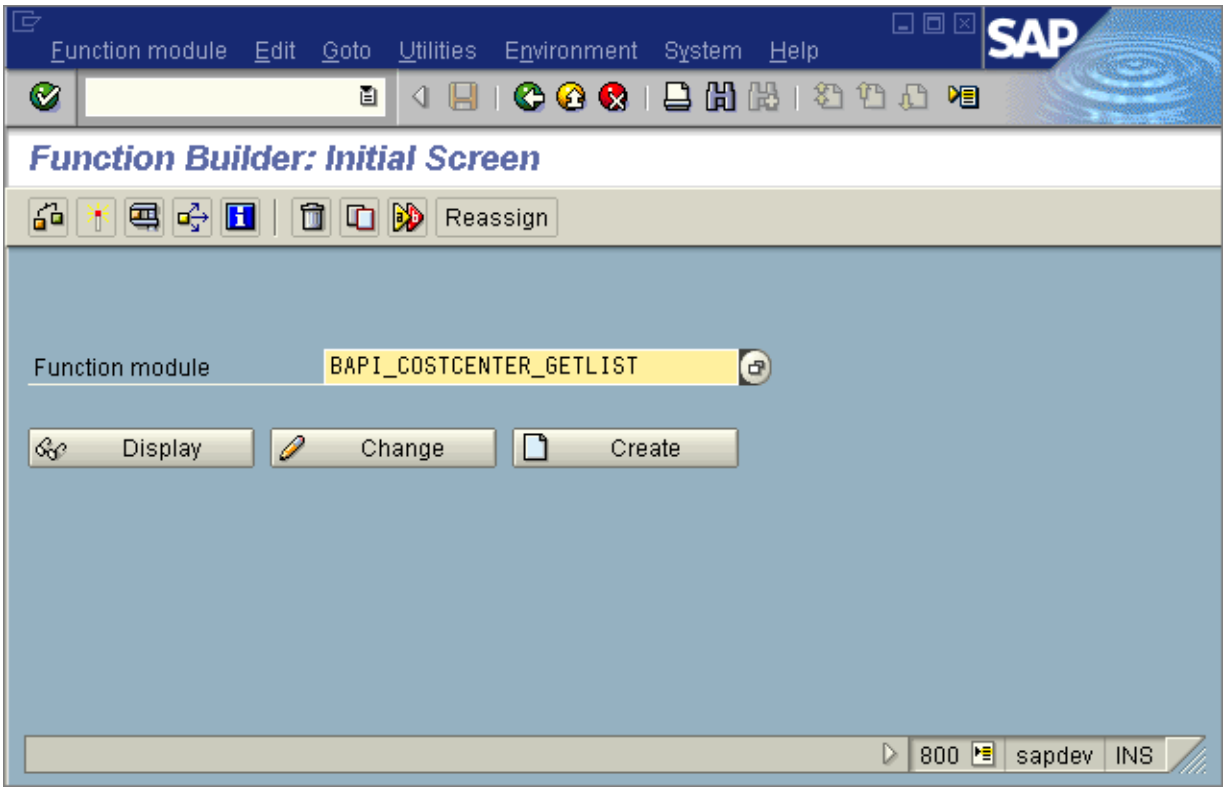
4.10.2 Exporting Data from SAP R/3

Sending a Request

To send a Request from SAP R/3 to a SAP BAPI e*Way, perform the following procedure:

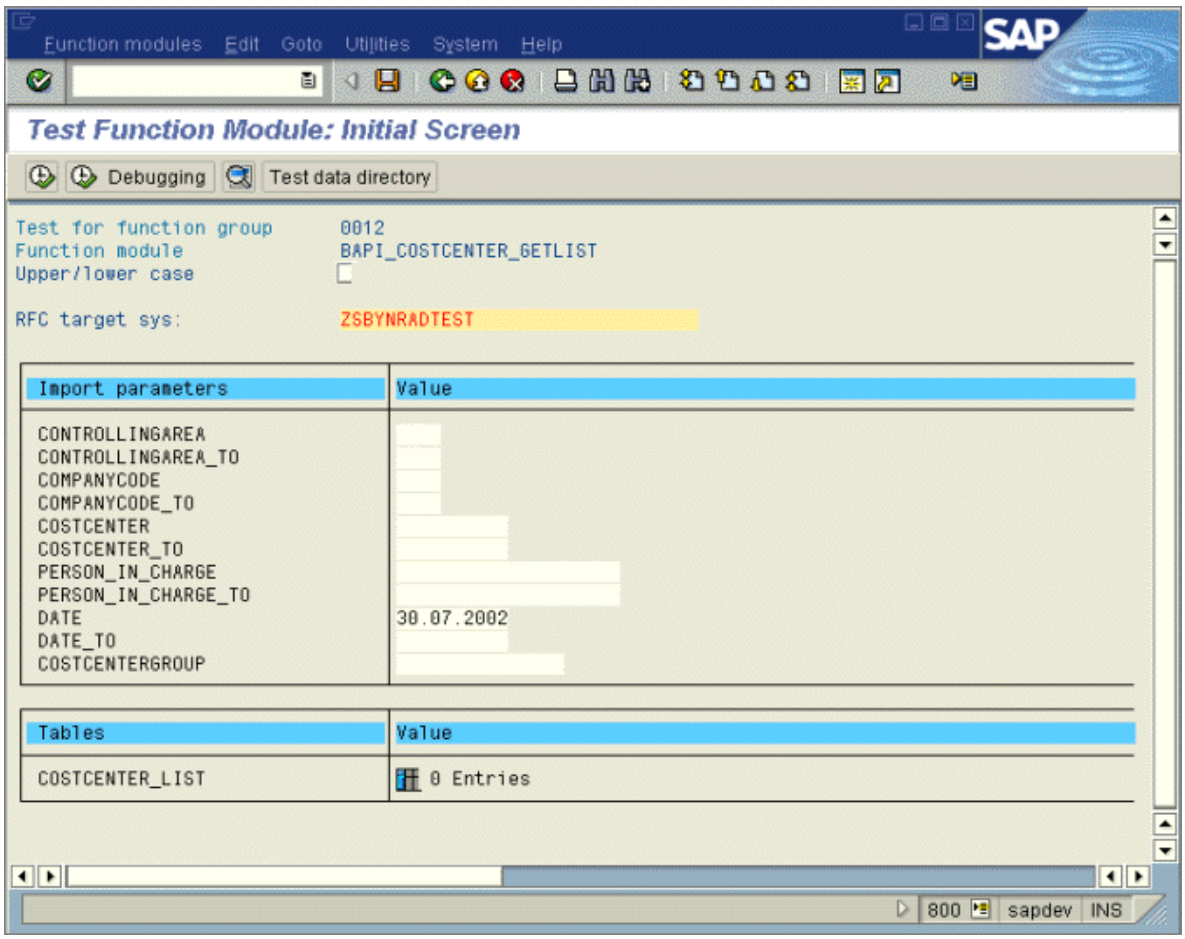
1 In SAP transaction **SE37**, *Function Builder*, select the desired RFM (see Figure 45). This example uses the RFM BAPI_COSTCENTER_GETLIST, which is used in the BapiJava sample schema in Server mode.

Figure 45 SAP - Function Builder (Initial)



2 Perform a single test of this RFM by *Executing* this module for your program ID (see Figure 46).

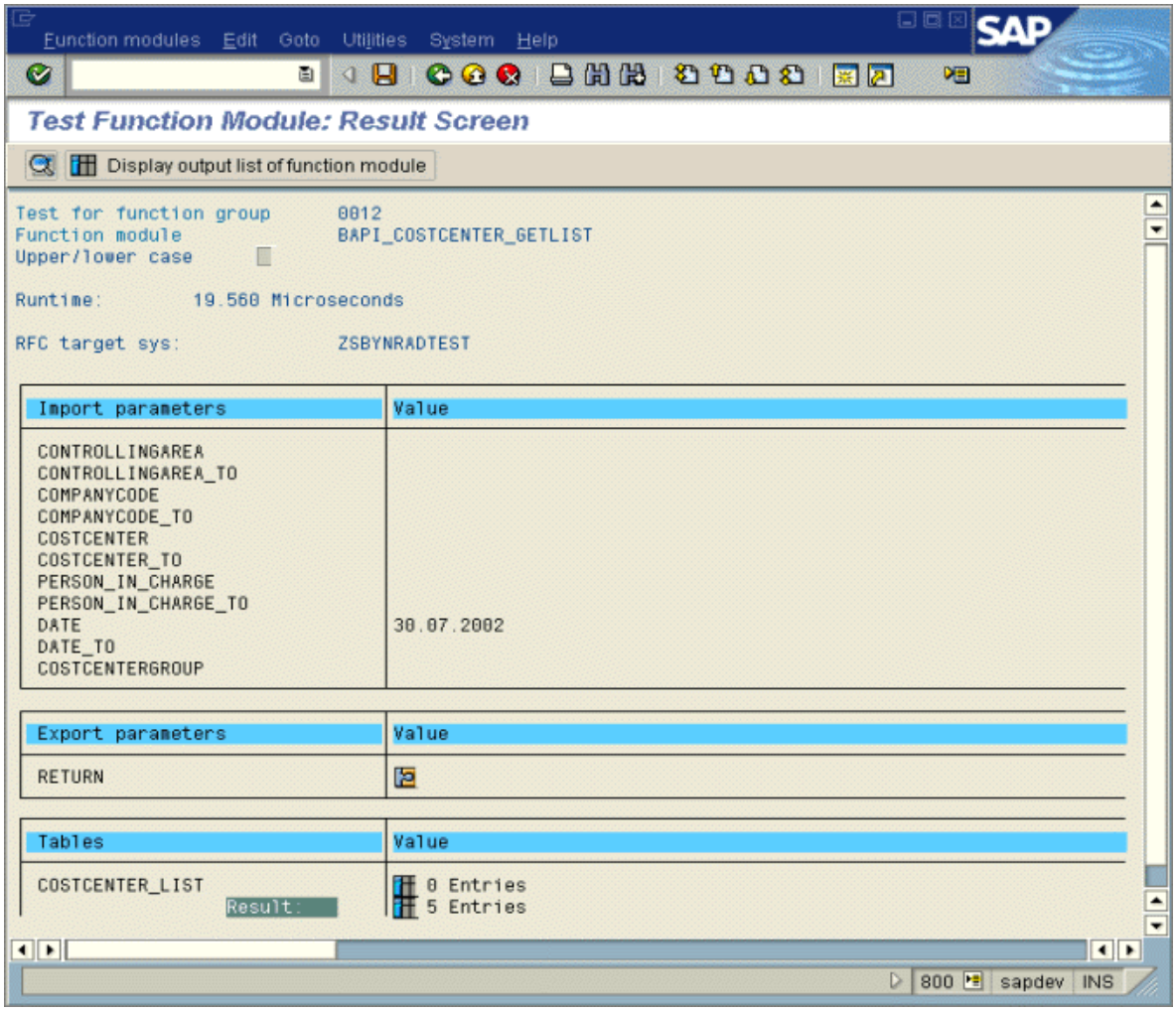
Figure 46 SAP - Test Function Module (Initial)



3 The default data shows up as follows, once the request is executed in the BAPI e*Way.

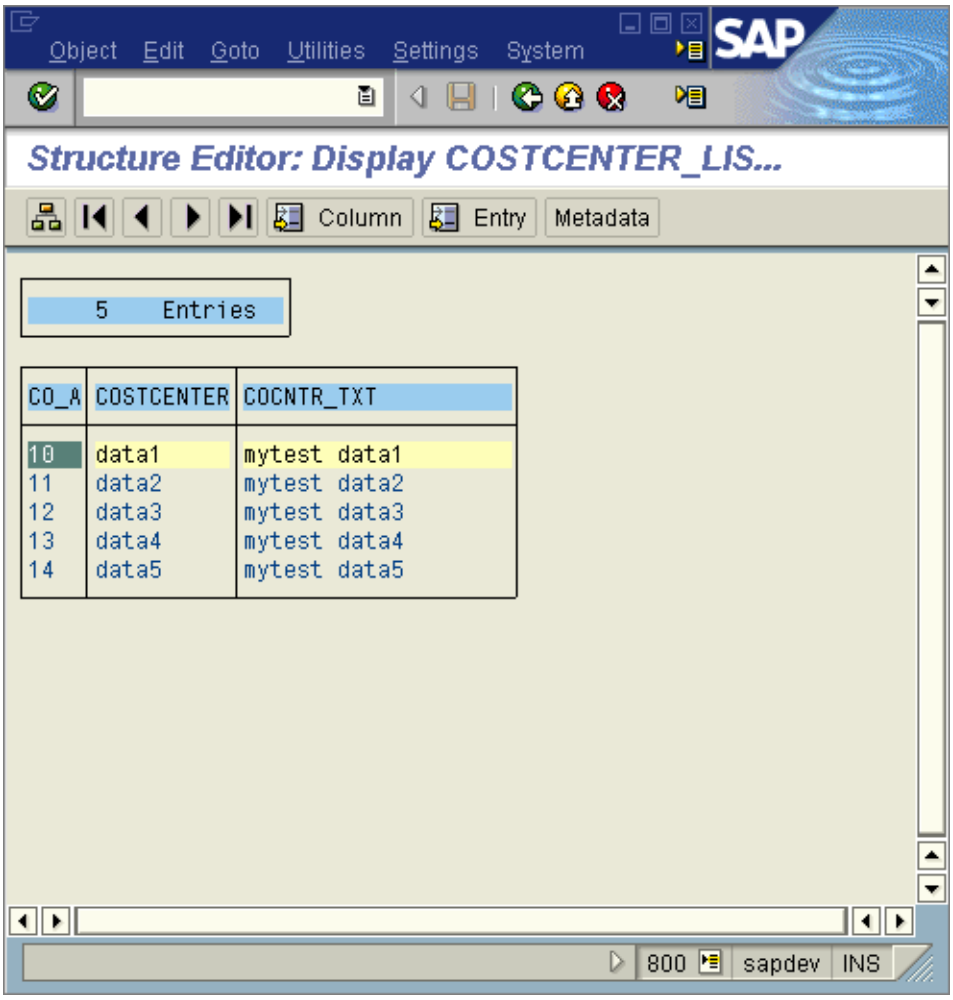
A Select the 5 populated entries (see Figure 47).

Figure 47 SAP - Test Function Module (Result)



B The results appear as shown in Figure 48.

Figure 48 SAP - Structure Editor (Display)

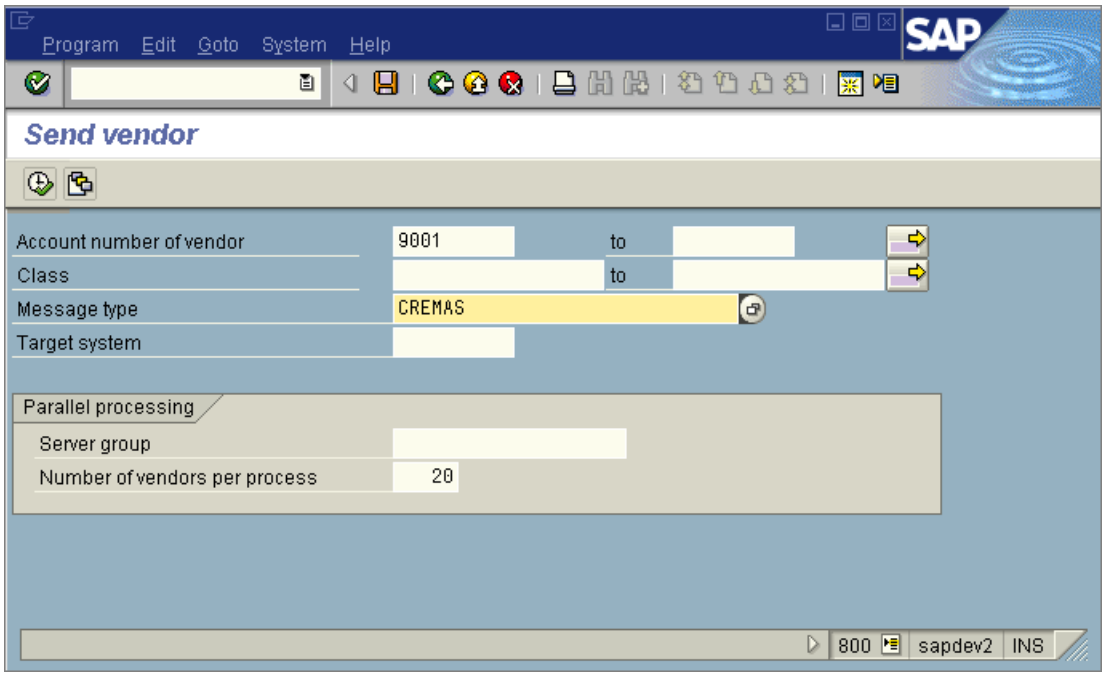


Sending an IDoc

Data can be sent from SAP R/3 to the BAPI e*Way by selecting a vendor in SAP transaction **BD14**, *Send Vendor* (see Figure 49), and then **Executing**.

Note: *The message type is always* CREMAS.

Figure 49 SAP - Send Vendor



4.10.3 Importing Data to SAP R/3

Viewing IDoc Lists

When an IDoc is sent to SAP R/3 successfully, it will be listed in SAP transaction **WE05**, *IDoc Lists*. Figure 50 and Figure 51 illustrate this for a CREMAS message type.

Figure 50 SAP - IDoc Lists

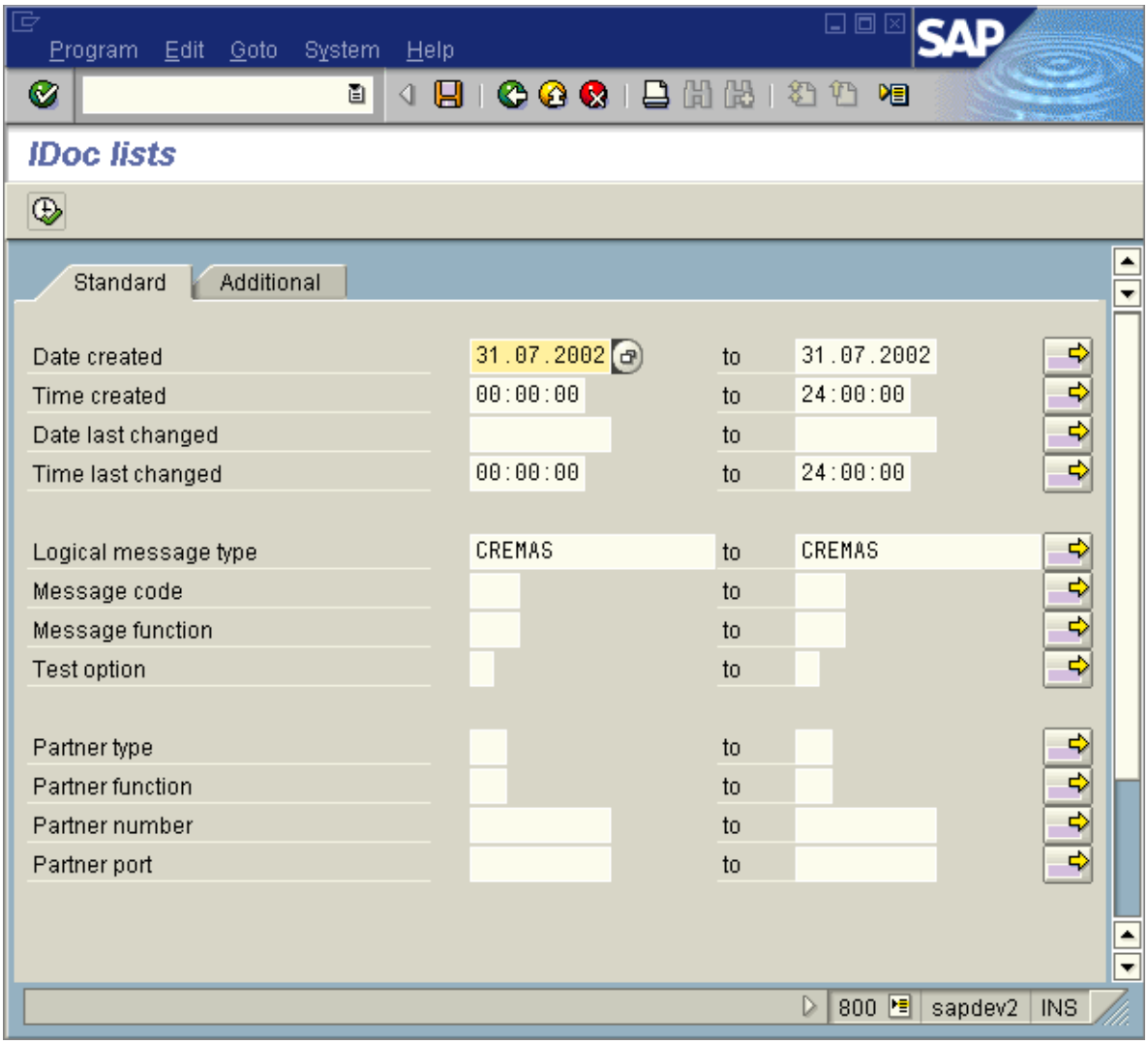
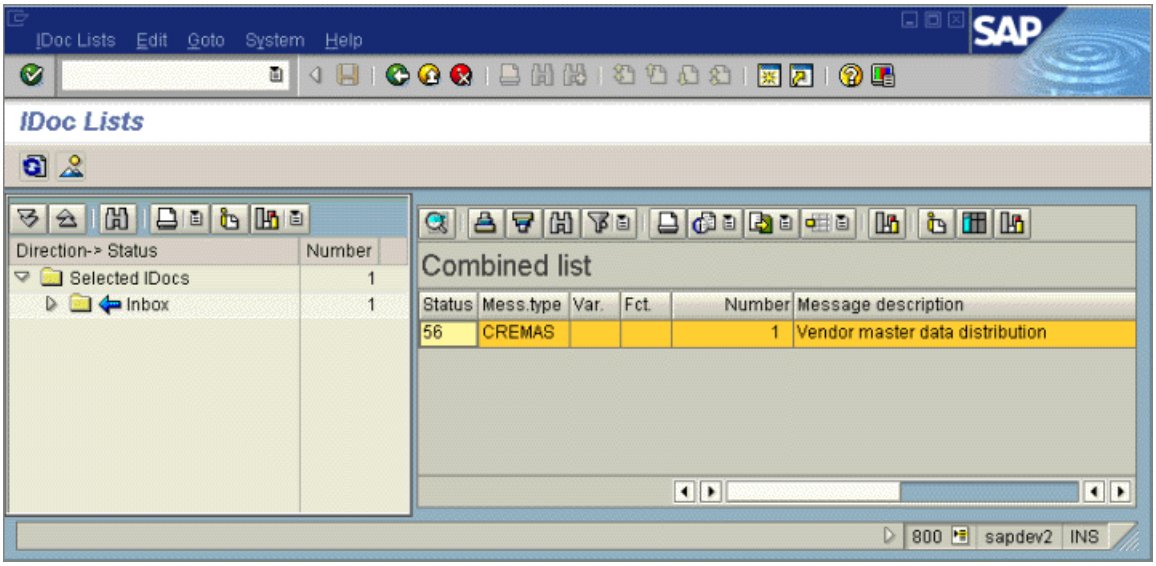


Figure 51 SAP - IDoc Lists (Combined List)



Chapter 5

Sample Schemas

Sample schemas are included in the software package for you to use in testing your system following installation and, if appropriate, as templates that you can modify to produce your own schemas.

Note: *When using SAP Java Connector 2.1.x, all Java Collaborations must have the new SAP JCo files added to the Java Collaboration classpath. For more information see* **Adding Third-party JAR Files to the Collaboration Classpath** *on page 79.*

5.1 Overview

Two sample schemas are provided for the Java SAP BAPI e*Way. They are imported into your system as described in **Optional Example Files** on page 26. This chapter describes the various components of the schemas, and how information is propagated throughout the schemas.

- **BAPI Sample (BapiJava.zip)**

  This schema demonstrates the operation of the SAP BAPI e*Way in both Client (to SAP) and Server (from SAP) mode.

- **IDoc Sample (TrfcAleBapiJava.zip)**

  This schema demonstrates the operation of the e*Way when sending and receiving an IDoc Event using tRFC, by means of the IDOC_INBOUND_ASYNCHRONOUS Remote Function Module. As with the BAPI sample, this schema contains components for both Client (to SAP) and Server (from SAP) mode operation.

Note: *The components of the sample schemas are created when the schema is imported and only require changes to the configuration parameters of the e*Ways and e*Way Connections to match your specific system.*

5.1.1 Connecting to SAP R/3

After importing the sample schemas, you must set e*Way Connection configuration parameters to match those of your specific system (see **Creating e*Way Connections** on page 154). Then run the BAPI Wizard to connect to your installation of SAP R/3. The example shown in **Using the BAPI Wizard** on page 55 demonstrates connecting to the **CostCenter** BAPI.

5.2 BAPI Sample - Client Mode

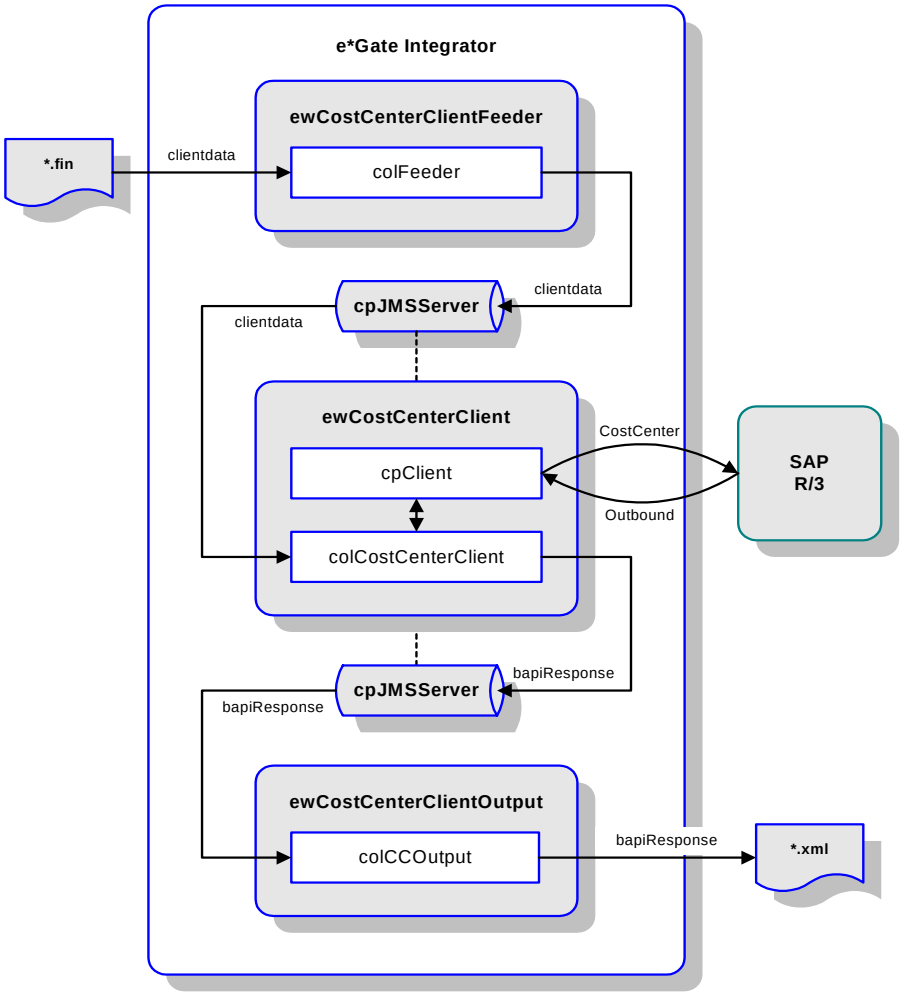
5.2.1 Overview

In this example, a File e*Way, **ewCostCenterClientFeeder**, subscribes to an external input file and sends a sample message, in this case an R 300 (cost controlling area), to the JMSServer, cpJMSServer. The BAPI e*Way, **ewCostCenterClient**, subscribes to the JMSServer, **cpJMSServer**, and publishes the Event to the e*Way Connection **cpC4XClient**.

The Collaboration executes the BAPI CostCenter GetList function, which returns a value of CostCenter and CostCenter text. This information is copied to a BAPI Response ETD and published to the JMSServer. The File e*Way **ewCostClientOutput** subscribes to the JMSServer and publishes the Event to an external output file, which contains a list of the cost center controlling areas and cost center text.

This scenario is conceptually diagrammed in **Figure 52 on page 98**.

Figure 52 BAPI Sample - Client Mode



5.2.2 e*Ways

The Client-mode BAPI sample schema contains three e*Way:

- **ewCostCenterClient**
- **ewCostCenterClientFeeder**
- **ewCostCenterClientOutput**

These e*Ways were created as described in **Creating the e*Way** on page 144. They must be configured for your system as described in **Configuring the Components** on page 108.

ewCostCenterClient

Table 7 BAPI e*Way - ewCostCenterClient

Property	Name/Value
Type	BAPI e*Way
Executable	stceway.exe
Collaboration Performed	colCostCenterClient

ewCostCenterClientFeeder

Table 8 BAPI e*Way - ewCostCenterClientFeeder

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colFeeder

ewCostCenterClientOutput

Table 9 BAPI e*Way - ewCostCenterClientOutput

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colCCOutput

5.2.3 Event Type Definitions

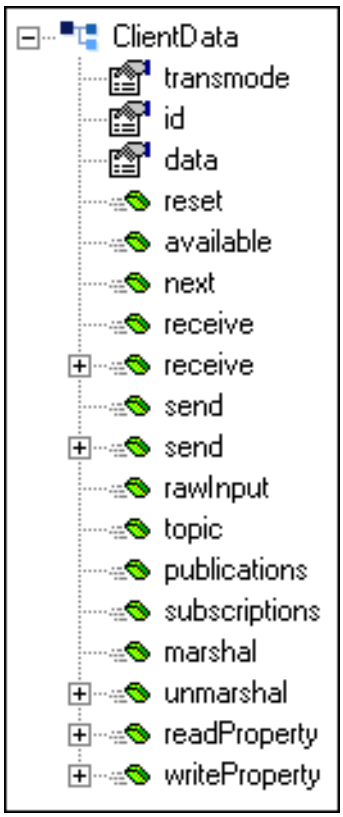
The Client-mode BAPI sample uses three Event Type Definitions (ETDs):

- **ClientData**
- **CostCenter**
- **ClientOut**

ClientData

This ETD is used by the e*Way **ewCostCenterClientFeeder** to retrieve the data from the input file. It was created using the Custom ETD Wizard. The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 53.

Figure 53 BAPI Sample ETD - ClientData

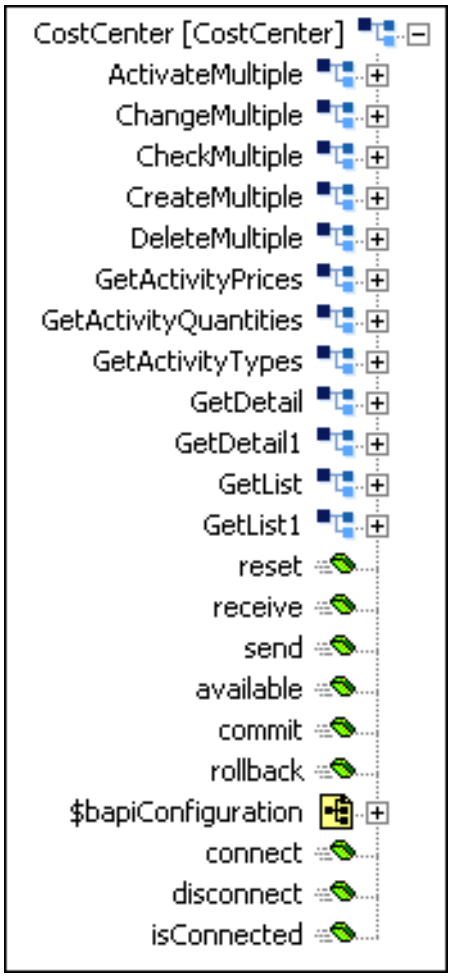


CostCenter

This ETD displays all available BAPIs in the CostCenter Business Object. It was created using the BAPI Wizard (see **Using the BAPI Wizard** on page 55). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 54.

This ETD contains the embedded ETD **$bapiConfiguration**, which is used to set the e*Way Connection configuration parameters (see **Establishing Connections from an ETD** on page 81).

Figure 54 BAPI Sample ETD - CostCenter

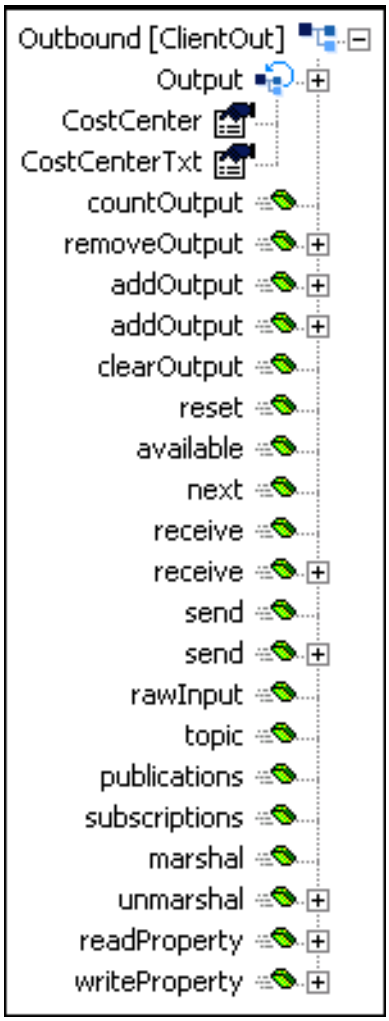


ClientOut

This ETD receives results from the CostCenter BAPI, and writes the results to the output file published by the outbound e*Way. It was created using the Custom ETD Wizard (see **Using the Custom ETD Wizard** on page 63). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 55.

This ETD contains one repeating node, **Output**, which contains two fields: **CostCenter** and **CostCenterTxt**. These respectively contain the list of the cost center controlling areas and cost center text.

Figure 55 BAPI Sample ETD - ClientOut



5.2.4 Collaborations

The e*Ways in the Client-mode BAPI sample schema perform the following Collaborations:

- **colFeeder**
- **colCostCenterClient**
- **colCCOutput**

These Collaboration Rules were created as described in **Creating Collaboration Rules** on page 67. The embedded business rules are defined as described in **Defining Business Rules** on page 70.

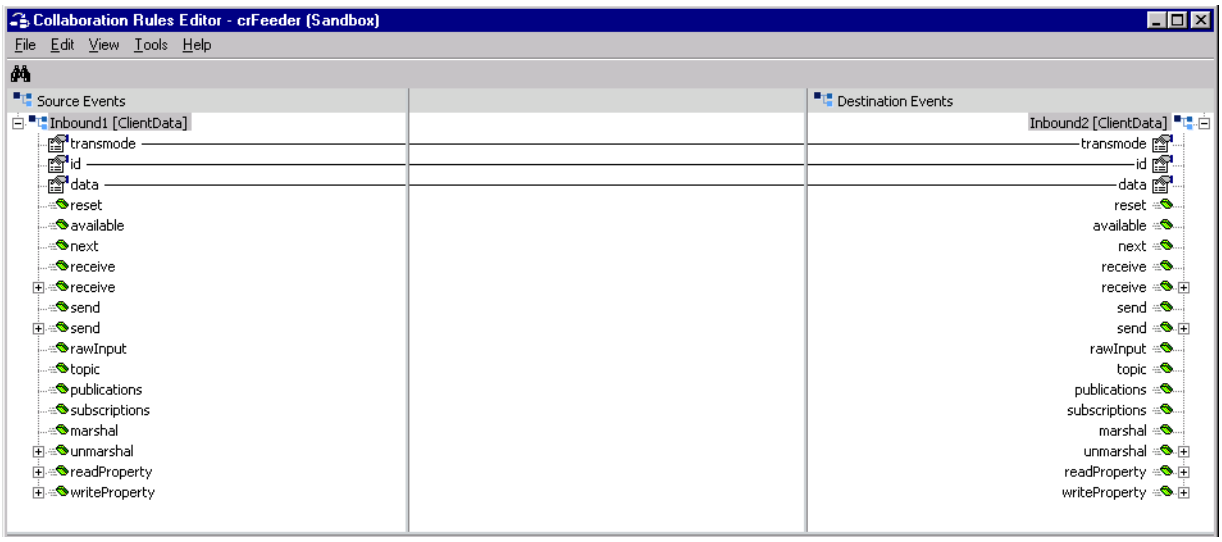
colFeeder

Table 10 Collaboration - colFeeder

Property		Name/Value
Collaboration Rule		crFeeder
Collaboration Service		Java
Performing e*Way		ewCostCenterClientFeeder
Source	ETD	ClientData.xsc
	Instance Name	Inbound1
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	ClientData.xsc
	Instance Name	Inbound2
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)

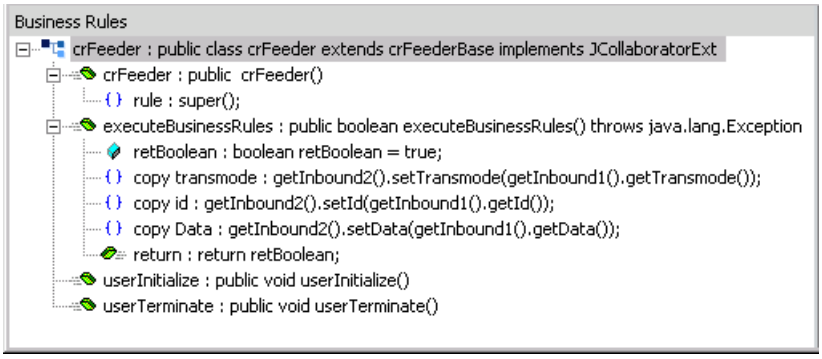
The mapping between the ETDs for **crFeeder**, as displayed in the Collaboration Rules Editor, appear in Figure 56. The Collaboration Rule basically performs a pass-through function, having Source and Destination Events based on the same ETD.

Figure 56 Collaboration Rule Map - crFeeder



The Business Rules for **crFeeder**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 57. The procedure that was followed in creating these business rules is described in **Example: Pass-Through** on page 78.

Figure 57 Business Rules - crFeeder



Business Rules Summary for crFeeder

1. The **Copy** rules simply copy the contents of the respective fields from the Source Event to the Destination Event.
2. The **Publish** rule sends the completed Event to **cpJMSServer.**

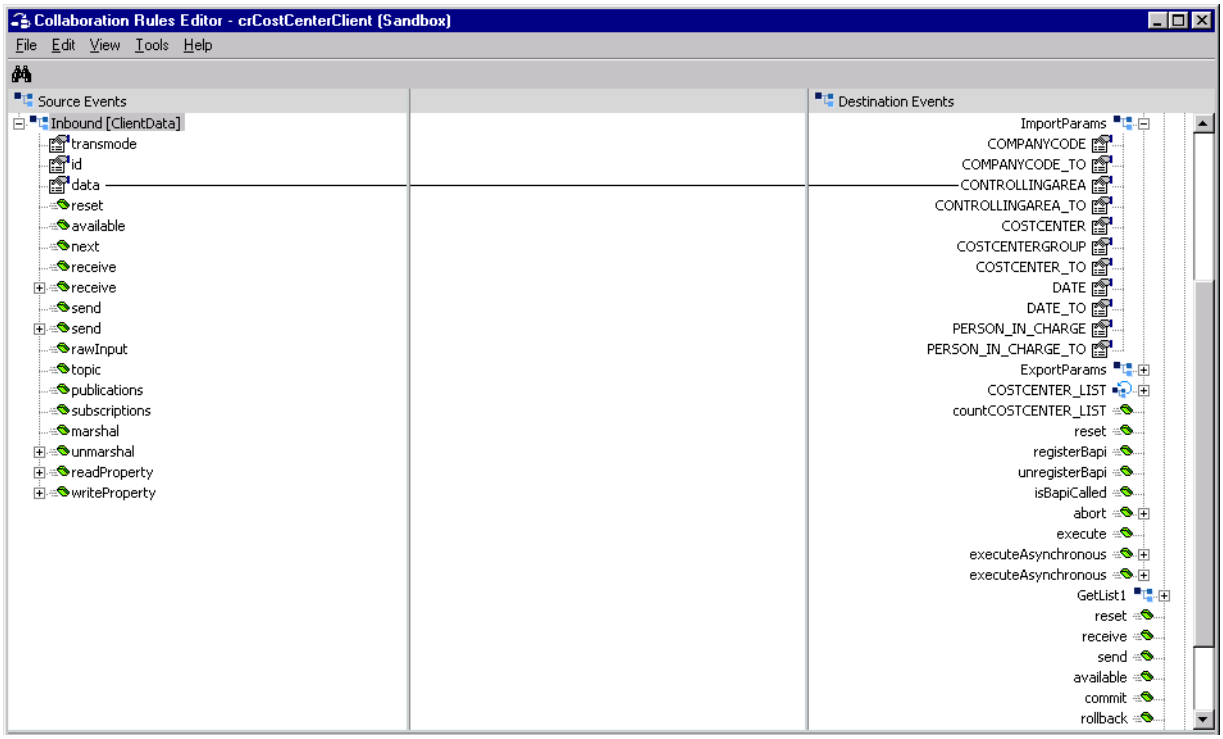
colCostCenterClient

Table 11 Collaboration - colCostCenterClient

Property		Name/Value
Collaboration Rule		crCostCenterClient
Collaboration Service		Java
Performing e*Way		ewCostCenterClient
Source	ETD	ClientData.xsc
	Instance Name	Inbound
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination (1)	ETD	CostCenter.xsc
	Instance Name	CostCenter
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)
Destination (2)	ETD	ClientOut.xsc
	Instance Name	Outbound
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)

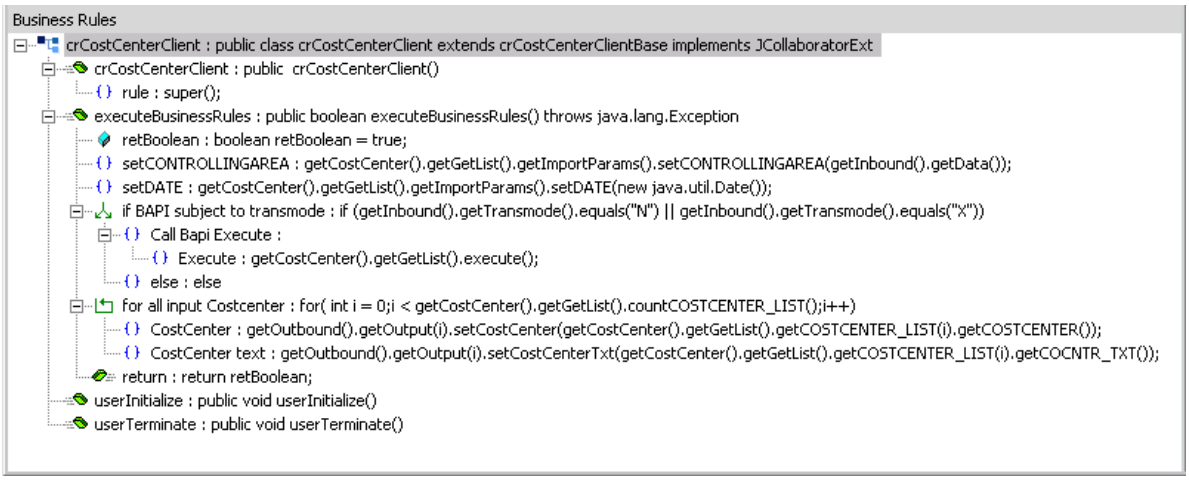
The mapping between the ETDs for **crCostCenterClient**, as displayed in the Collaboration Rules Editor, appear in Figure 58. Note that the Destination Event is defined by two concatenated ETDs: **CostCenter** and **ClientOut**.

Figure 58 Collaboration Rule Map - crCostCenterClient



The business rules for **crCostCenterClient**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 59. The procedure that was followed in defining these business rules is described in **Example: BAPI Client** on page 71.

Figure 59 Business Rules - crCostCenterClient



Business Rules Summary for crCostCenterClient

1. The first **set** rule sets the Controlling Area.
2. The second **set** rule sets the Date.
3. The **if** rule checks whether to call **execute** on a BAPI or on the RFM IDOC_INBOUND_ASYNCHRONOUS.
4. If a BAPI, the **For** loop then loops through the export parameters and copies the data to SAP R/3.

colCCOutput

Table 12 Collaboration - colCCOutput

Property		Name/Value
Collaboration Rule		crEater
Collaboration Service		Java Pass-through
Performing e*Way		ewCostCenterClientOutput
Source	ETD	GenericInEvent.ssc
	Instance Name	JavaPassThroughIn
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	GenericOutEvent.ssc
	Instance Name	JavaPassThroughOut
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)

A Java Pass-through Collaboration Service uses a predefined Collaboration Rule, containing preset Business Rules, that transports data without transforming it.

5.2.5 Configuring the Components

e*Ways

The e*Ways are configured as described in **Configuring the e*Way** on page 146. The respective configuration parameters are listed in the following tables.

ewCostCenterClientFeeder

Table 13 Configuration Parameters - ewCostCenterClientFeeder

Section	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	NO
Outbound Settings	OutputDirectory	data\output
	OutputFileName	output%d.dat (default)
	MultipleRecordsPerFile	YES (default)
	MaxRecordsPerFile	10000 (default)
	AddEOL	YES (default)
Poller Settings	PollDirectory	data
	InputFileMask	*.fin (default)
	PollMilliseconds	1000 (default)
	RemoveEOL	YES (default)
	MultipleRecordsPerFile	NO
	MaxBytesPerLine	4096 (default)
	BytesPerLineIsFixed	NO (default)
	File Records Per eGate Event	1 (default)
Performance Testing	Performance Testing	100 (default)
	InboundDuplicates	1 (default)

ewCostCenterClient

These Configuration Parameters retain their default values.

ewCostCenterClientOutput

Table 14 Configuration Parameters - ewCostCenterClientOutput

Section	Parameter	Value
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
	PerformanceTesting	NO
Outbound Settings	OutputDirectory	output
	OutputFileName	CostCenteroutput%d.xml
	MultipleRecordsPerFile	YES
	MaxRecordsPerFile	10000
	AddEOL	YES
Poller Settings	PollDirectory	data\input
	InputFileMask	*.fin
	PollMilliseconds	1000
	RemoveEOL	YES
	MultipleRecordsPerFile	YES
	MaxBytesPerLine	4096
	BytesPerLineIsFixed	NO
	File Records Per eGate Event	1
Performance Testing	Performance Testing	100
	InboundDuplicates	1

e*Way Connections

Two e*Way Connections were created for the Bapi Client sample. To run this sample schema you must configure the e*Way Connection to match those of your specific system. See **Creating e*Way Connections** on page 154 for procedural information.

cpJMSServer

The e*Way Connection Type is **SeeBeyond JMS**. The parameter values you may need to enter are:

- Host Name (if you have changed it from *localhost*)
- Port Number (if necessary)

For more information on JMS e*Way Connection parameters see the *e*Gate Integrator User's Guide*.

cpClient

The e*Way Connection Type is **SAP BAPI**. The parameter values you need to enter are:

- Application Server Hostname
- Router String (if needed)
- System Number
- Client
- User
- Password
- System ID

See **e*Way Connections** on page 37 for information on the SAP BAPI e*Way Connection Parameters.

5.2.6 Executing the Schema

After configuring all e*Way components, you can execute the IDoc Client sample schema by performing the following steps:

1 Start the schema using the following command:

```
stccb -ln localhost_cb -rh localhost -rs <schema name>-un
    Administrator -up <password>
```

2 Start the Schema Designer and the component e*Ways:

- **ewCostCenterClient**
- **ewCostCenterClientFeeder**
- **ewCostCenterClientOutput**

3 If **ewCostCenterClient** is properly connected, an entry similar to the following appears in **ewCostCenterClient.log**:

```
14:37:15.356 EWY I 2300 (java_extensions.cxx:1073):
BapiConnector.open(): Successfully became a RFC client of:
    R/3 Application Server = sapdev2
    System Number = 00
    User = sbyn1
    Client = 800
    Language = EN
```

If desired, the client connection can be viewed in SAP transaction **SMGW** (see **Viewing Connection Status** on page 85).

4 Copy **clientdata.~in** from the data directory where your schema is located, to the directory specified in the configuration for **ewCostCenterClientFeeder**.

5 Rename the file with the appropriate extension, for example, **clientdata.fin**. The following then occurs:

A The e*Way **ewCostCenterClientFeeder** publishes the data to the JMSServer.

B The e*Way **ewCostCenterClient** retrieves the data from the JMSServer and publishes it to the **cpC4XClient** e*Way Connection.

C Upon execution of the BAPI CostCenter GetList function,a list of cost centers is retrieved from SAP and appears in the **ewCostCenterClient** log file.

```
14:53:52.709 EWY T 2300 (java_extensions.cxx:1073):
     BapiConnector.execute(): Successfully executed function
     [BAPI_COSTCENTER_GETLIST] on R/3
14:53:52.709 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - Dummy cost center
14:53:52.709 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - HR/Admin. cost car
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - Accounting
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - DC: Vehicles
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - Goods Issue
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - Goods Receipt
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
     - Goods Issue
```

```
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Goods Receipt
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Purch.grp retail R30
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Purch.grp retail R30
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Clothing/Sportswear
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Tools/Elect./Apply.
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Food/Drinks
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Clothing/Sportswear
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Tools/Elect./Apply.
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Food/Drinks
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Clothing/Sportswear
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Tools/Elect./Apply.
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Food/Drinks
14:53:52.720 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Clothing/Sportswear
14:53:52.730 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Tools/Elect./Apply.
14:53:52.730 EWY T 2300 (java_extensions.cxx:1073): COCNTR_TEXT
      - Food/Drinks
```

D The e*Way **ewCostCenterClient** publishes this data to the JMSServer.

E Finally, **ewCostCenterClientOutput** retrieves the data and stores it in the external XML file indicated by the configuration.

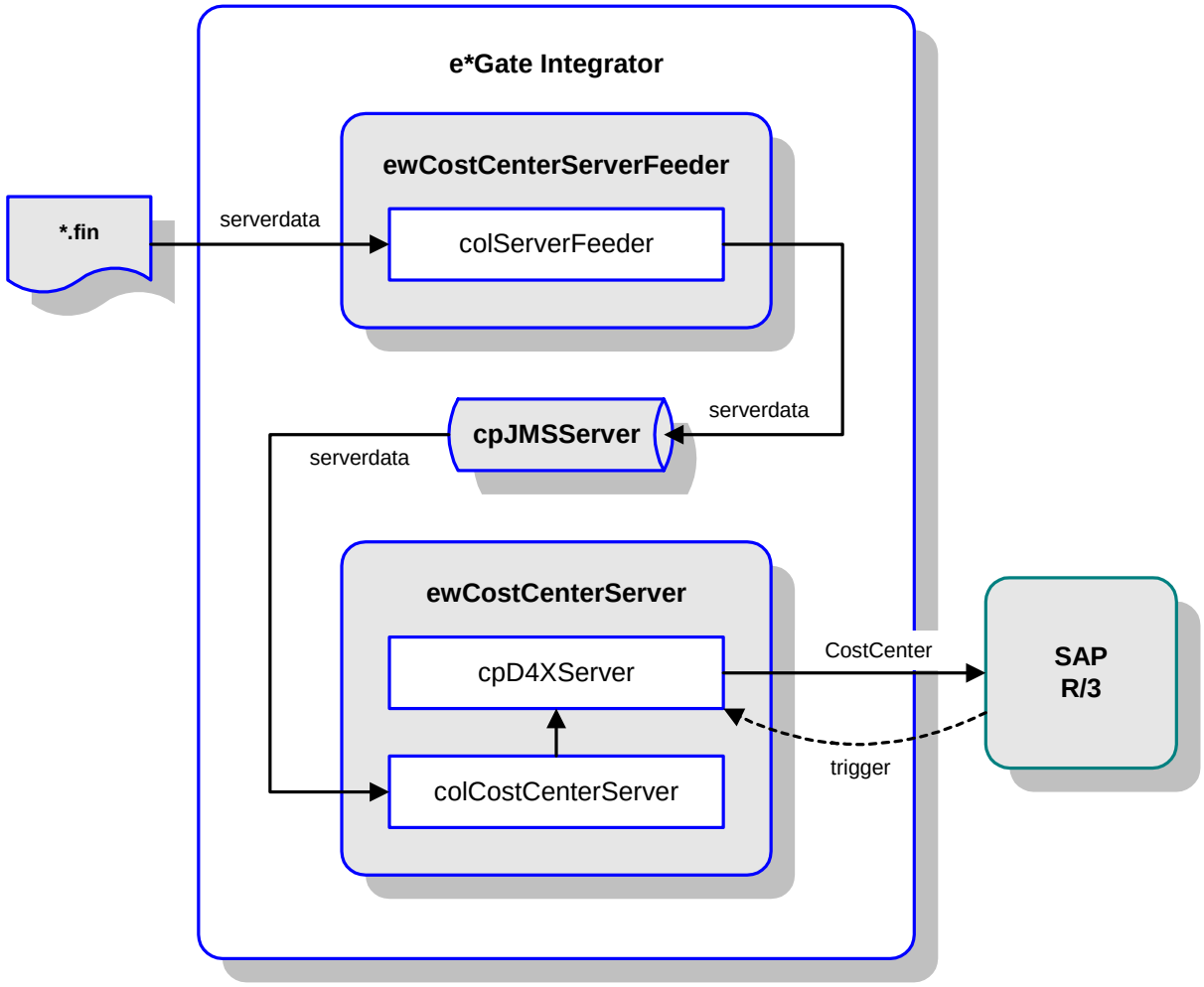
5.3 BAPI Sample - Server Mode

5.3.1 Overview

In this example, a File e*Way, **ewCostCenterServerFeeder**, subscribes to an external input file containing five records and publishes it to **cpJMSServer**, which acts as an IQ. The BAPI e*Way, **ewCostCenterServer**, subscribes to **cpJMSServer** and publishes to the e*Way Connection, **cpD4XServer**, which checks for the connection to SAP and BAPI registration, and then waits for a request from SAP to insert the data. When the trigger is received, the sample data is placed in the COSTCENTER_LIST table and can be viewed within SAP R/3.

This scenario is conceptually diagrammed in Figure 60.

Figure 60 BAPI Sample - Server Mode



5.3.2 e*Ways

The Server-mode BAPI sample schema contains two e*Ways:

- **ewCostCenterServer**
- **ewCostCenterServerFeeder**

These e*Ways were created as described in **Creating the e*Way** on page 144. They must be configured for your system as described in **Configuring the Components** on page 120.

ewCostCenterServer

Table 15 BAPI e*Way - ewCostCenterServer

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colCostCenterServer

ewCostCenterServerFeeder

Table 16 BAPI e*Way - ewCostCenterServerFeeder

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colServerFeeder

5.3.3 Event Type Definitions

The Server-mode BAPI sample schema uses two Event Type Definitions (ETDs):

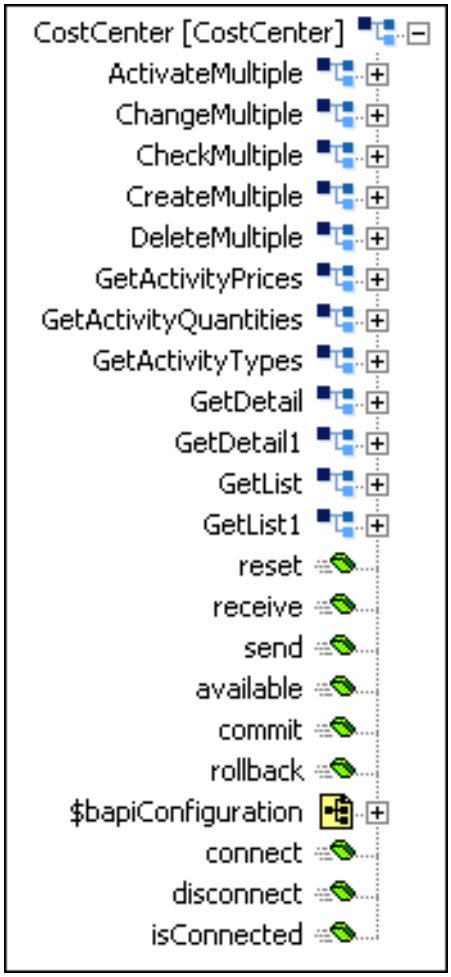
- **CostCenter.xsc**
- **ServerData.xsc**

CostCenter

This ETD displays all available BAPIs in the CostCenter Business Object. It was created using the BAPI Wizard (see **Using the BAPI Wizard** on page 55). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 61.

This ETD contains the embedded ETD **$bapiConfiguration**, which is used to set the e*Way Connection configuration parameters (see **Establishing Connections from an ETD** on page 81).

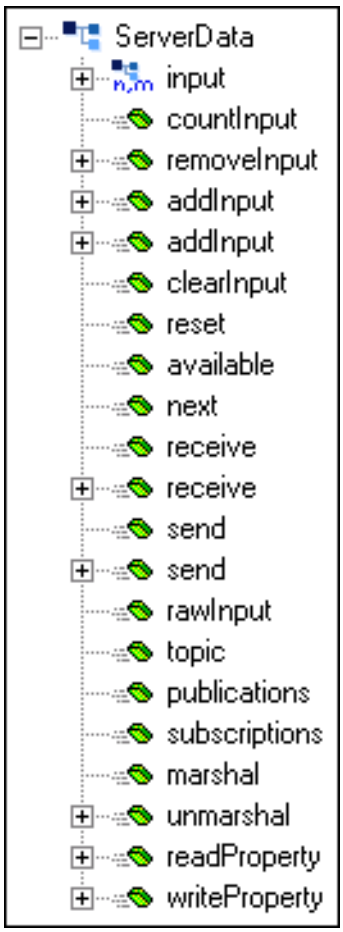
Figure 61 BAPI ETD - CostCenter



ServerData

This ETD is used by the e*Way **ewCostCenterServerFeeder** to retrieve the data from the input file. It was created using the Custom ETD Wizard (see **Using the Custom ETD Wizard** on page 63). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 62.

Figure 62 BAPI Server-Mode ETD - ServerData



5.3.4 Collaborations

The e*Ways in the Server-mode BAPI sample schema perform the following Collaborations:

- **colServerFeeder**
- **colCostCenterServer**

These Collaboration Rules were created as described in **Creating Collaboration Rules** on page 67. The embedded business rules are defined as described in **Defining Business Rules** on page 70.

colServerFeeder

Table 17 Collaboration - colServerFeeder

Property		Name/Value
Collaboration Rule		crServerFeeder
Collaboration Service		Java Pass-through
Performing e*Way		ewCostCenterServer
Source	ETD	GenericInEvent.ssc
	Instance Name	JavaPassThroughIn
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	GenericOutEvent.ssc
	Instance Name	JavaPassThroughOut
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)

A Java Pass-through Collaboration Service uses a predefined Collaboration Rule, containing preset Business Rules, that transports data without transforming it.

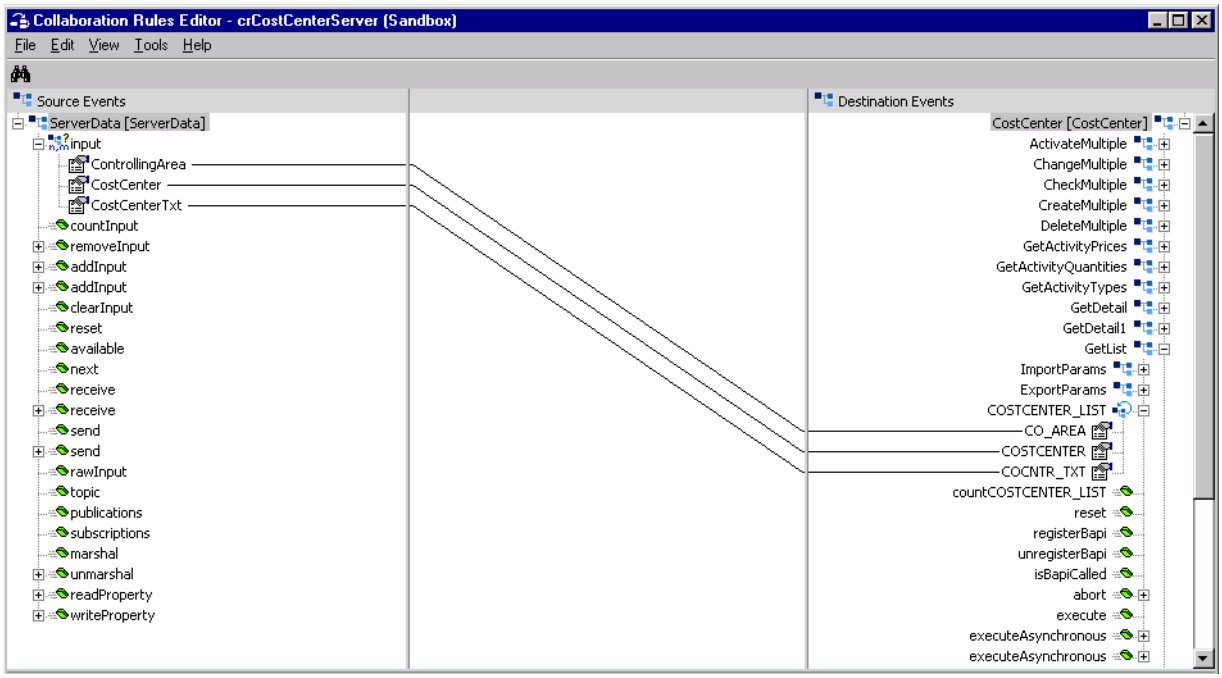
colCostCenterServer

Table 18 Collaboration - colCostCenterServer

Property		Name/Value
Collaboration Rule		crCostCenterServer
Collaboration Service		Java
Performing e*Way		ewCostCenterServerFeeder
Source	ETD	ServerData.xsc
	Instance Name	ServerData
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	CostCenter.xsc
	Instance Name	CostCenter
	Mode	Out
	Trigger	N/A
	Manual Publish	(checked)

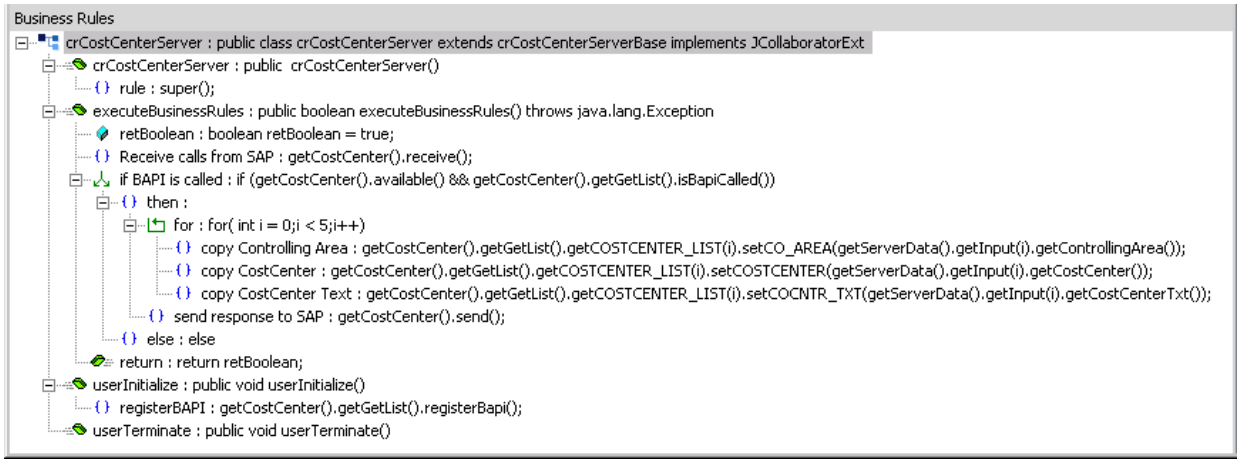
The mapping between the ETDs for **crCostCenterServer**, as displayed in the Collaboration Rules Editor, appear in Figure 63.

Figure 63 Collaboration Rule Map - crCostCenterServer



The Business Rules for **crCostCenterServer**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 64.

Figure 64 Business Rules - crCostCenterServer



Business Rules Summary for crCostCenterServer

1. Under **userInitialize**, the **registerBAPI** rule registers the BAPI e*Way with SAP R/3.
2. The **Receive** rule then responds to a call from SAP.
3. The **if** rule checks whether to call **execute** on a BAPI or on the RFM IDOC_INBOUND_ASYNCHRONOUS.
4. If a BAPI is detected, the **For** loop then loops through the import parameters and copies the data from SAP R/3.
5. The **send response to SAP** rule contains the explicit **send()** command (Manual mode).
6. The **else** rule concludes the **if ... else** pair.

5.3.5 Configuring the Components

e*Ways

The e*Ways are configured as described in **Configuring the e*Way** on page 146. The respective configuration parameters are listed in the following tables.

ewCostCenterServerFeeder

Table 19 Configuration Parameters - ewCostCenterServerFeeder

Section	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	(default)
Outbound (send) Settings	OutputDirectory	data\output
	OutputFileName	output%d.dat (default)
	MultipleRecordsPerFile	YES (default)
	MaxRecordsPerFile	10000 (default)
	AddEOL	YES (default)
Poller (inbound) Settings	PollDirectory	data\input
	InputFileMask	*.fin (default)
	PollMilliseconds	1000 (default)
	RemoveEOL	YES (default)
	MultipleRecordsPerFile	NO
	MaxBytesPerLine	4096 (default)
	BytesPerLineIsFixed	NO (default)
	File Records Per eGate Event	1 (default)
Performance Testing	Performance Testing	100 (default)
	InboundDuplicates	1 (default)

ewCostCenterServer

These Configuration Parameters retain their default values.

e*Way Connections

Two e*Way Connections were created for the BAPI Server sample. To run this sample schema you must configure the e*Way Connection to match those of your specific system. See **Creating e*Way Connections** on page 154 for procedural information.

cpJMSServer

The e*Way Connection Type is **SeeBeyond JMS**. The parameter values you may need to enter are:

- Host Name (if you have changed it from *localhost*)
- Port Number (if necessary)

For more information on JMS e*Way Connection parameters see the *e*Gate Integrator User's Guide*.

cpD4XServer

The e*Way Connection Type is **SAP BAPI**. In the Server section, the **Wait for request interval** parameter is set to 60000 milliseconds to give the e*Way adequate time to receive requests from SAP.

The parameter values you need to enter are:

- Application Server Hostname
- Router String (if needed)
- System Number
- Client
- User
- Password
- System ID

See **e*Way Connections** on page 37 for information on the SAP BAPI e*Way Connection Parameters.

5.3.6 Executing the Schema

After configuring all e*Way components, you can execute the BAPI Server sample schema by performing the following steps:

1. Start the schema using the following command:

```
stccb -ln localhost_cb -rh localhost -rs <schema name>-un
    Administrator -up <password>
```

2. If **ewCostCenterServer** is properly connected, a message similar to the following appears in the **ewCostCenterServer** log file:

```
15:22:23.978 EWY I 2532 (java_extensions.cxx:1073):
BapiConnector.open(): Successfully registered as a RFC server
    with:
    Program ID = ZSBYNRADTEST
    SAP Gateway = sapdev
    Service = sapgw00
    SID = SBYN
```

   The connection can be tested by using SAP transaction **SM59** (see **Testing an RFC Connection** on page 86).

3. Copy **serverdata.~in** from the data directory where the schema is located to the directory specified in the configuration for **ewCostCenterServerFeeder**.

4. Rename the file with the appropriate extension, for example, **serverdata.fin**. The following then occurs:

   A. The **ewCostCenterServerFeeder** e*Way publishes the data to the JMSServer.

   B. The **ewCostCenterServer** e*Way retrieves the data and publishes it to the **cpD4XServer** e*Way Connection.

5. Send a request from SAP to the **ewCostCenterServer** e*Way as described in **Sending a Request** on page 90.

   The **cpD4XServer** e*Way Connection sends the data to SAP after receiving the request.

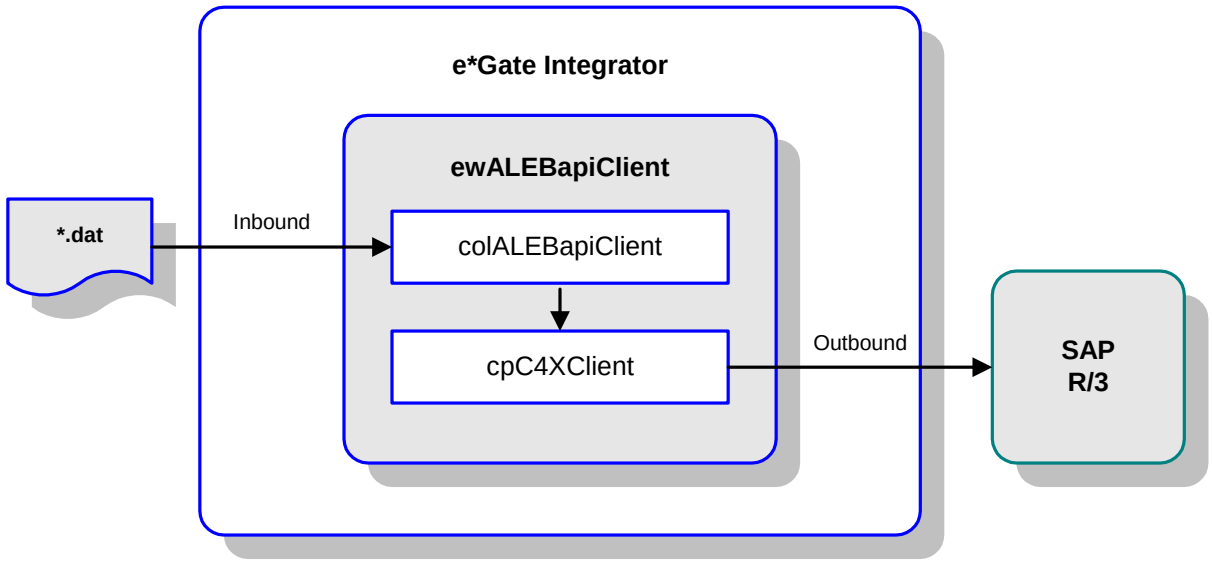
5.4 IDoc Sample - Client Mode

5.4.1 Overview

A File e*Way, **ewALEBapiClient**, subscribes to an external input file containing an IDoc record and publishes it to the **cpC4XClient** e*Way Connection. The record is then inserted into the SAP database using the **executeAsynchronous** method. The IDoc employed is a Vendor Master Data Distribution (CREMAS03) record.

This scenario is conceptually diagrammed in Figure 65.

Figure 65 IDoc Sample - Client Mode



5.4.2 e*Way

The Client-mode IDoc sample schema contains a single e*Way, as described in Table 20. This e*Ways was created as described in **Creating the e*Way** on page 144. It must be configured for your system as described in **Configuring the Components** on page 129.

ewALEBapiClient

Table 20 BAPI e*Way - ewALEBapiClient

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colALEBapiClient

5.4.3 ETDs

The Client-mode IDoc sample schema uses two Event Type Definitions (ETDs).

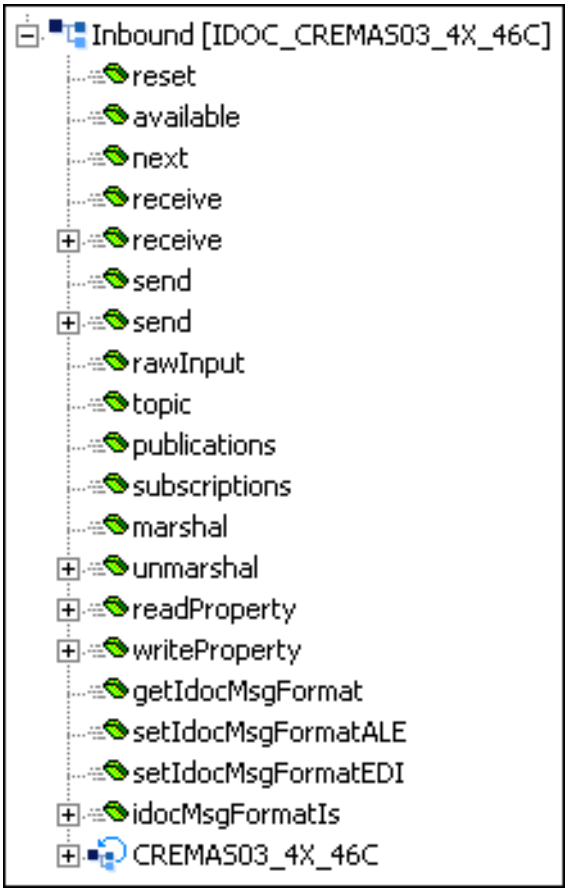
- **IDOC_CREMAS03_46C**
- **IDOC_INBOUND_ASYNCHRONOUS**

IDOC_CREMAS03_46C

This ETD was created with the IDoc Wizard (see **Using the IDoc Wizard** on page 59). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 66.

Note: *In this sample schema,* IDOC_CREMAS03_46C.xsc *is the default ETD file. If you run this schema under e*Gate 4.5.2 or 4.5.1, you must replace this file with* IDOC_CREMAS03_46C_452.xsc *or* IDOC_CREMAS03_46C_451.xsc*, respectively. You also must change the module workslice to point to 4.5.2 or 4.5.1, as is appropriate (the default is 4.5.1).*

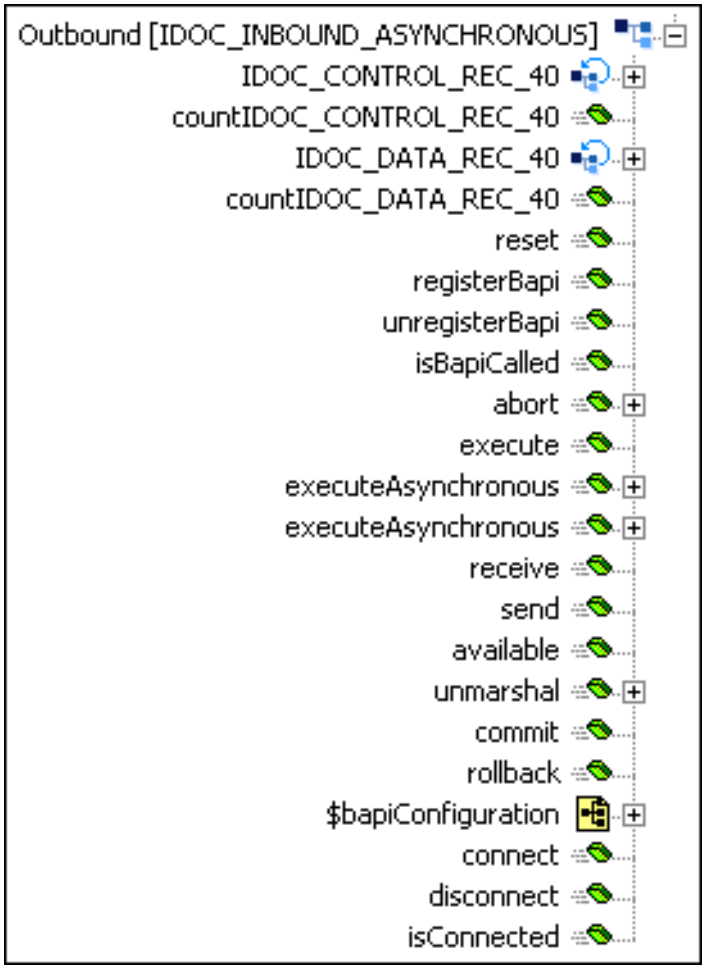
Figure 66 IDoc Client Sample ETD - IDOC_CREMAS03_46C



IDOC_INBOUND_ASYNCHRONOUS

This ETD was created with the BAPI Wizard, using the RFC object type option (see **Using the BAPI Wizard** on page 55). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 67.

Figure 67 IDoc Client Sample ETD - IDOC_INBOUND_ASYNCHRONOUS



5.4.4 Collaborations

The e*Way in the Client-mode IDoc sample schema performs the following Collaboration:

- **colALEBapiClient**

These Collaboration Rules were created as described in **Creating Collaboration Rules** on page 67. The embedded business rules are defined as described in **Defining Business Rules** on page 70.

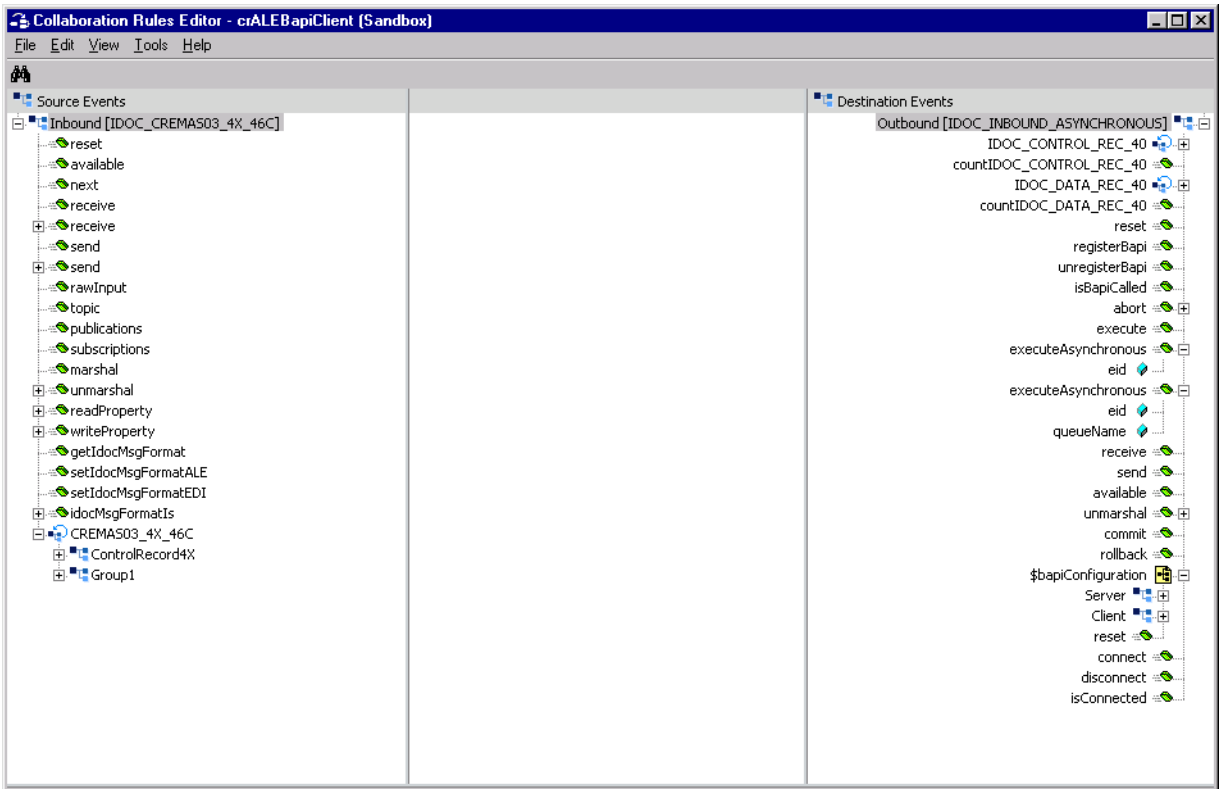
colALEBapiClient

Table 21 Collaboration - colALEBapiClient

Property		Name/Value
Collaboration Rule		crALEBapiClient
Collaboration Service		Java
Performing e*Way		ewALEBapiClient
Source	ETD	IDOC_CREMAS03_46C.xsc
	Instance Name	Inbound
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	IDOC_INBOUND_ASYNCHRONOUS.xsc
	Instance Name	Outbound
	Mode	Out
	Trigger	N/A
	Manual Publish	(checked)

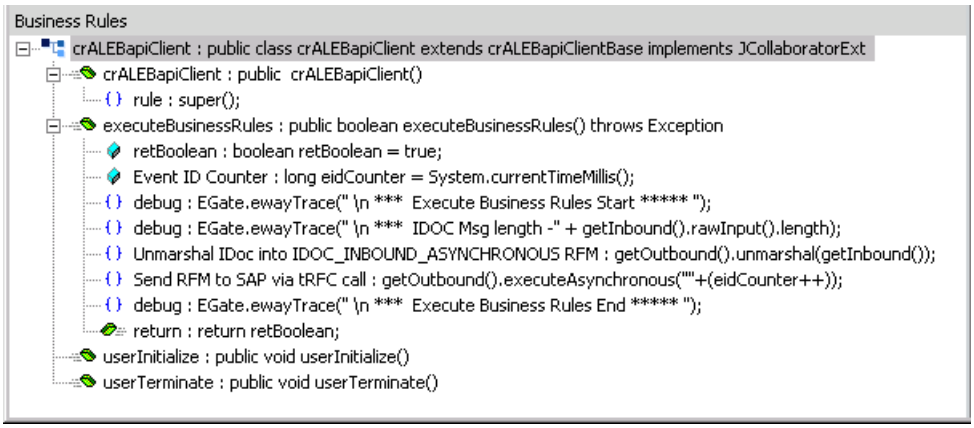
The mapping between the ETDs for **crALEBapiClient**, as displayed in the Collaboration Rules Editor, appear in Figure 68.

Figure 68 Collaboration Rule Map - crALEBapiClient



The Business Rules for **crALEBapiClient**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 69. The procedure that was followed in creating the business rules contained in the **crALEBapiClient** Collaboration Rule is described in **Example: IDoc Client** on page 75.

Figure 69 Business Rules - crALEBapiClient



Business Rules Summary for crALEBapiClient

1. The **Unmarshal** rule converts the IDoc into a linear structure, allowing it to be copied to the single node IDOC_INBOUND_ASYNCHRONOUS as a BLOB.
2. The **Send** rule transmits the RFM IDOC_INBOUND_ASYNCHRONOUS to SAP via tRFC, incrementing the EID counter.
3. The **debug** rules simply add annotation to the log.

5.4.5 Configuring the Components

e*Ways

The e*Way is configured as described in **Configuring the e*Way** on page 146. The configuration parameters are listed in the following table.

ewALEBapiClient

Table 22 Configuration Parameters - ewALEBapiClient

Section	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	(default)
Outbound (send) Settings	OutputDirectory	(default)
	OutputFileName	output%d.dat (default)
	MultipleRecordsPerFile	YES (default)
	MaxRecordsPerFile	10000 (default)
	AddEOL	YES (default)
Poller (inbound) Settings	PollDirectory	data\input
	InputFileMask	*.dat
	PollMilliseconds	1000 (default)
	RemoveEOL	YES (default)
	MultipleRecordsPerFile	NO
	MaxBytesPerLine	40960
	BytesPerLineIsFixed	NO (default)
	File Records Per eGate Event	1 (default)
Performance Testing	Performance Testing	100 (default)
	InboundDuplicates	1 (default)

e*Way Connections

One e*Way Connection was created for the IDoc Client sample. To run this sample schema you must configure the e*Way Connection to match those of your specific system. See **Creating e*Way Connections** on page 154 for procedural information.

cpC4XClient

The e*Way Connection Type is **SAP BAPI**. The parameter values you need to enter are:

- Application Server Hostname
- Router String (if needed)
- System Number
- Client
- User
- Password
- System ID

See **e*Way Connections** on page 37 for information on the SAP BAPI e*Way Connection Parameters.

5.4.6 Executing the Schema

After configuring all e*Way components, you can execute the IDoc Client sample schema by performing the following steps:

1. Start the schema using the following command:

```
stccb -ln localhost_cb -rh localhost -rs <schema name>-un
    Administrator -up <password>
```

2. Start the Schema Designer and start the **ewAleBapiClient** e*Way.
3. Copy **CREMAS03.dat** from the data directory where your schema is located to the directory specified in the configuration for **ewAleBapiClient**. The following then occurs:
   A. The **ewAleBapiClient** e*Way takes the file and publishes the data to the **cpC4XClient** e*Way Connection.
   B. The e*Way connects to the SAP system. If the **ewAleBapiClient** e*Way connects properly, a message similar to the following appears in the **ewAleBapiClient** log file:

```
11:04:52.877 EWY I 2664 (java_extensions.cxx:1073):
BapiConnector.open(): Successfully became a RFC client of:
      R/3 Application Server = sapdev2
      System Number = 00
      User = sbyn1
      Client = 800
      Language = EN
```

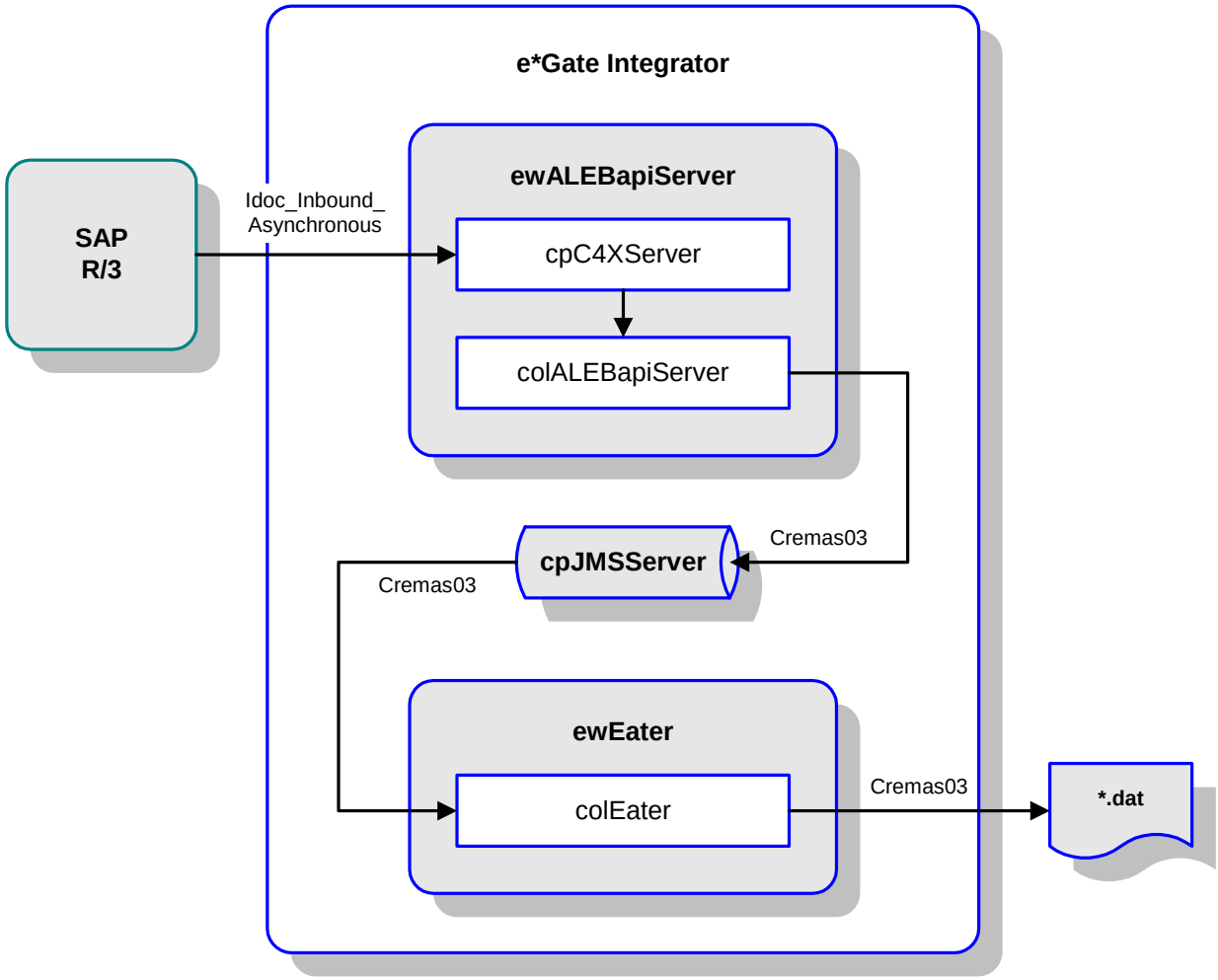
   If desired, the client connection can be viewed in SAP transaction **SMGW** (see **Viewing Connection Status** on page 85).
   C. The **ewAleBapiClient** e*Way executes IDOC_INBOUND_ASYNCHRONOUS using a unique TID. The **CREMAS03** record now should appear in SAP transaction **WE05**, *IDoc Lists* (see **Viewing IDoc Lists** on page 95).

5.5 IDoc Sample - Server Mode

5.5.1 Overview

In the IDoc Server-mode sample, **ewAleBapiServer** receives IDoc data from SAP R/3 via the BAPI e*Way Connection (**cpC4XServer**) and publishes it to the JMSServer (**cpJMSServer**). The **ewEater** e*Way subscribes to the JMSServer queue and publishes the data to a file. The IDoc employed is a Vendor Master Data Distribution (CREMAS03) record. This scenario is conceptually diagrammed in Figure 70.

Figure 70 IDoc Sample - Server Mode



5.5.2 e*Ways

The Server-mode IDoc sample schema contains two e*Ways:

- **ewALEBapiServer**
- **ewEater**

These e*Ways were created as described in **Creating the e*Way** on page 144. They must be configured for your system as described in **Configuring the Components** on page 140.

ewALEBapiServer

Table 23 BAPI e*Way - ewALEBapiserver

Property	Name/Value
Type	BAPI e*Way
Executable	stceway.exe
Collaboration Performed	colALEBapiServer

ewEater

Table 24 BAPI e*Way - ewEater

Property	Name/Value
Type	File e*Way
Executable	stcewfile.exe
Collaboration Performed	colEater

5.5.3 ETDs

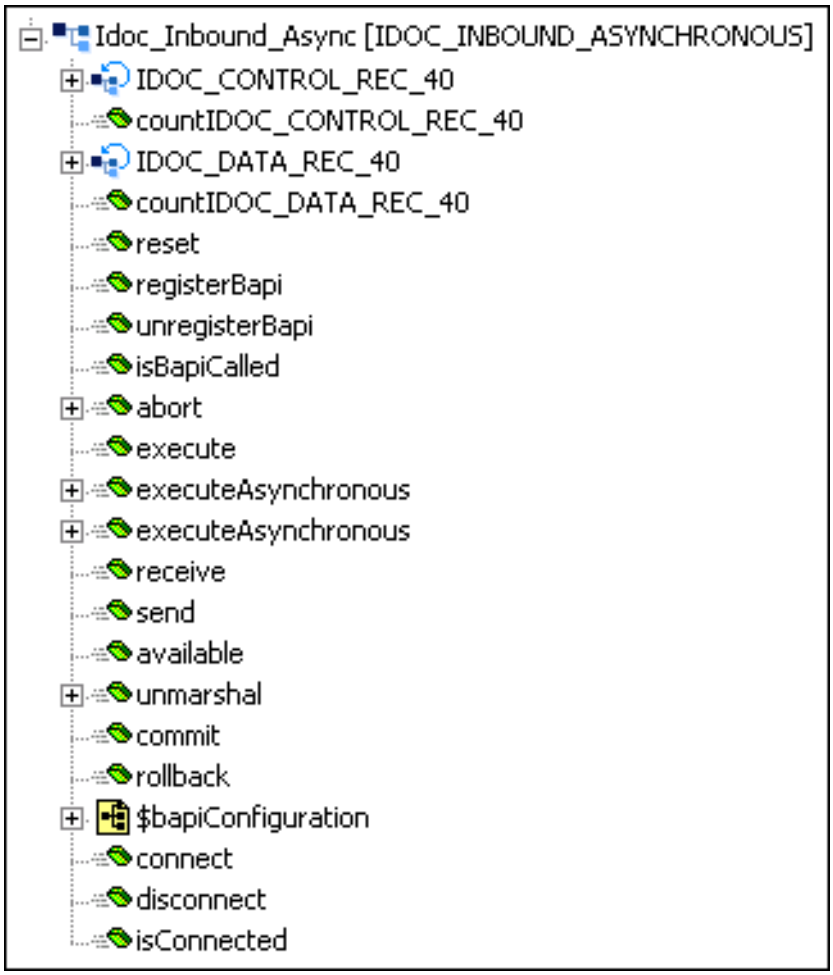
The Server-mode IDoc sample schema uses two Event Type Definitions (ETDs).

- **IDOC_INBOUND_ASYNCHRONOUS**
- **IDOC_CREMAS03_46C**

IDOC_INBOUND_ASYNCHRONOUS

This ETD was created with the BAPI Wizard, using the RFC object type option (see **Using the BAPI Wizard** on page 55). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 67.

Figure 71 IDoc Server Sample ETD - IDOC_INBOUND_ASYNCHRONOUS

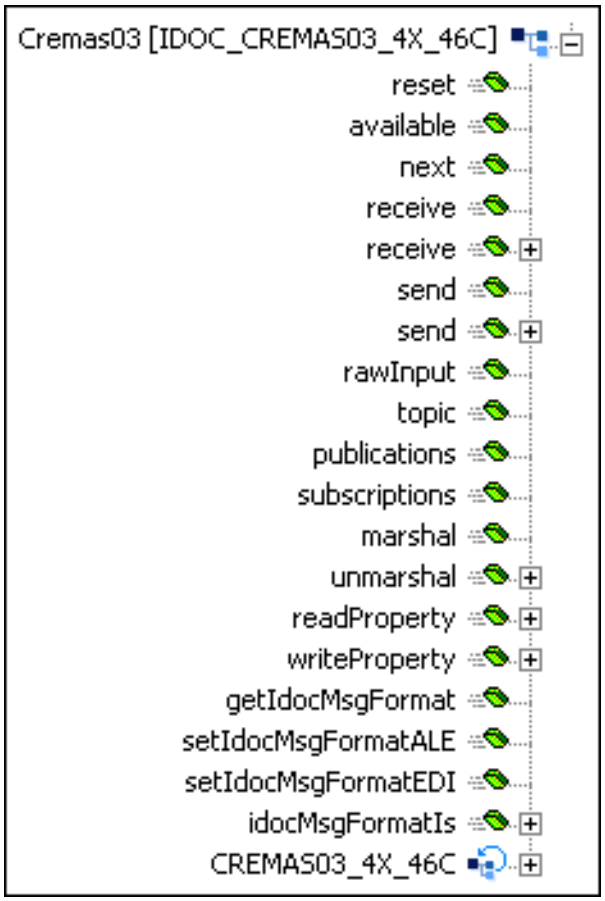


IDOC_CREMAS03_46C

This ETD was created with the IDoc Wizard (see **Using the IDoc Wizard** on page 59). The structure of the ETD, as displayed in the Collaboration Rules Editor, is shown in Figure 66.

Note: *In this sample schema,* IDOC_CREMAS03_46C.xsc *is the default ETD file. If you run this schema under e*Gate 4.5.2 or 4.5.1, you must replace this file with* IDOC_CREMAS03_46C_452.xsc *or* IDOC_CREMAS03_46C_451.xsc*, respectively. You also must change the module workslice to point to 4.5.2 or 4.5.1, as is appropriate (the default is 4.5.1).*

Figure 72 IDoc Server Sample ETD - IDOC_CREMAS03_46C



5.5.4 Collaborations

The e*Ways in the Server-mode IDoc sample schema respectively perform the following Collaboration:

- **colALEBapiServer**
- **colEater**

These Collaboration Rules were created as described in **Creating Collaboration Rules** on page 67. The embedded business rules were defined as described in **Defining Business Rules** on page 70.

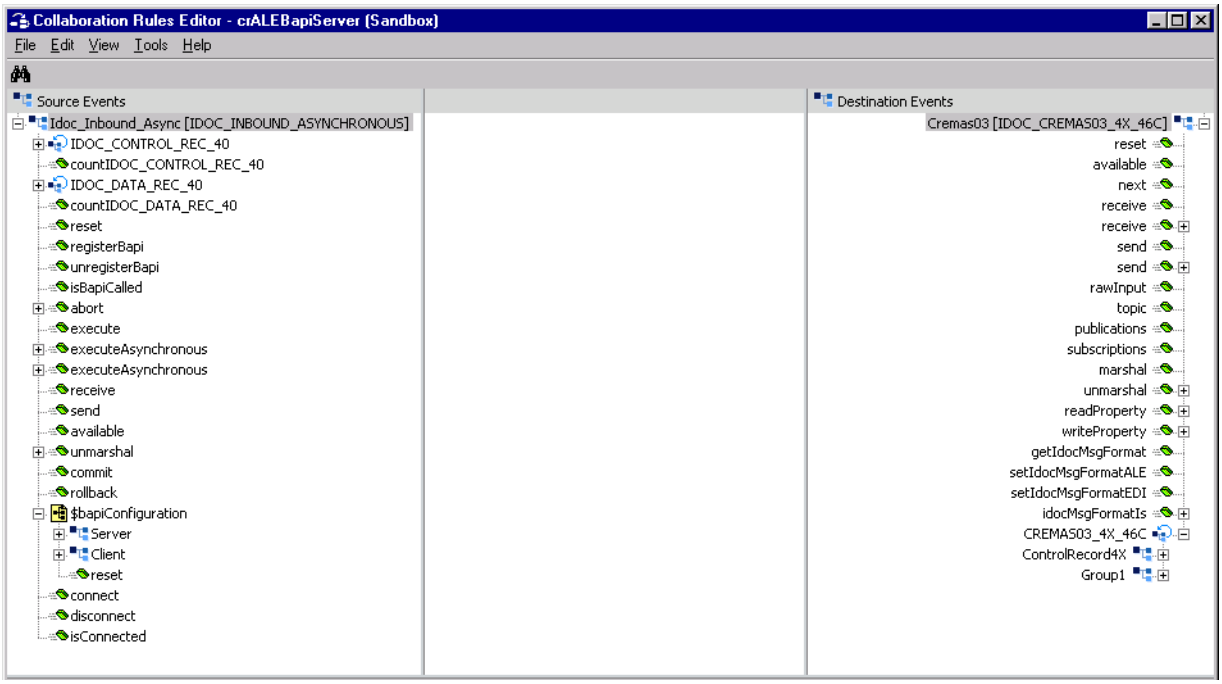
colALEBapiServer

Table 25 Collaboration - colALEBapiServer

Property		Name/Value
Collaboration Rule		crALEBapiServer
Collaboration Service		Java
Performing e*Way		ewALEBapiServer
Source	ETD	IDOC_INBOUND_ASYNCHRONOUS.xsc
	Instance Name	Idoc_Inbound_Asynchronous
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	IDOC_CREMAS03_46C.xsc
	Instance Name	Cremas03
	Mode	Out
	Trigger	N/A
	Manual Publish	(checked)

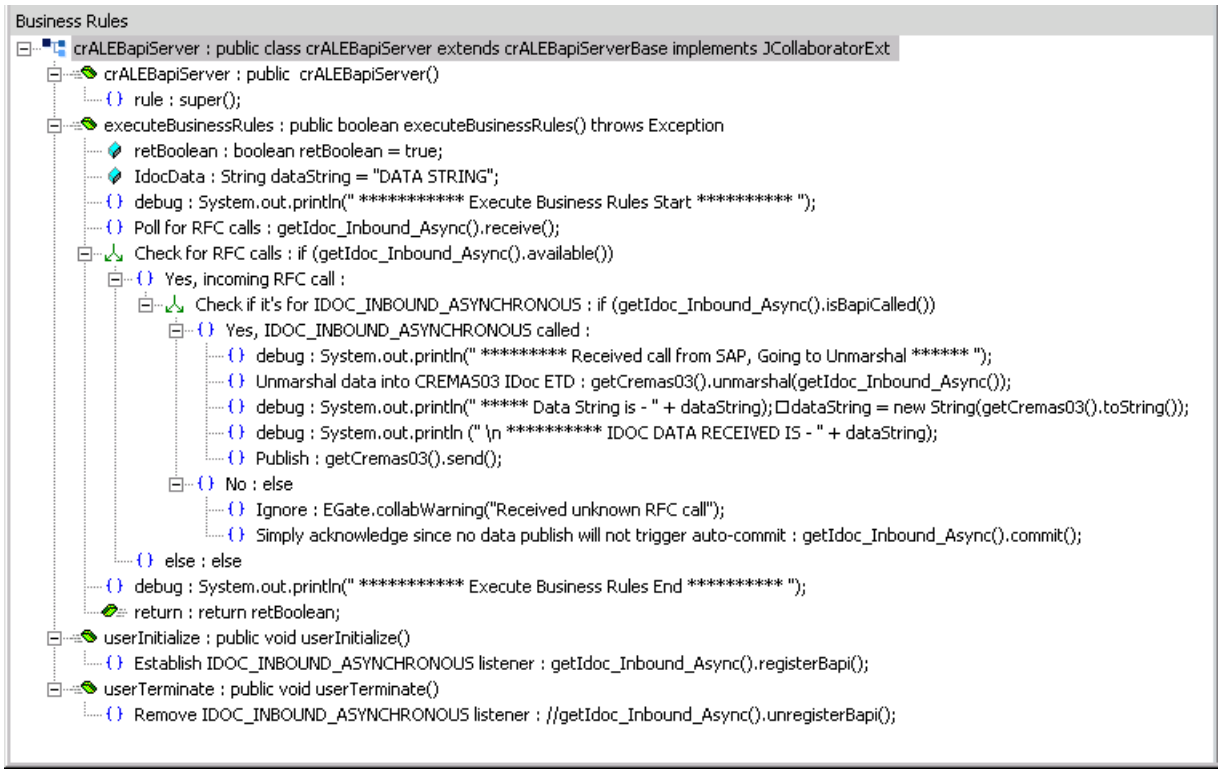
The mapping between the ETDs for **crALEBapiServer**, as displayed in the Collaboration Rules Editor, appear in Figure 73.

Figure 73 Collaboration Rule Map - crALEBapiServer



The Business Rules for **crALEBapiServer**, as displayed in the *Business Rules* pane of the Collaboration Rules Editor, appear in Figure 74.

Figure 74 Business Rules - crALEBapiServer



Business Rules Summary for crALEBapiServer

1. Under **userInitialize**, the **Establish listner** rule registers the BAPI e*Way with SAP R/3.
2. The **Poll for RFC calls** rule allows RFC calls to be received from SAP.
3. The first **if** rule checks for RFC calls.
4. The embedded **if** rule checks whether or not the RFC call involved the RFM IDOC_INBOUND_ASYNCHRONOUS.
5. The **Unmarshal** rule converts the data into a linear structure, allowing it to be copied to the single node CREMAS03 as a BLOB.
6. The **Publish** rule sends the IDoc CREMAS03 to **cpJMSServer**.
7. The **Establish** and **Remove** rules respectively establish and remove the IDOC_INBOUND_ASYNCHRONOUS listener, which detects the presence of the RFM IDOC_INBOUND_ASYNCHRONOUS containing an IDoc.
8. Under **userTerminate**, the **Remove listner** rule unregisters the BAPI e*Way with SAP R/3.
9. The **debug** rules simply add annotation to the log.

colEater

Table 26 Collaboration - colEater

Property		Name/Value
Collaboration Rule		crEater
Collaboration Service		Java Pass-through
Performing e*Way		ewEater
Source	ETD	GenericInEvent.ssc
	Instance Name	JavaPassThroughIn
	Mode	In
	Trigger	(checked)
	Manual Publish	N/A
Destination	ETD	GenericOutEvent.ssc
	Instance Name	JavaPassThroughOut
	Mode	Out
	Trigger	N/A
	Manual Publish	(cleared)

A Java Pass-through Collaboration Service uses a predefined Collaboration Rule, containing preset Business Rules, that transports data without transforming it.

5.5.5 Configuring the Components

e*Ways

The e*Ways are configured as described in **Configuring the e*Way** on page 146. The respective configuration parameters are listed in the following tables.

ewALEBapiServer

These Configuration Parameters retain their default values.

ewEater

Table 27 Configuration Parameters - ewEater

Section	Parameter	Value
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
	PerformanceTesting	(default)
Outbound (send) Settings	OutputDirectory	data\output
	OutputFileName	IDOCoutput%d.dat
	MultipleRecordsPerFile	YES (default)
	MaxRecordsPerFile	10000 (default)
	AddEOL	YES (default)
Poller (inbound) Settings	PollDirectory	data\input
	InputFileMask	*.fin (default)
	PollMilliseconds	1000 (default)
	RemoveEOL	YES (default)
	MultipleRecordsPerFile	YES
	MaxBytesPerLine	4096 (default)
	BytesPerLineIsFixed	NO (default)
	File Records Per eGate Event	1 (default)
Performance Testing	Performance Testing	100 (default)
	InboundDuplicates	1 (default)

e*Way Connections

Two e*Way Connections were created for the IDoc Server sample. To run this sample schema you must configure the e*Way Connection to match those of your specific system. See **Creating e*Way Connections** on page 154 for procedural information.

cpJMSServer

The e*Way Connection Type is **SeeBeyond JMS**. The parameter values you may need to enter are:

- Host Name (if you have changed it from *localhost*)
- Port Number (if necessary)

For more information on JMS e*Way Connection parameters see the *e*Gate Integrator User's Guide*.

cpC4XServer

The e*Way Connection Type is **SAP BAPI**. The parameter values you need to enter are:

- Application Server Hostname
- Router String (if needed)
- System Number
- Client
- User
- Password
- System ID

See **e*Way Connections** on page 37 for information on the SAP BAPI e*Way Connection Parameters.

5.5.6 Executing the Schema

After configuring all e*Way components, you can execute the IDoc Client sample schema by performing the following steps:

1. Start the schema using the following command:

```
stccb -ln localhost_cb -rh localhost -rs <schema name>-un
    Administrator -up <password>
```

2. From the Schema Designer, start the e*Ways:
   - **ewAleBapiServer**
   - **ewEater**.
3. If the **ewAleBapiServer** e*Way is properly connected, a message similar to the following appears in **ewAleBapiServer.log**:

```
12:12:34.849 EWY I 3744 (java_extensions.cxx:1073):
BapiConnector.open(): Successfully registered as a RFC server
    with:
    Program ID = sonic.EWYRECEIVE
    SAP Gateway = sapdev2
    Service = sapgw00
    SID = SBYN
```

   The connection can be tested by using SAP transaction **SM59** (see **Testing an RFC Connection** on page 86).
4. Send the IDoc from SAP R/3 using SAP transaction **BD14**, *Send Vendor (see* **Sending an IDoc** on page 94). The following now occurs:
   A. Data is sent from SAP to the BAPI e*Way **ewAleBapiServer**, which publishes the data to the JMSServer **cpJMSServer**.
   B. The File e*Way **ewEater** subscribes to the JMSServer and publishes the data to an external file.

Chapter 6

e*Way Setup

This chapter describes the procedures required to customize the Java e*Way Intelligent Adapter for SAP BAPI to operate within your production system.

6.1 Overview

After creating a schema, you must instantiate and configure the SAP BAPI e*Way 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 are:

6.2 Creating and Configuring e*Ways

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

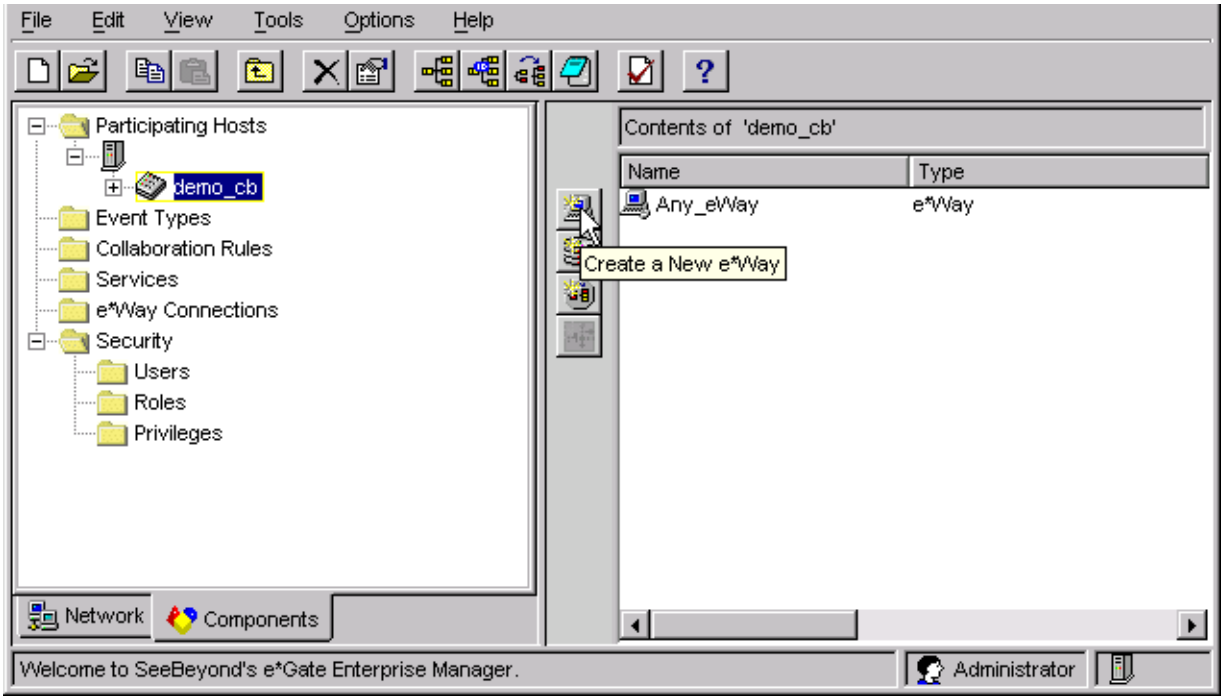
6.2.1 Creating the e*Way

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 75 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.

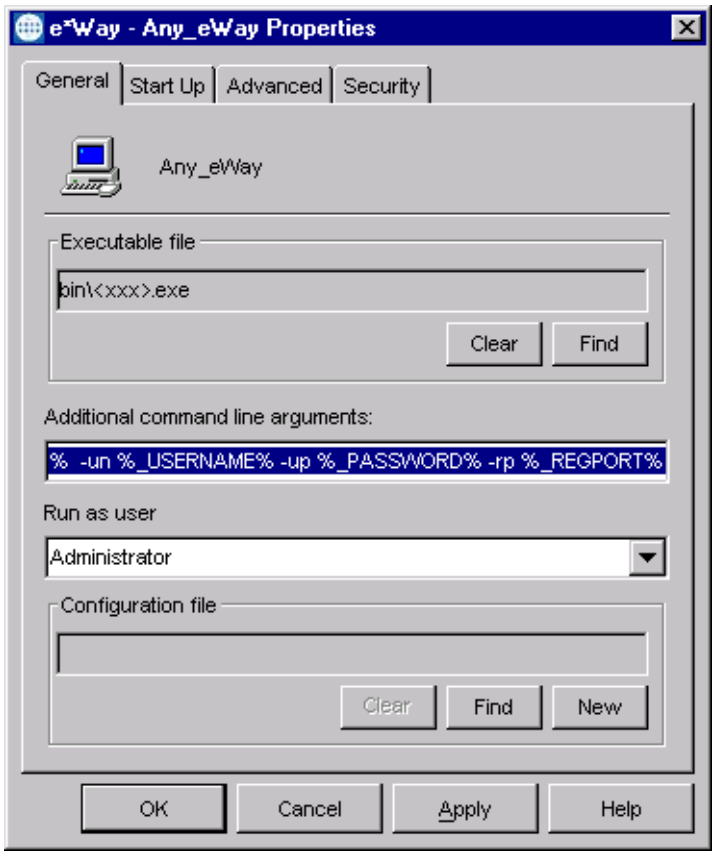
6.2.2 Modifying e*Way Properties

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 76).

Note: *The executable file is* **stceway.exe**.

Figure 76 e*Way Properties (General Tab)



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

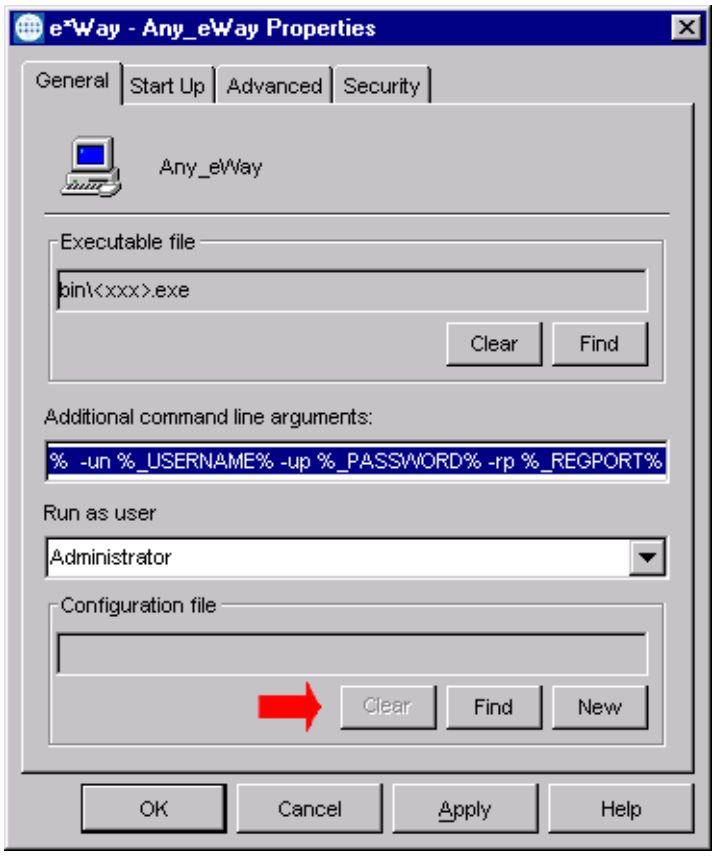
6.2.3 Configuring the e*Way

To change e*Way configuration parameters

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

Note: *The default configuration file is* **BapiConnector.def**.

Figure 77 e*Way Properties - General Tab



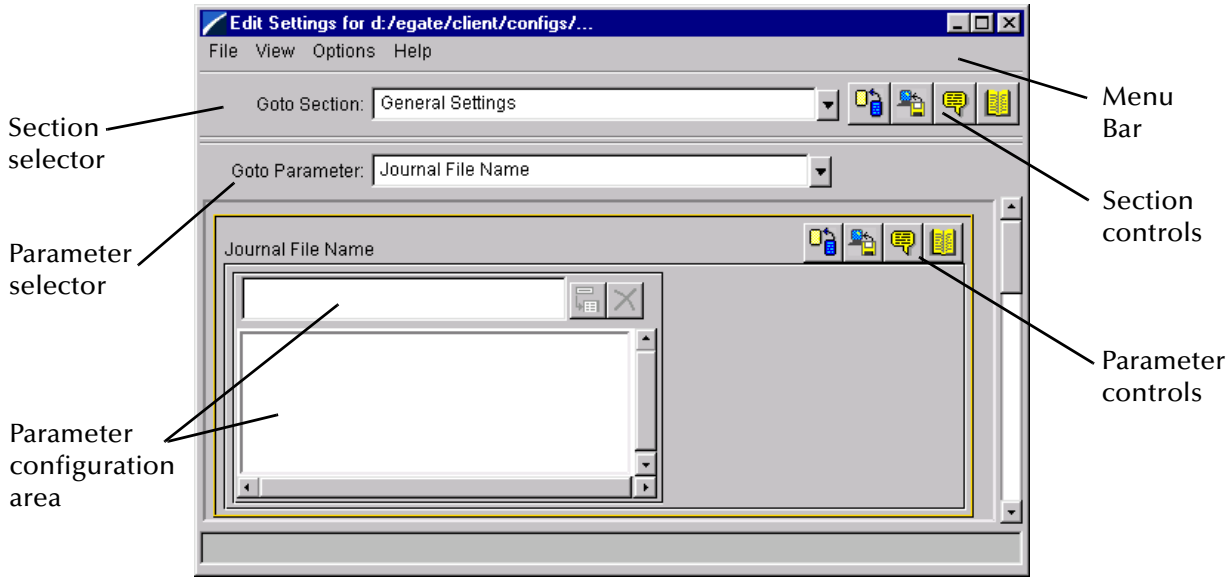
2 Under **Configuration File**, click **New** to create a new file or **Find** to select an existing configuration file. If you select an existing file, an **Edit** button appears; click the button to edit the currently selected file.

3 You now are in the e*Way Configuration Editor. The e*Way's configuration parameters are described in **Chapter 3**.

Using the e*Way Configuration 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 78 The e*Way Configuration Editor



The e*Way Configuration 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

Section and Parameter Controls

The section and parameter controls are shown in Table 28 below.

Table 28 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.*

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Option buttons
- Selection lists, which have controls as described in Table 29

Table 29 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.

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.

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 online Help system. 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*.

6.2.4 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.

6.2.5 Setting Startup Options or Schedules

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, the e*Gate Schema Manager.

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 79 on page 151**). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components. Also note the following special case.

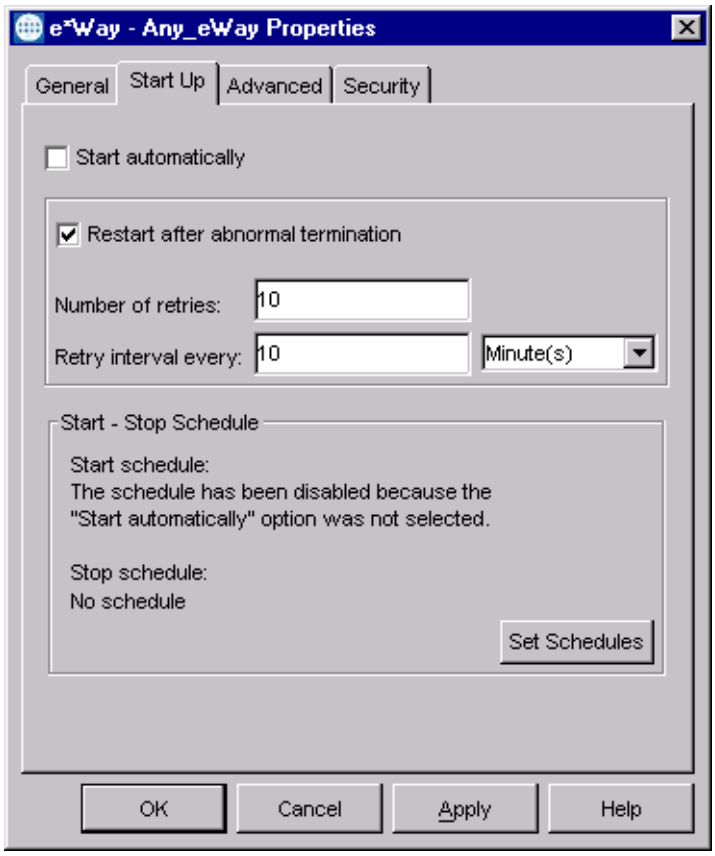
Server-Mode e*Way

When the e*Way starts up, it creates a JCo server connection (SAP Java Connector) using the supplied connection parameters. This is followed by a stabilization period of up to two minutes. If incorrect parameters are supplied, the server is created but the connection cannot stabilize. While the JCo server is attempting to stabilize the connection, the e*Way ignores any request to shut down; it will shut down, however, before attempting to reconnect.

Thus any shutdown request given to the e*Way via the Schema Manager will shut it down when:

- The JCo server has not yet been created to connect to SAP
- After the two-minute stabilizing interval, when the JCo server connection has failed to connect

Figure 79 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**.

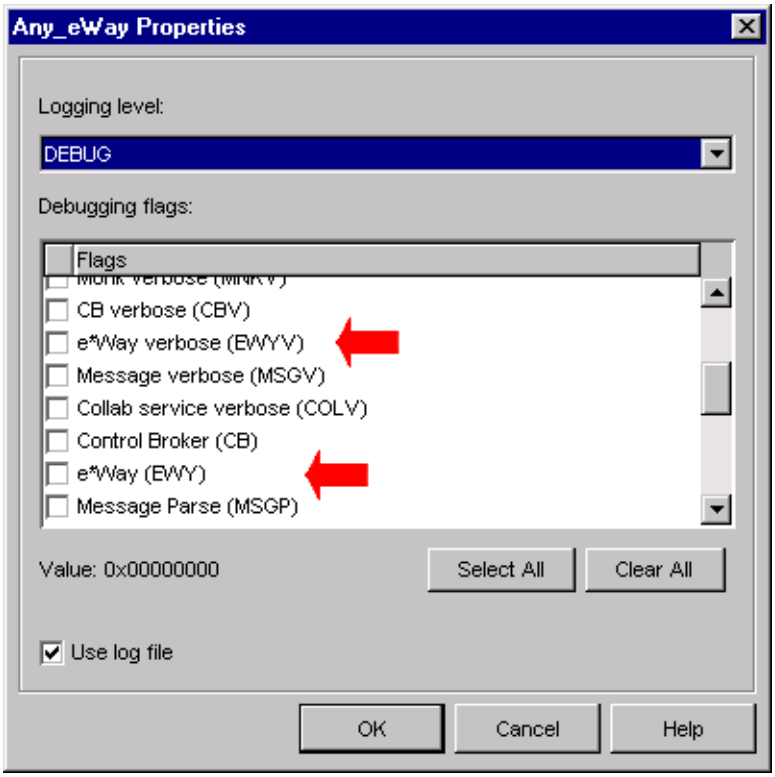
6.2.6 Activating or Modifying Logging Options

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**. The dialog window appears, as shown in Figure 80.

Figure 80 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 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.

6.2.7 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. Select 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.

6.3 Creating e*Way Connections

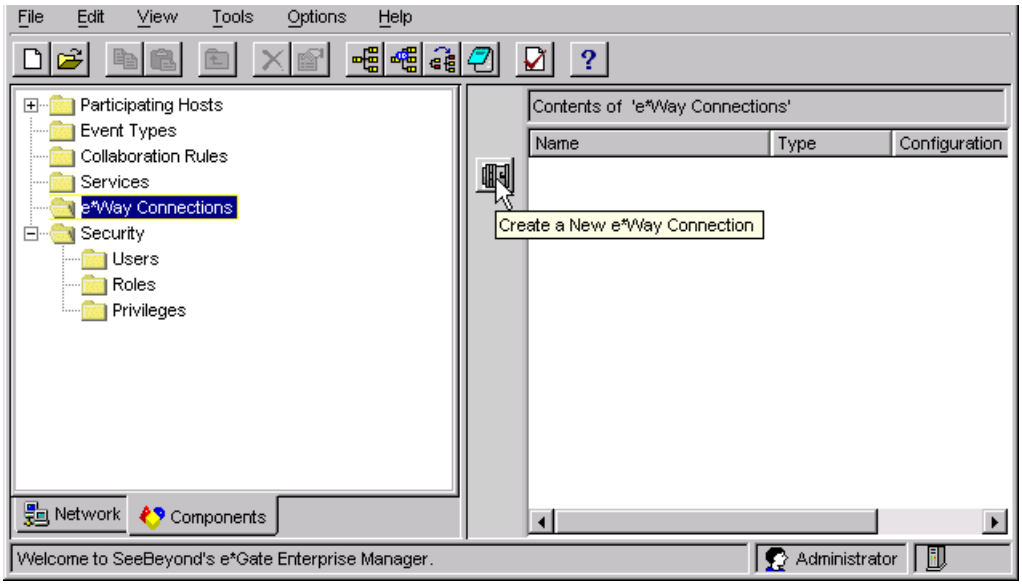
The e*Way Connections are created and configured in the Schema Designer.

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

To create and configure the e*Way Connections

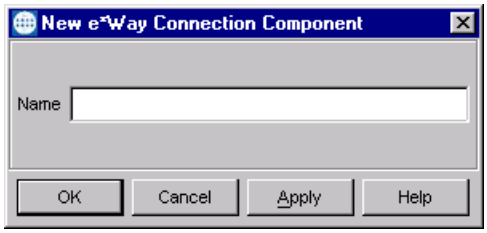
1 In the Schema Designer's Component editor, select the **e*Way Connections** folder.

Figure 81 Schema Designer - e*Way Connections Folder (1)



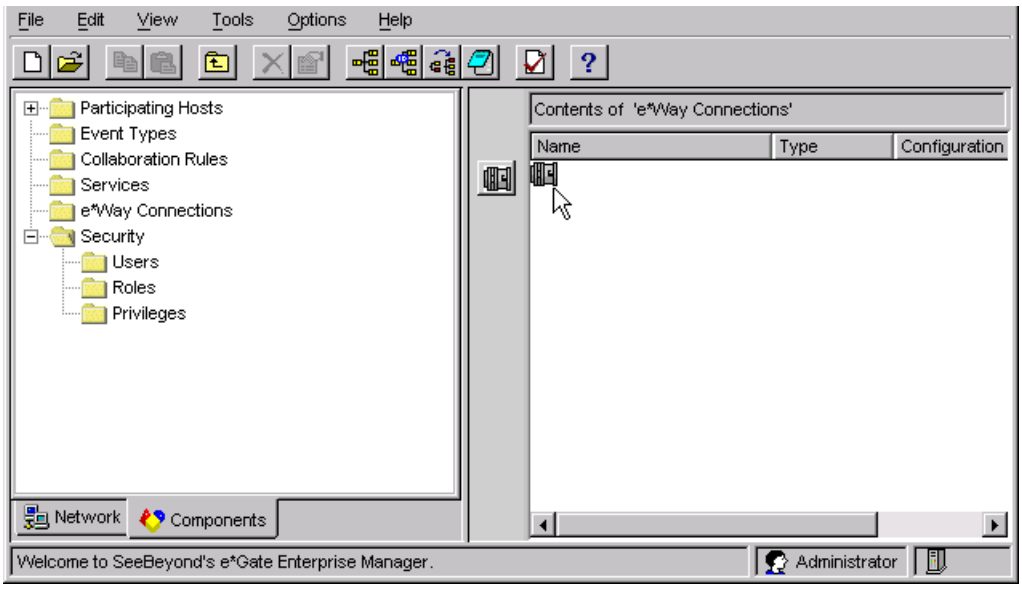
2 On the Palette, click the **Create a New e*Way Connection** button  , which opens the New e*Way Connection Component dialog box.

Figure 82 New e*Way Connection Component Dialog Box



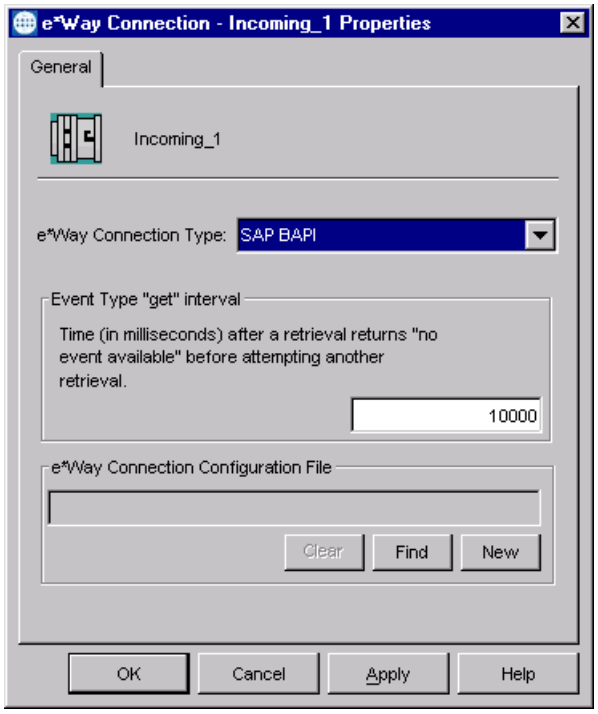
3 Enter a name for the e*Way Connection and click **OK**. The new e*Way Connection appears in the Schema Designer Contents pane.

Figure 83 Schema Designer - e*Way Connections Folder (2)



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

Figure 84 e*Way Connection Properties Dialog Box



5 From the e*Way Connection Type drop-down box, select **SAP BAPI**.

6 Enter the Event Type "get" interval in the dialog box provided (optional).

7 Click **New** to start the e*Way Connection Configuration File Editor, where you can create a new e*Way Connection Configuration File.

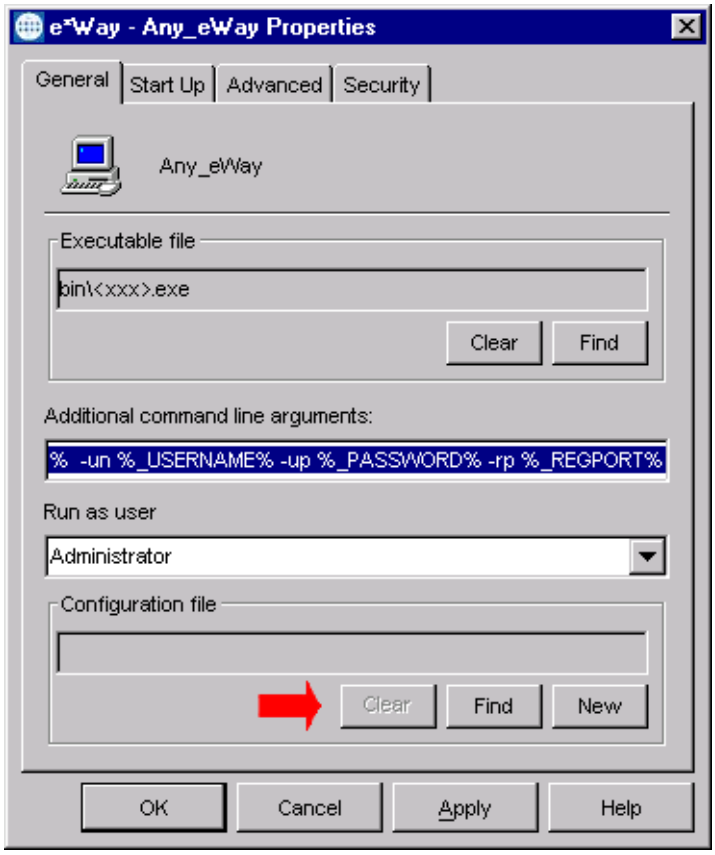
8 When you are finished, click the **File** menu, select **Save** and then **Promote to Run Time**.

To modify the e*Way Connections

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

Note: *The executable and default configuration files used by this e*Way are listed in* **Components** *on page 17.*

Figure 85 e*Way Properties - General Tab



2 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; click the button to edit the currently selected file.

3 You are now in the e*Way Configuration Editor. (see **Using the e*Way Configuration Editor** on page 147). The e*Way Connection's configuration parameters are described in **Chapter 3**.

4 From the **File** menu, select **Save** and then **Promote to Run Time**.

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

6.4 Managing e*Way Connections

The Connection Manager allows you to define the functionality of your e*Way connection. You can:

- Select when to initiate an e*Way connection
- Select when to close the e*Way connection and disconnect
- Select the status of your e*Way connection
- Have the Collaboration call the **On Connection Down** method when the connection fails

The Connection Manager is specifically designed to take full advantage of the enhanced functionality in e*Gate. In e*Gate 4.5.1 or earlier, this enhanced functionality is visible, but ignored.

The Connection Manager is controlled as described in **Connection Establishment Mode** on page 48. The Java methods available for the different connection establishment modes is shown in Table 30.

Table 30 e*Way Connection Controls

Java Method	Connection Establishment Mode		
	Automatic	**On-Demand**	**Manual**
onConnectionUp	yes	no	no
onConnectionDown	yes	only if the connection attempt fails	no
Automatic Transaction (XA)	yes	no	no
Manual Transaction (XA)	yes	no	no
connect	no	no	yes
isConnect	no	no	yes
disconnect	no	no	yes
timeout or connect	yes	yes	no
verify connection interval	yes	no	no

Note: *Connection Establishment Mode settings are ignored in e*Gate 4.5.1; the only mode available is the equivalent of* **Automatic**.

Controlling When a Connection is Made

By using Connector Settings, you may have e*Way connections controlled manually (through the Collaboration) or automatically (through the e*Way Connection Configuration). If you choose to control the connection manually, you can have the e*Way Connection made:

- When the Collaboration is loaded
- When the Collaboration is executed
- By using an additional connection method in the ETD
- By overriding any custom values you have assigned in the Collaboration
- By using the **is Connected**() method (if your ETD has multiple connections, the **is Connected**() method is called *per connection*)

Controlling When a Connection is Disconnected

In addition to controlling when a connection is made, you can also control when an e*Way Connection is terminated or disconnected either manually or automatically. You can have the e*Way Connection disconnect:

- At the end of a Collaboration
- At the end of the execution of the Collaborations Business Rules
- During a timeout
- After a method call

Controlling the Connectivity Status

You can control how often the e*Way Connection checks to verify is still live (see **Connection Verification Interval** on page 49).

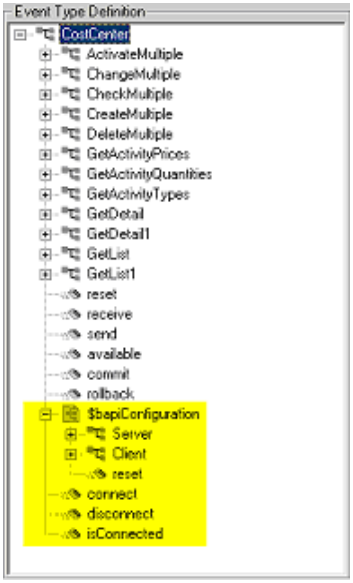
Connection State Trapping

When a BAPI e*Way Connection is lost or regained during use, the user can place code in the Collaboration's **on Connection Up()** and/or **on Connection Down()** methods to carry out special actions.

Using the ETD to access BAPI e*Way Connection Configuration

Figure 86 shows the node and available methods of the ETD.

Figure 86 Configuring Parameters Using the ETD



$bapiConfiguration node

- The **$bapiConfiguration** node gives access to all the configurable parameters for the Server and Client sections of the BAPI e*Way Connection configuration. These parameters get their initial values from the actual e*Way Connection configuration used.

Methods

- A **reset()** method is located under the **$bapiConfiguration** node, allowing you to reset all configurable parameters back to the initial set from the actual e*Way Connection configuration used.
- The **is Connected()** method verifies whether a Server and/or Client connection with SAP is established.

Note: *The following methods are valid only in Manual Connection Establishment mode.*

- The **connect()** method is used to establish a Server and/or Client connection to SAP, depending upon what connection parameters have been configured.
- The **disconnect()** method is used to end both Server and Client connection with SAP.

6.5 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.

6.5.1 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 provide information to this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components to which this e*Way sends information properly configured, and are they subscribing to the appropriate Events correctly?

In the e*Way Editor

- Check that all e*Way connection 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 the *path* environmental variable includes the location of the SAP BAPI dynamically-loaded libraries. The name of this variable on the different operating systems is:
  - PATH (Windows)
  - LD_LIBRARY_PATH (Solaris)
  - LIBPATH (AIX)
  - SHLIB_PATH (HP-UX)
- Check that your *classpath* environmental variable includes the location of **sapjco.jar**.

In the SAP Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.
- Especially important is that the program is configured correctly for Server-mode operation.

6.5.2 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 Schema Manager system to monitor operations and performance.

Chapter 7

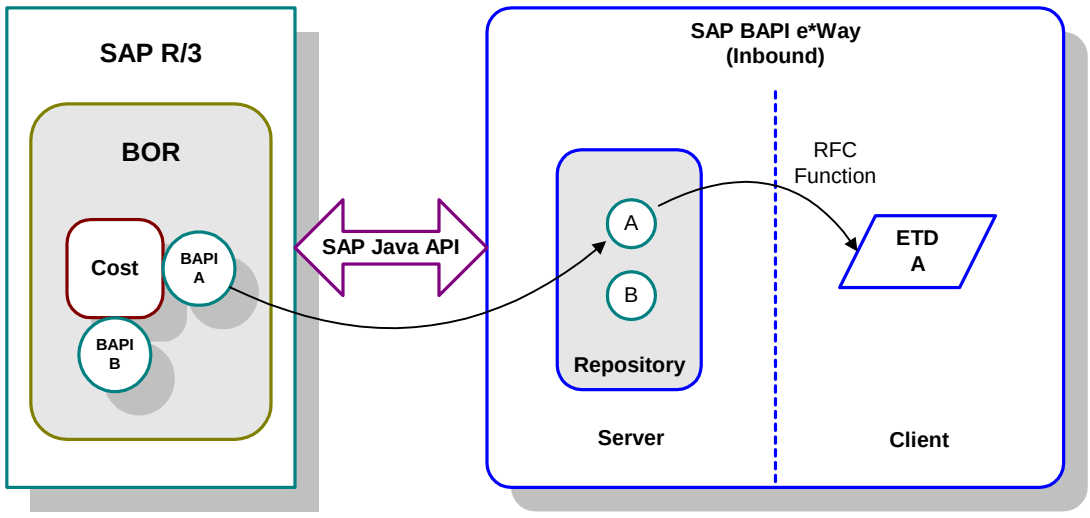
e*Way Operation

This chapter expands on the operational overview given in the Introduction, and describes the overall process of using the e*Way to interchange data with SAP R/3.

7.1 e*Way Overview

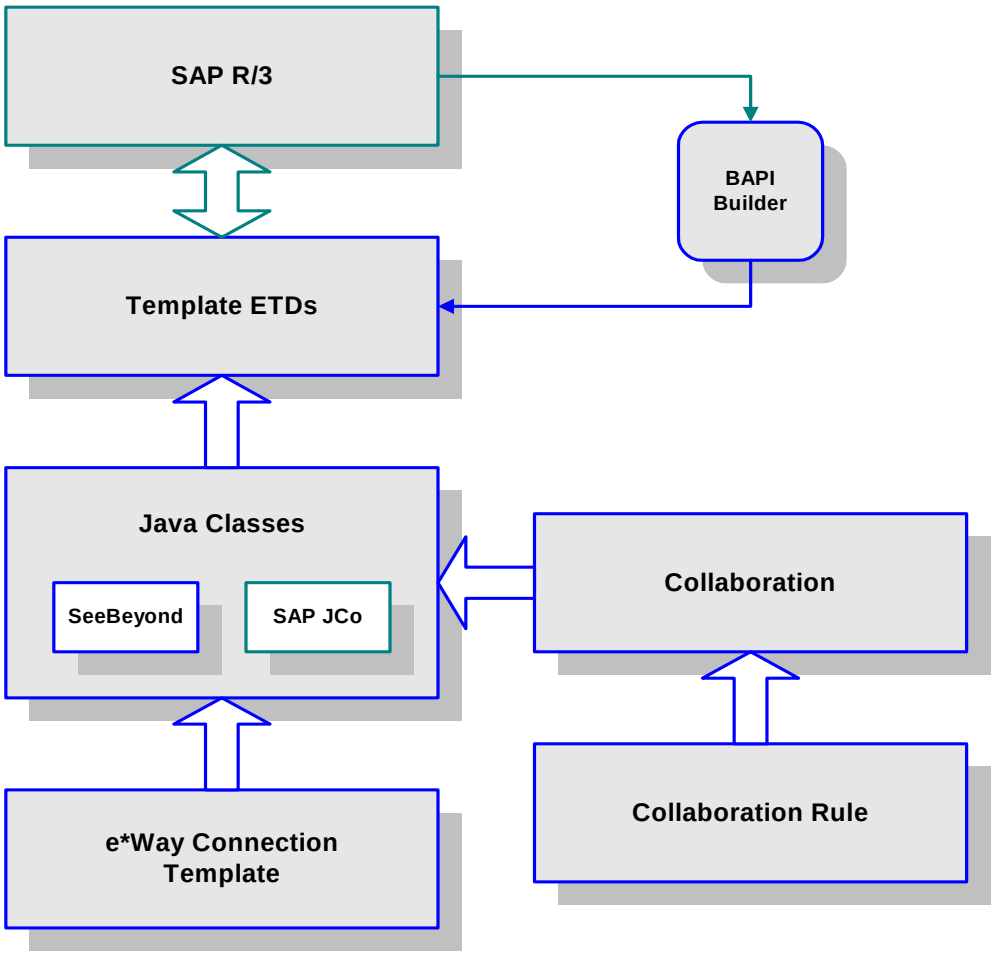
The SAP BAPI e*Way provides an interface with the SAP R/3 system by means of the SAP Java API set contained in the file **sapjco.jar**. Applicable BAPI methods are held in the e*Gate server's repository, and are invoked by an RFC call from the R/3 system. When invoked, they are passed as an RFC Function into an ETD.

Figure 87 BAPI e*Way



Several functional entities are involved in communicating with SAP R/3, as represented in Figure 88. These are described briefly, starting from the bottom. Additional information of a basic nature can be found in **e*Gate/e*Way Basics** on page 177.

Figure 88 BAPI e*Way Operation



e*Way Connection Template

This user-modifiable template, contained in **BapiConnector.def**, provides the system configuration parameters required to drive the Java classes for the connection.

Collaborations and Collaboration Rules

Collaborations implement the business logic, as specified by Collaboration Rules, required to implement the desired operation.

Java Classes

The Java classes execute the operations directed by the Collaborations, using the configuration parameters contained in the e*Way Connection template. These Java classes include both eBI Suite classes and those in the SAP JCo.

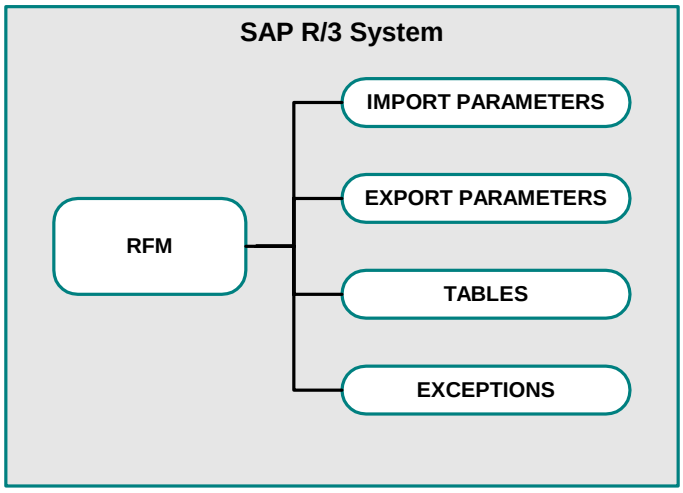
Template Event Type Definitions

These ETDs are used to invoke the desired interaction with SAP R/3. They are based on metadata obtained from SAP R/3, as explained in **Event Type Definitions** on page 181.

7.2 Remote Function Calls

A Remote Function Call (RFC) is SAP's term for a call to a function, or function module, in one system (the server) from another system (the client). An SAP Remote Function Module (RFM) has four main components, as depicted in Figure 89.

Figure 89 Remote Function Module



When making a call to a Remote Function Module, two different environments are involved (server and client). As a result, information is passed in a different manner than if the function call was being performed locally.

- **Import/Export Parameters**

  The actual values must be passed between systems, instead of being done by reference.

- **Tables**

  The entire contents of a table must be passed, creating a local copy on the server. After the RFC is completed, then the original table (on the client) is updated.

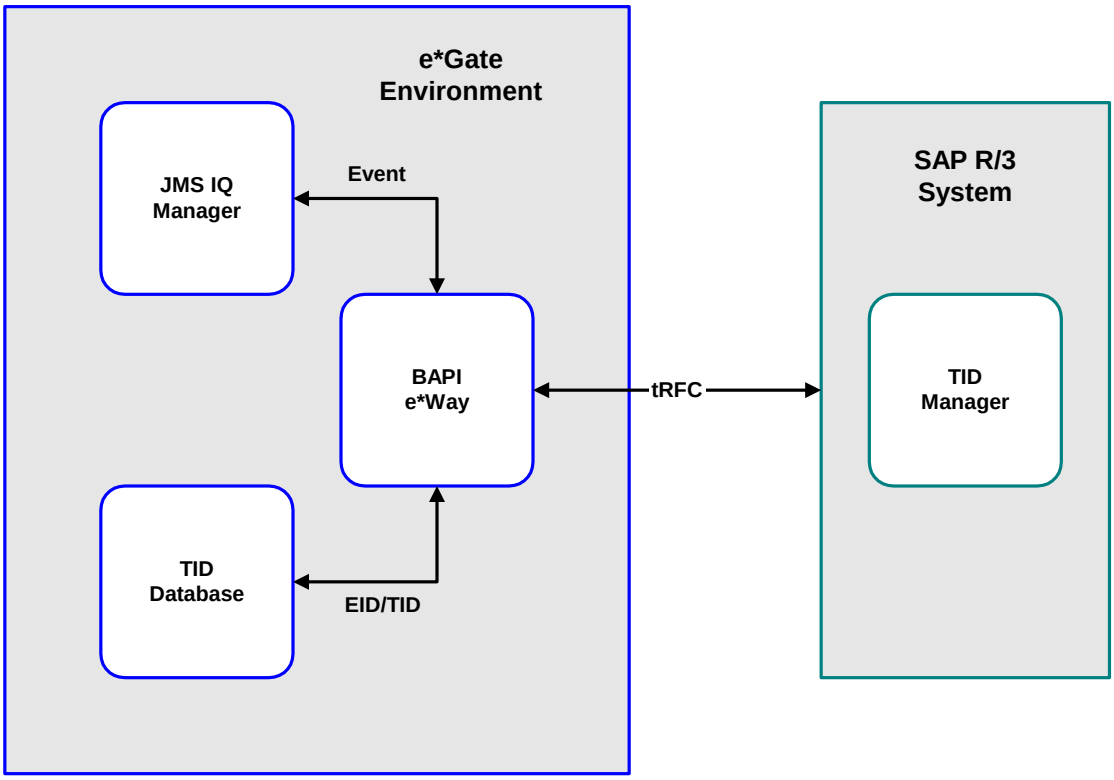
7.3 tRFC Process

Events can be sent to the SAP R/3 host via Transactional RFC (tRFC) or regular RFC. To use the tRFC mode, the e*Way Configuration parameter **Transactional Mode** must be enabled. Otherwise, the e*Way sends or receives the Event via regular RFC, which has been described previously.

With tRFC, the receiving SAP system relies on an unique Transactional ID (TID) sent with the message to ascertain whether or not a transaction has ever been processed by it before. This TID is assigned by the SAP R/3 system. Every Event received from this e*Way is checked against an internal TID database to ensure that it has not already been processed.

A receiving e*Way exhibits the same basic behavior for Events sent to it by the SAP system, checking the TID against a TID database. Internally, the Event is assigned an Event ID (EID) to track the Event within the e*Gate environment. When sending an Event to, or receiving an Event from, SAP R/3, there is an interchange process that occurs between EIDs and TIDs. The TIDs (assigned by SAP R/3) are stored in the e*Way's TID database, referenced to the associated EIDs (assigned by e*Gate).

Figure 90 tRFC Communications



7.4 XA Transaction Processing

e*Gate Integrator manages the publication and subscription of Events (messages) to and from JMS IQs (internal sources) and external systems. The customized interaction between sources—the Collaboration—is executed by the SAP BAPI e*Way using logic in the *Collaboration container* within the e*Way. The Collaboration container calls the user-created Collaboration.

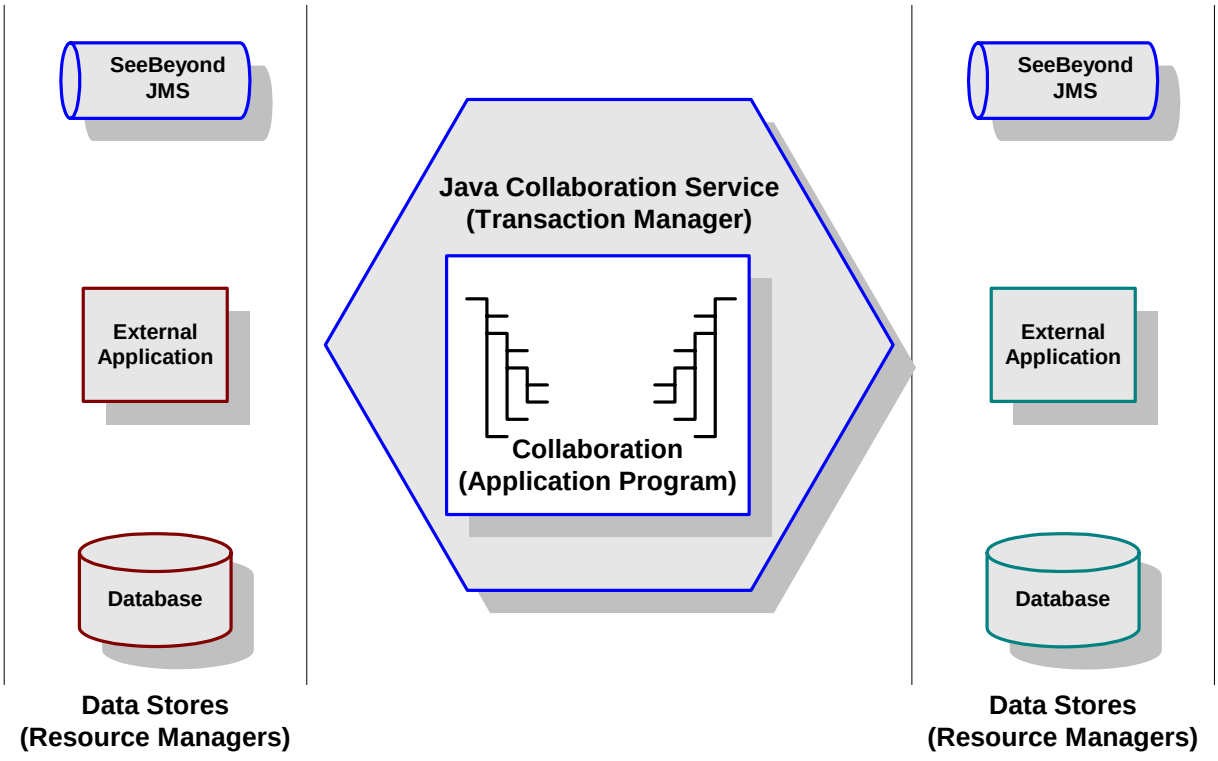
According to the X/Open specification of XA, Distributed Transaction Processing (DTP) systems are composed of three parties:

- Application Programs (AP) - User applications
- Resource Managers (RM) - Managers of the storage of data
- Transaction Managers (TM) - Coordinates activity across Resource Managers in order to guarantee the integrity of the data across multiple data storage systems

As depicted in Figure 91, the equivalent entities in e*Gate are, respectively:

- Collaborations
- IQs, external application and database systems
- Java Collaboration Service (JCS)

Figure 91 e*Gate-XA DTP Scenario



For additional information on e*Gate and XA, see the *e*Gate Integrator User's Guide*.

7.5 Client Mode (e*Gate to SAP)

In this mode, the e*Way typically receives data from the e*Gate system and sends it to the SAP R/3 system by calling a specific BAPI/RFC method. In return, the called BAPI method may provide some ensuing data to be sent back to the e*Gate system.

Before any BAPI methods on R/3 can be called, the e*Way has to log onto the R/3 system using pre-configured parameters (see **Client** on page 41).

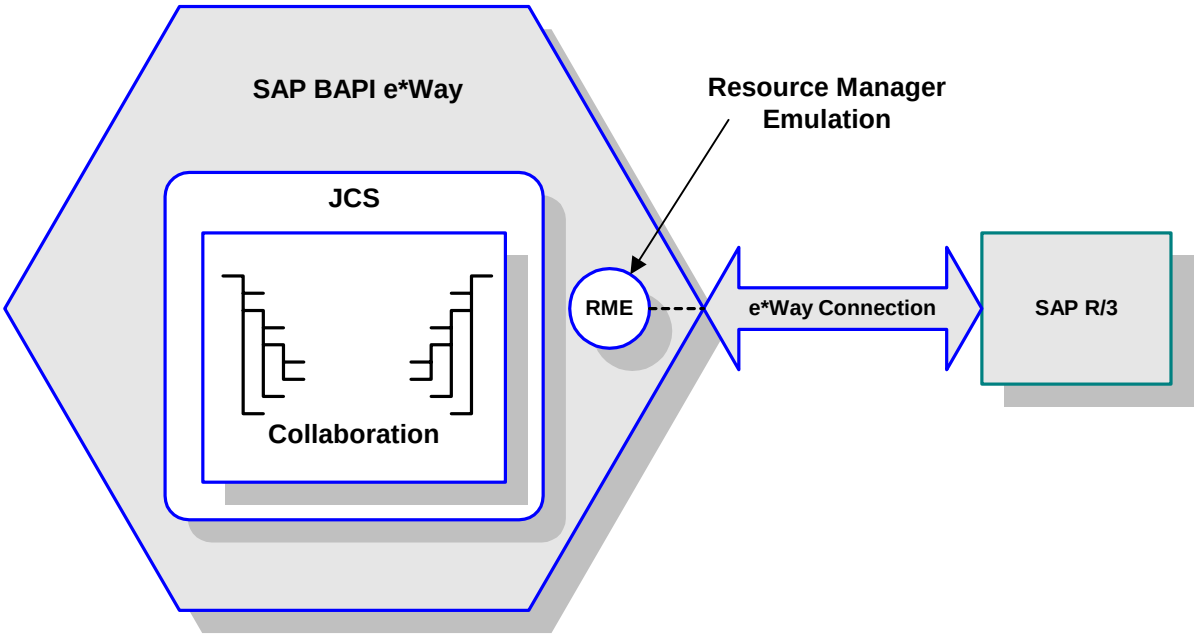
Events can be sent to SAP R/3 by means of four different mechanisms (listed in the order of preference):

- **XA-Compliant** (2-phase commit)
- **Transactional RFC (tRFC)**
- **Via COMMIT/ROLLBACK BAPI** (single-phase commit)
- **Non-Transactional**

7.5.1 XA-Compliant

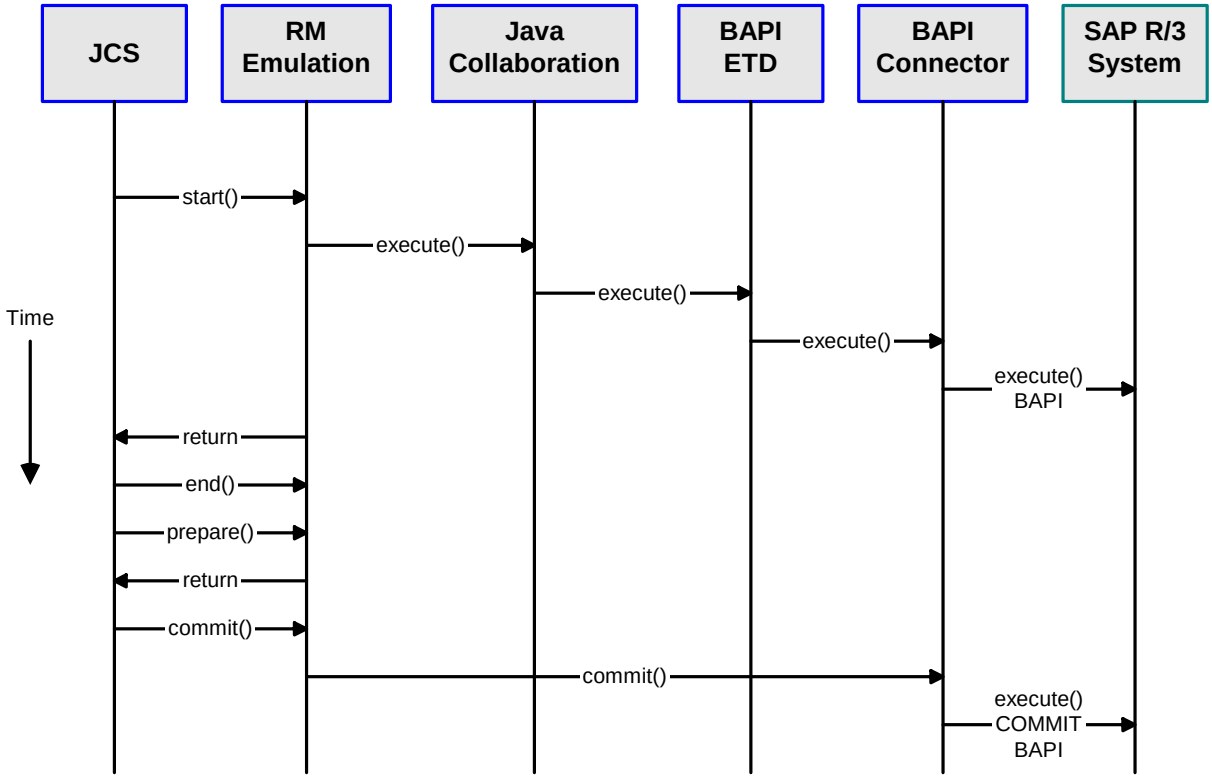
XA-compliance guarantees that related transactions processed by various XA-compliant resources are committed when all are successful; otherwise all transactions are rolled back if one or more resources fail. The SAP Java Connector API (JCo), however, is inherently non-XA-compliant. The SAP BAPI e*Way provides this compliance by emulating an XA Resource Manager (see Figure 92). This gives the SAP system the *appearance* of being XA-compliant.

Figure 92 SAP BAPI e*Way - XA Compliance



XA-compliant transaction processing employs a two-phase commit mechanism, which is depicted in Figure 93.

Figure 93 XA-Compliant Client Sequence



Client-mode XA-compliant Action Sequence

1. The JCS (acting as a Transaction Manager) starts the transaction demarcation in the emulated Resource Managers using the **start**() method.
2. When the RM has completed its task, it informs the JCS.
3. The JCS ends the transaction demarcation using the **end**() method.
4. The JCS prepares the RM to commit using the **prepare**() method.
5. If it is ready to commit, the RM returns.
6. The RM must return successfully for the JCS to call **commit**(); otherwise, the JCS calls **rollback**().

7.5.2 Transactional RFC (tRFC)

A subscribed Event is forwarded to the e*Way by the e*Gate Integrator system. The e*Way associates the next TID (assigned by the SAP R/3 system) with the outbound Event and prepares to send it via tRFC to SAP.

Using tRFC, the e*Way must first determine whether or not the EID has ever been processed by attempting to find an entry for the EID. There are three possible scenarios—the EID associated with the Event may or may not be found; and if it is, the TID may either be reserved or not used. Each scenario involves a different procedure, as outlined below and depicted in **Figure 94 on page 170**.

EID Not Found

This is the most straightforward case, in which the e*Way:

1. Obtains a new TID from the SAP system and *reserves* it, referencing it to the Event's EID.
2. Sends the Event to SAP by calling the appropriate ABAP function (on SAP), which then fetches the Event from the e*Way.
3. Marks the EID/TID pair as being *processed*.

EID Found but TID Not Used

In this case, the EID is found in the database, in which the e*Way:

1. Mark the TID as *reserved*.
2. Send the Event to SAP by calling the appropriate ABAP function (on SAP), which then fetches the Event from the e*Way.
3. Mark the EID/TID pair as being *processed*.

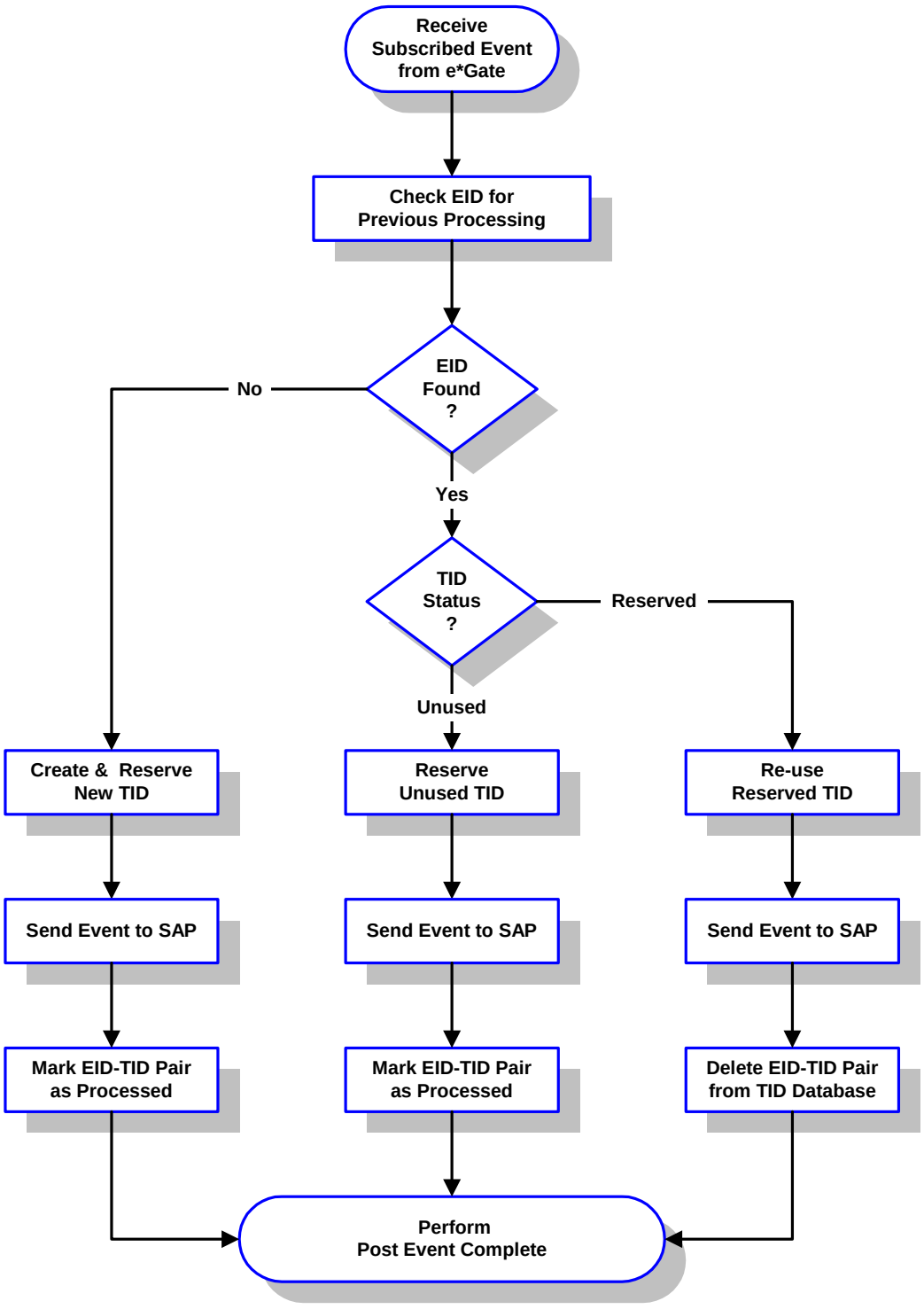
EID Found and TID Reserved

In this case, the EID is found in the database, and the associated TID is *reserved*, but not *processed*. The e*Way:

1. *Re-uses* the previously-reserved TID.
2. Sends the Event to SAP by calling the appropriate ABAP function (on SAP), which then fetches the Event from the e*Way.
3. Deletes the EID/TID pair from the TID database.

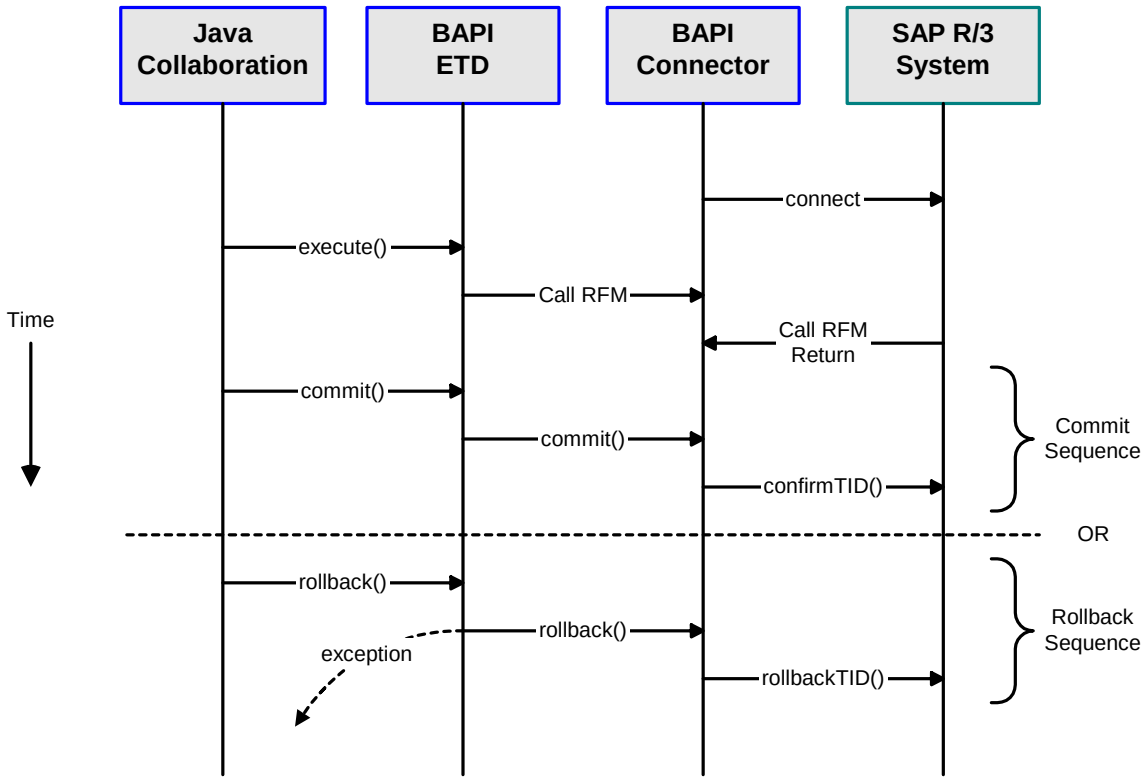
When the preceding process is complete, the e*Way performs a **Post Event Complete** to the e*Gate system, indicating that it is ready for the next Event.

Figure 94 tRFC Client Mode Process Flow



Transaction Processing

Figure 95 Transactional RFC Client Sequence

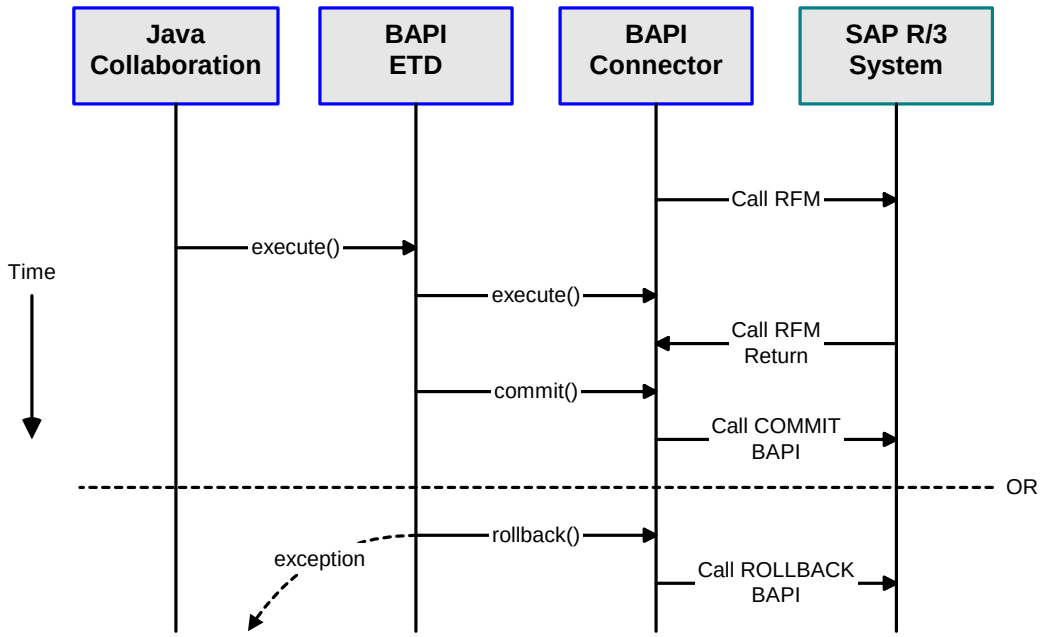


Client-mode XA-compliant Action Sequence

1. The e*Way populates the appropriate BAPI/RFC **Import** and/or **Table** parameter nodes with data from an Inbound ETD.
2. The e*Way then calls the BAPI ETD's **executeAsynchronous()** method, supplying it with the Event/message ID of the Inbound ETD.
3. Returns successfully.
4. When the Collaboration has successfully committed all read messages from the JMS Queue, the BAPI ETD's **commit()** method is called automatically. This, in turn, calls the **confirmTID()** method of the SAP TID Manager which informs the SAP system that the processed TIDs will never be used again.

7.5.3 Via COMMIT/ROLLBACK BAPI

Figure 96 Via COMMIT/ROLLBACK BAPI Client Sequence



Client-mode Via COMMIT/ROLLBACK BAPI Action Sequence

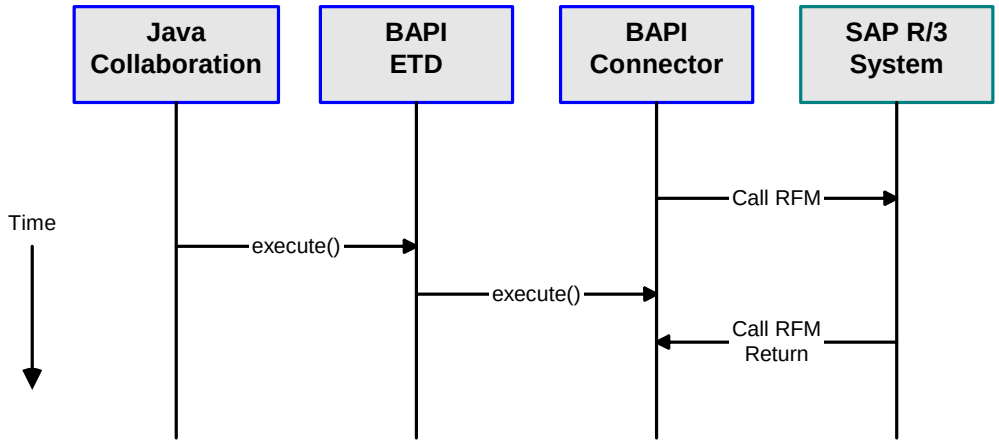
1. The e*Way populates the appropriate BAPI/RFC **Import** and/or **Table** parameter nodes with data from an Inbound ETD.
2. The e*Way then calls the BAPI ETD's **execute()** method.
3. SAP returns successfully.
4. The BAPI ETD's **commit()** or **rollback()** method is called which, in turn, calls the BAPI_TRANSACTION_COMMIT or BAPI_TRANSACTION_ROLLBACK BAPIs respectively, when the Connection Establishment Mode has been set to **Automatic**.

Note: *The ABAP/4 code in some BAPI/RFCs do not issue the COMMIT WORK instruction. These require a subsequent call using the same Client Session to the BAPI_TRANSACTION_COMMIT or BAPI_TRANSACTION_ROLLBACK BAPI to respectively instruct the system to COMMIT WORK or ROLLBACK WORK.*

7.5.4 Non-Transactional

Most BAPI/RFCs fall into this category. Once they have been successfully invoked, any consequent work done on the SAP system is immediately committed by SAP (autocommit). Therefore, it is not possible for the Java Collaboration Service to commit or rollback transactions involving these BAPI/RFCs.

Figure 97



Client-mode Non-Transactional Action Sequence

1. The e*Way populates the appropriate BAPI/RFC **Import** and/or **Table** parameter nodes with data from an Inbound ETD.
2. The e*Way then calls the BAPI ETD's **execute()** method.
3. SAP returns successfully.

7.6 Server Mode (SAP to e*Gate)

Before the e*Way can install functions on the SAP system, it must first register its Program ID. This Program ID is associated with an SAP RFC Destination, which should have been set up previously on SAP (transaction **SM59**).

After the e*Way is started, it registers itself with the SAP system using the pre-configured parameters (see **Server** on page 37). It then waits for RFM calls from SAP. When a call is received, the e*Way retrieves the data and sends the it to an e*Gate IQ. It also sends any required results back to SAP.

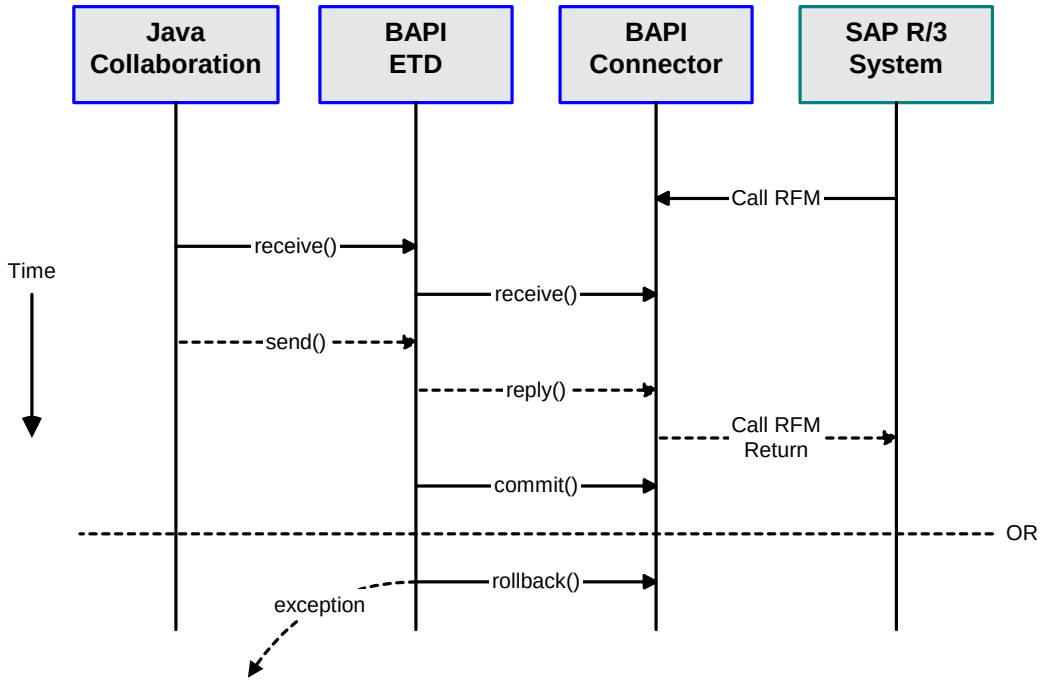
Events can be retrieved from SAP R/3 and responses, if any, can be returned to SAP by means of two different mechanisms:

- **Internal**
- **Transactional RFC (tRFC)**

7.6.1 Internal

Internal mode provides a simple RFC data transmission with a single-phase commit, as depicted in Figure 98.

Figure 98 Internal RFC Server Sequence



Server-mode Internal RFC Action Sequence

1 The Java Collaboration periodically calls the BAPI ETD's **receive**() method, in accordance with the **Wait for Request Interval** configuration parameter.

2 Finding that data from an RFM is available, the Collaboration accesses all pertinent data nodes and sends the gathered information to other e*Gate components.

3 If a response (other than a simple acknowledgment) needs to be returned, the Java Collaboration calls the BAPI ETD's **send**() method.

4 Receiving this notification, the e*Way Connection returns the RFM call to SAP.

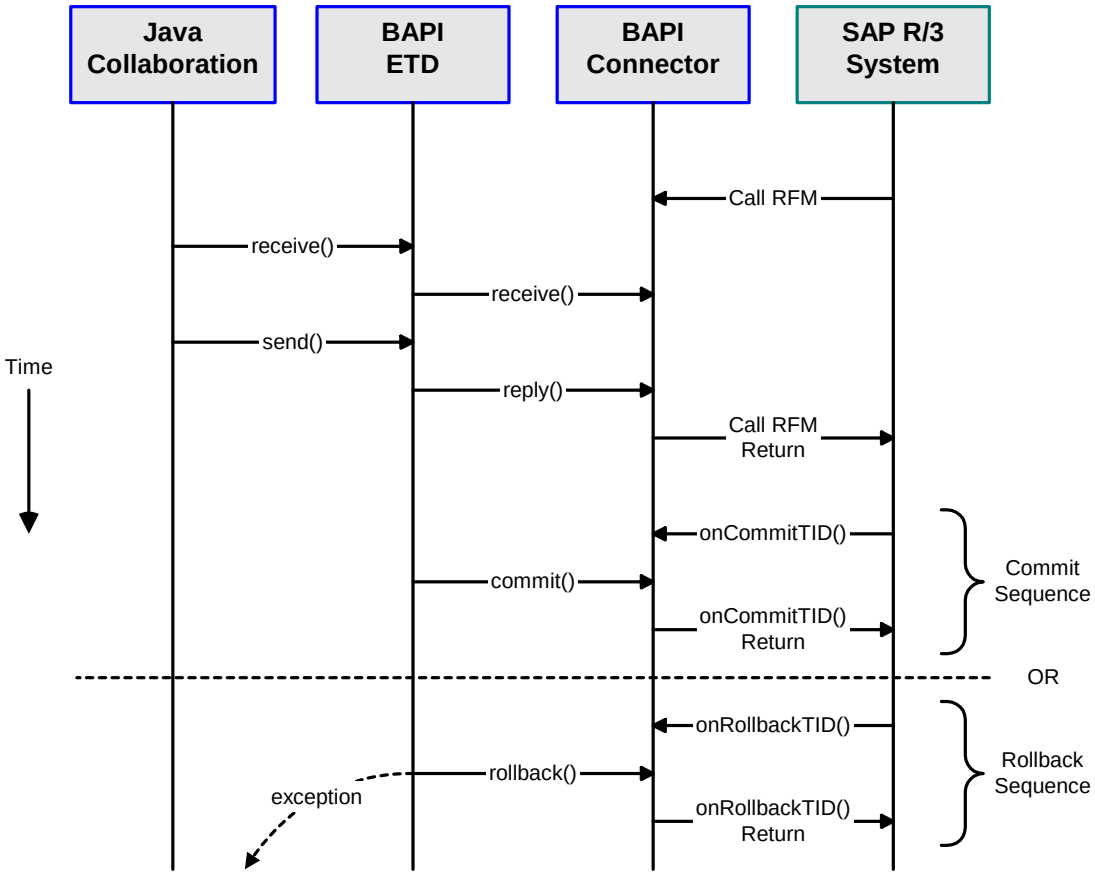
5 The BAPI ETD's **commit**() or **rollback**() method is called automatically, depending upon whether the data was posted successfully or rejected.

6 Note that the **disconnect()** method cannot be called until the SAP caller has been automatically acknowledged by means of the **commit()** method.

7.6.2 Transactional RFC (tRFC)

The Transactional RFC mode resembles the Internal mode, with the addition of transactional verification steps prior to committing or rolling back. This is the preferred method, when it can be used. By using unique TIDs associated with a BAPI/RFC call, the SAP system will process the data once, and only once. The sequence of the tRFC process is depicted in Figure 99. See **tRFC Process** on page 165 for additional information.

Figure 99 Transactional RFC Server Sequence



Server-mode tRFC Action Sequence

1. The Java Collaboration periodically calls the BAPI ETD's **receive**() method, in accordance with the **Wait for Request Interval** configuration parameter.
2. Finding that data from an RFM is available, the Collaboration accesses all pertinent data nodes and sends the gathered information to other e*Gate components.
3. The Java Collaboration calls the BAPI ETD's **send**() method, which results in a reply to the BAPI e*Way Connector.
4. Receiving this notification, the e*Way Connection returns the RFM call to SAP.
5. If the RFM call returned successfully, without exceptions being thrown, SAP informs the e*Way that the data can be committed by calling **onCommitTID**().
6. The BAPI ETD's **commit**() method is then called, which updates the TID in the file database as being **Committed**, commits the data, and sends an **onCommitTID**() return to SAP.
7. If the RFM call did not return successfully, SAP informs the e*Way that the data must be rolled back by calling **onRollbackTID**().
8. The BAPI ETD's **rollback**() method is then called, which rolls back the data and throws an exception. It also sends an **onRollbackTID**() return to SAP, confirming that the data was rolled back and not committed.

7.6.3 e*Way Startup and Shutdown

When the e*Way starts up, it creates a JCo server connection (SAP Java Connector) using the supplied connection parameters. This is followed by a stabilization period of up to two minutes. If incorrect parameters are supplied, the server is created but the connection cannot stabilize. While the JCo server is attempting to stabilize the connection, the e*Way ignores any request to shut down; it will shut down, however, before attempting to reconnect.

Thus any shutdown request given to the e*Way via the e*Gate Schema Manager will shut it down when:

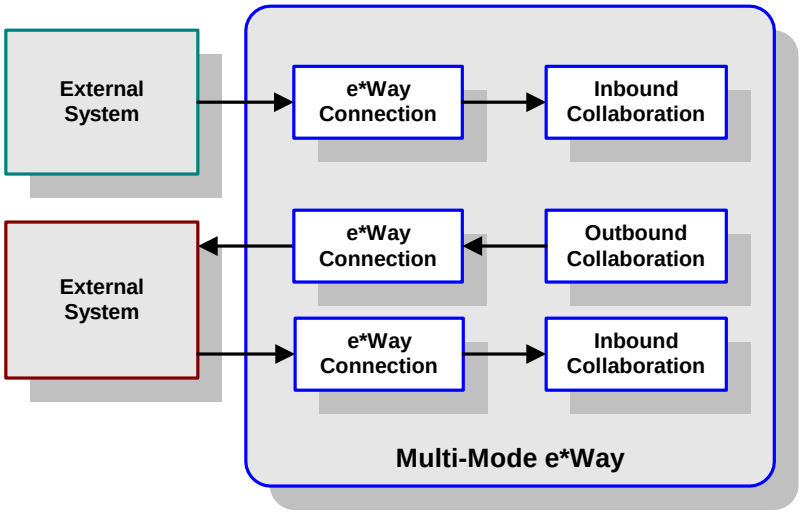
- The JCo server has not yet been created to connect to SAP
- After the two-minute stabilizing interval, when the JCo server connection has failed to connect

7.7 e*Gate/e*Way Basics

7.7.1 Multi-Mode e*Way

The SAP BAPI e*Way is based on the 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 100).

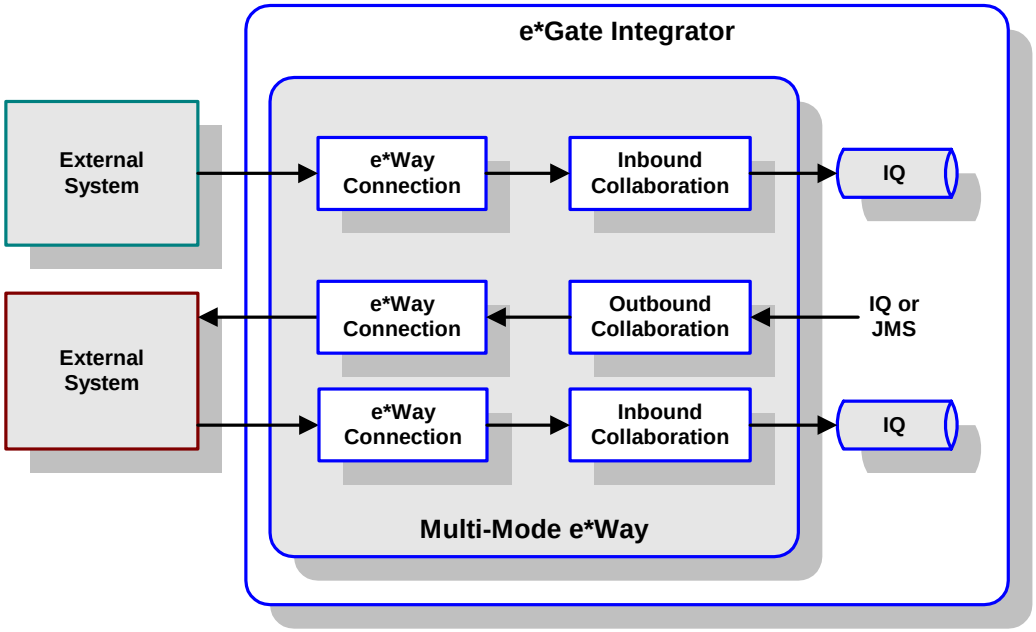
Figure 100 Multi-Mode e*Way



Each e*Way performs one or more *Collaborations* (see **Collaborations** on page 179). Bidirectional data flow requires at least two Collaborations, one *Inbound* and one *Outbound*, as shown in Figure 100. Each Collaboration processes a stream of messages, or *Events*, containing data or other information.

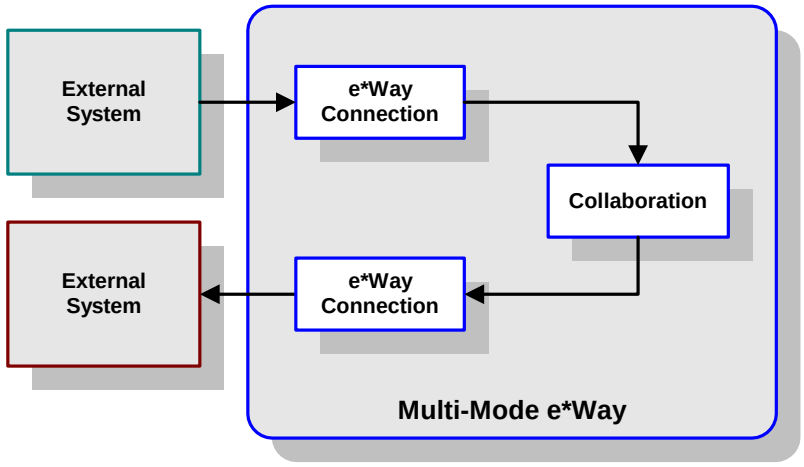
Each Collaboration that publishes its processed Events internally (within e*Gate Integrator) requires one or more *Intelligent Queues* (IQs) to receive the Events (see Figure 101). Any Collaboration that publishes its processed Events only to an external system *does not* require an IQ to receive Events.

Figure 101 e*Way within e*Gate Integrator



Although usually implemented as above, this e*Way also can be implemented as a stand-alone bridge between two or more external systems (see Figure 102).

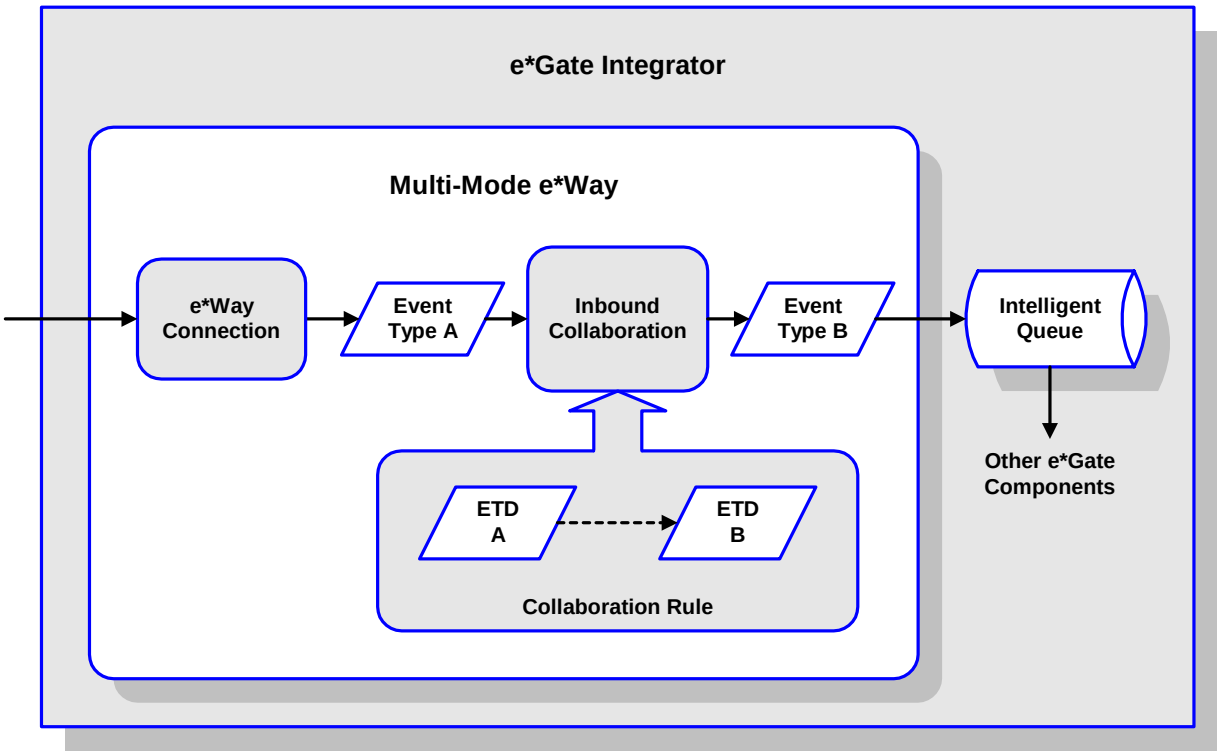
Figure 102 Stand-alone e*Way



7.7.2 Collaborations

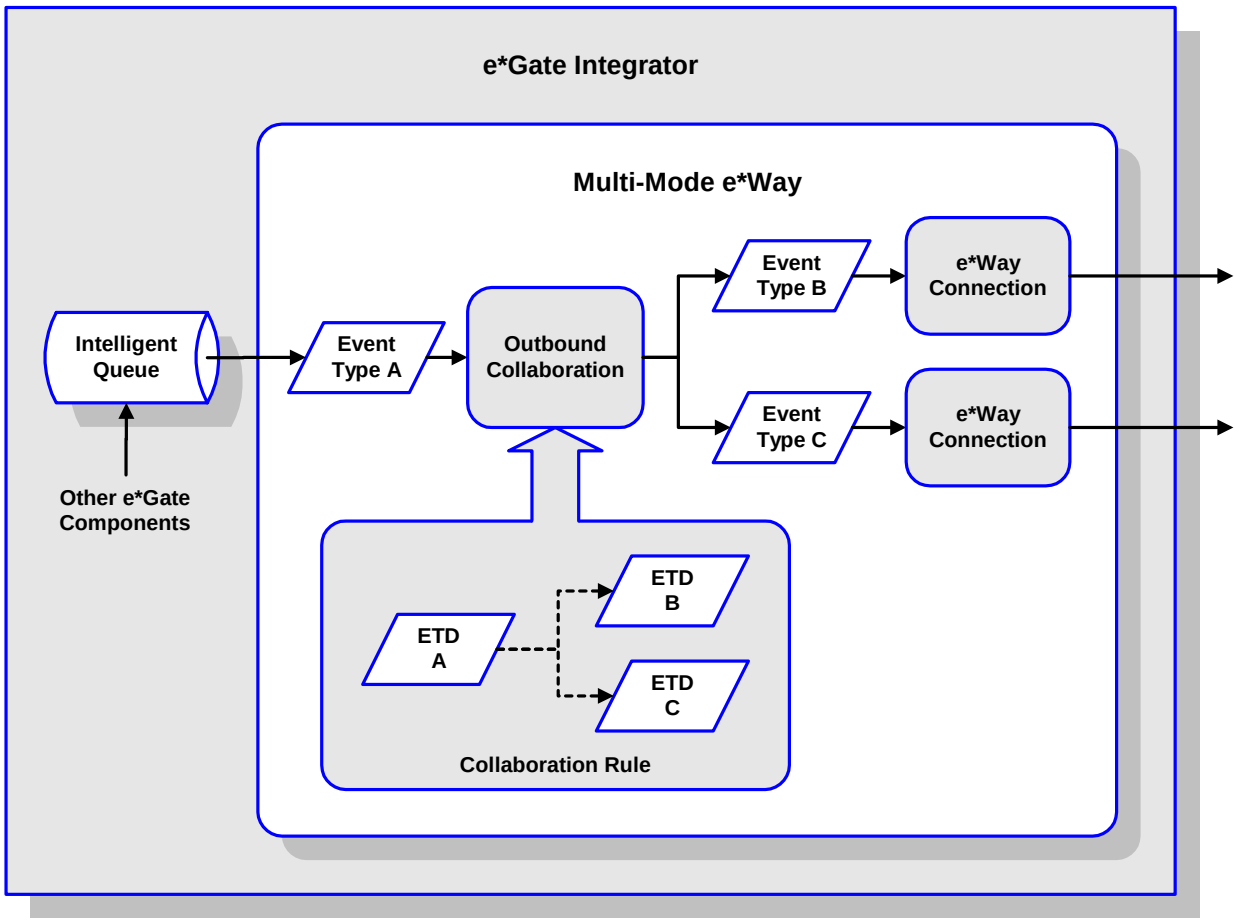
Collaborations execute the business logic that enable the e*Way to perform its intended task. Each Collaboration executes a specified *Collaboration Rule*, which contains the actual instructions to execute the business logic and specifies the applicable *Event Type Definitions* (ETDs). Events Types represent *instances* of their corresponding ETDs. A look inside a typical inbound Collaboration is shown in Figure 103.

Figure 103 Inbound Collaboration



A corresponding look inside a typical outbound Collaboration is shown in Figure 104. 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 104 Outbound Collaboration



7.7.3 Subcollaboration Rules

Java methods are also provided to allow you to use a Collaboration Rule as a parent or child of another Collaboration Rule. This allows you to insulate connectivity from transformation by separating out transformation-specific Collaboration Rules and maintaining them as individual units. See the *e*Gate Integrator User's Guide* for detailed information on Subcollaborations.

Note: *Subcollaboration support is not available on e*Gate 4.5.1.*

7.7.4 Event Type Definitions

ETDs can be categorized in two types:

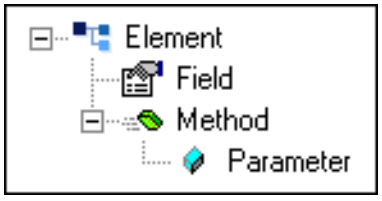
- *Data* ETDs, which represent the structure of Events (messages)
- *API-based* ETDs, or *Complex* ETDs, primarily contain APIs for communicating with external systems (the BAPI ETD is of this type)

Transforming data from one format to another is a major part of the processing performed by the e*Way, and data ETDs represent the data structure required by specific external systems. Generally, these ETDs are referred to as being *marshalable* because they can be *marshaled* into a flat, non hierarchical structure. See the *e*Gate Integrator User's Guide* for an extensive explanation of ETDs.

ETD Components

There are four possible components (nodes) in the Java Event Type Definition as shown in Figure 105.

Figure 105 The Java-enabled ETD

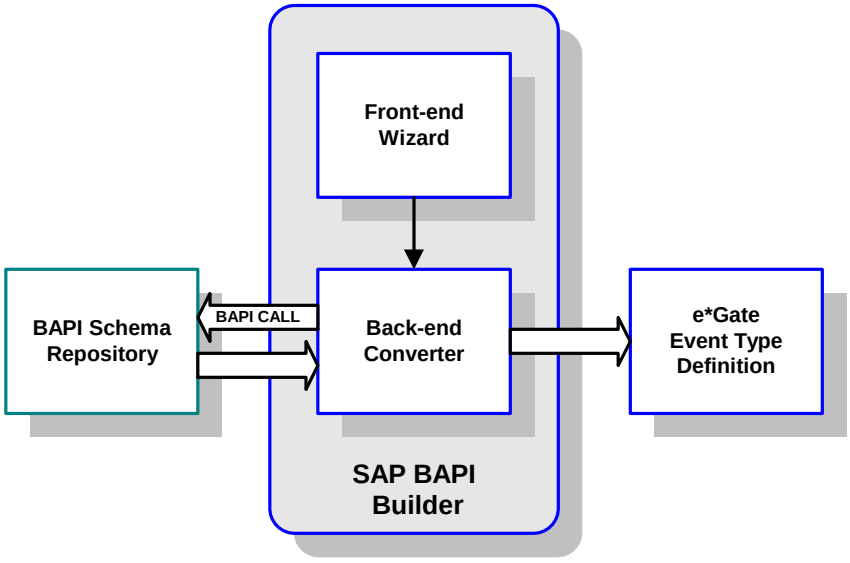


- *Elements* are the basic containers that holds the other parts of the ETD, and can contain fields and methods
- *Fields* are used to represent data, which can be in any of the following formats:
  - string
  - boolean
  - integer
  - double
  - float
- *Method* nodes represent actual Java methods
- *Parameter* nodes represent the Java method's parameters

ETD Builders

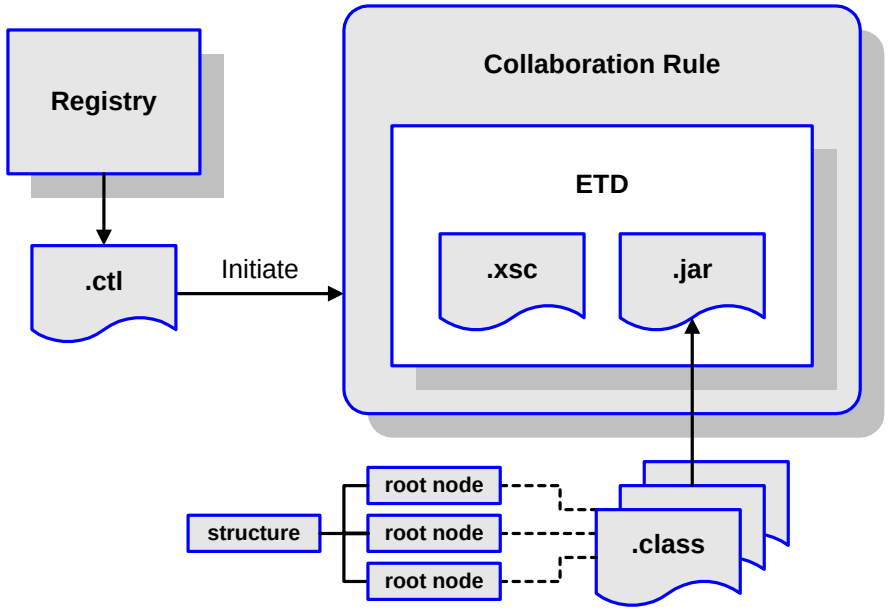
Building an ETD obviously requires knowledge of the internal data structure of the specific application. This information is obtained by extracting metadata from SAP R/3, which is automated by using the *SAP BAPI Wizard (see* Figure 106).

Figure 106 SAP BAPI Wizard



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 107).

Figure 107 Event Type Definitions

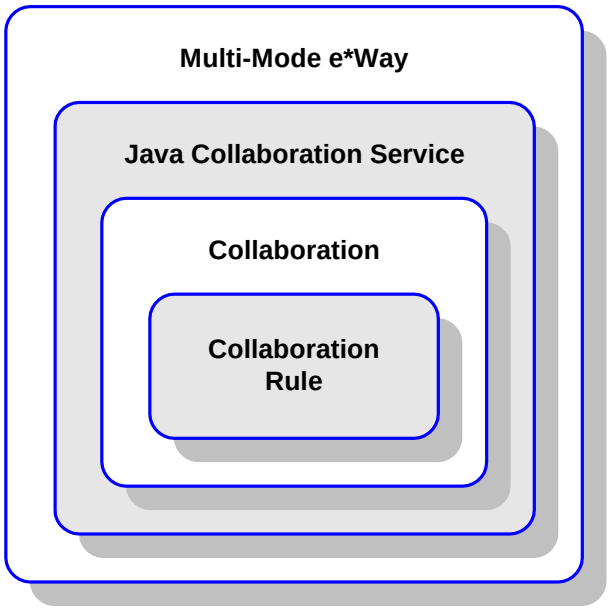


7.7.5 Java Collaboration Service

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 108.

Figure 108 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*.

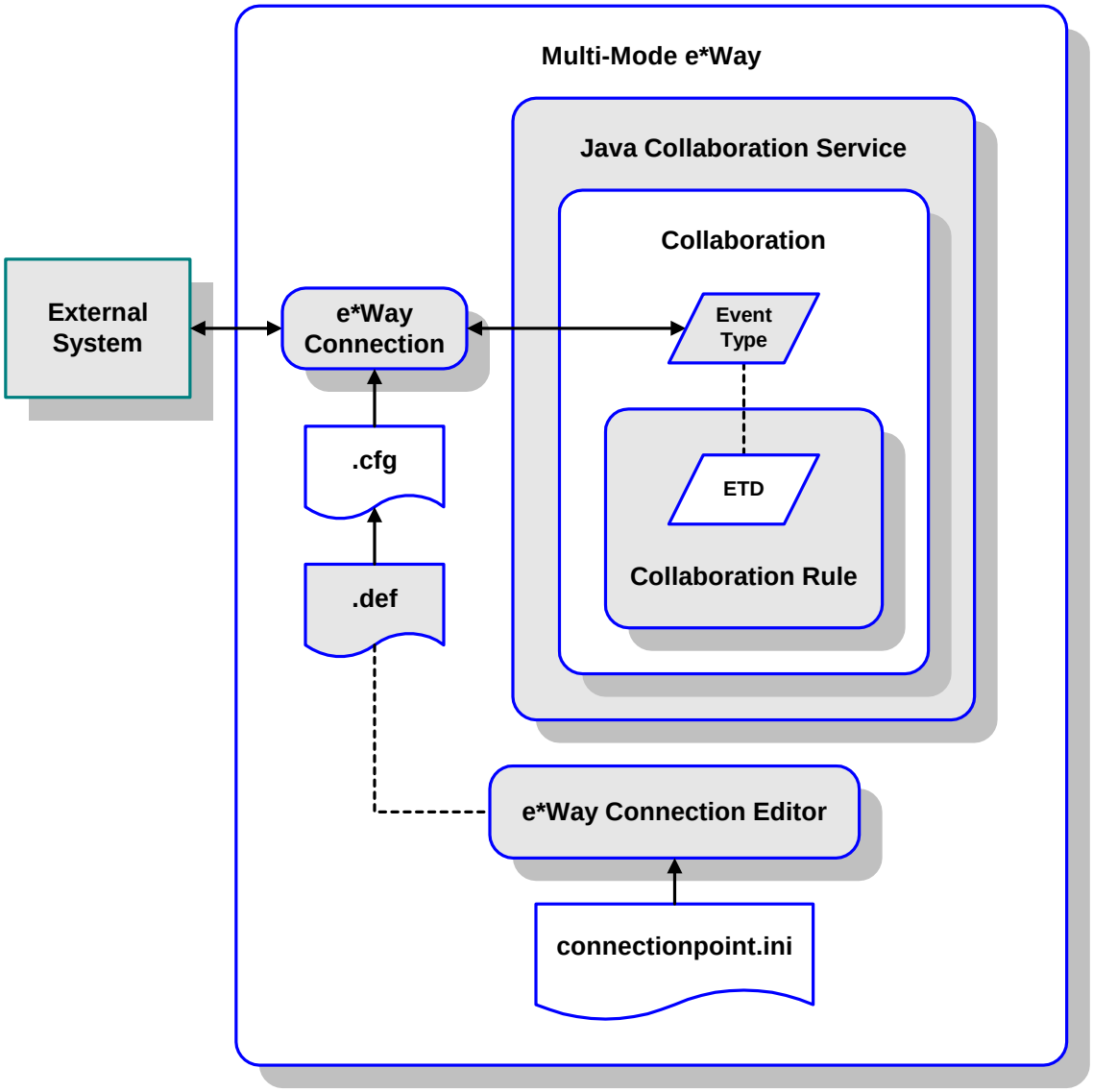
7.7.6 e*Way Connections

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.

Establishing Connections

An e*Way Connection to an external application is set up as depicted in Figure 109. 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 109 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*.

Chapter 8

Java Classes and Methods

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

8.1 Overview

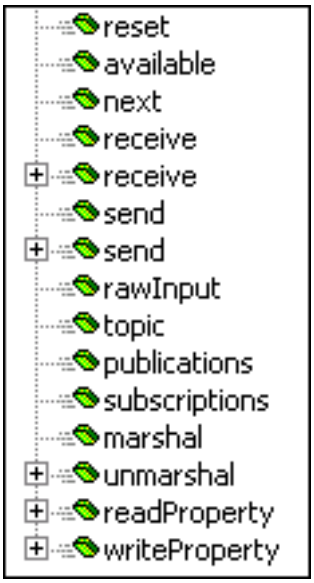
This chapter contains descriptions of methods that are exposed in the user interface. Additional methods contained in the e*Way should only be accessed or modified by qualified Oracle personnel. Unless otherwise noted, all classes and methods described in this chapter are **public**.

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

8.2 Basic ETD Methods

Every Standard ETD is automatically given a basic set of methods such as **available()**, **next()**, and **send()**, to allow the Collaboration to work with the ETD in its entirety; for example, to marshal the data in the next Event of the current Event Type. For complete information on these methods, see the *e*Gate Integrator User's Guide*.

Figure 110 Basic ETD Method Set



Additional basic methods, specific to the SAP BAPI e*Way, are described below.

Figure 111 Additional BAPI ETD Methods

available

Description

This method checks to see whether or not the ETD has input data, returns **true** if data exists.

Signature

```
available()
```

Parameters

None.

Return Type

boolean

Throws

None.

execute

Description

This method synchronously executes (calls) a BAPI/RFC method on the SAP R/3 Application Server.

Signature

```
execute()
```

Parameters

None.

Return Type

void

Throws

None.

reset

Description

This method resets BAPI data content, returns **true** if successful.

Signature

```
reset()
```

Parameters

None.

Return Type

boolean

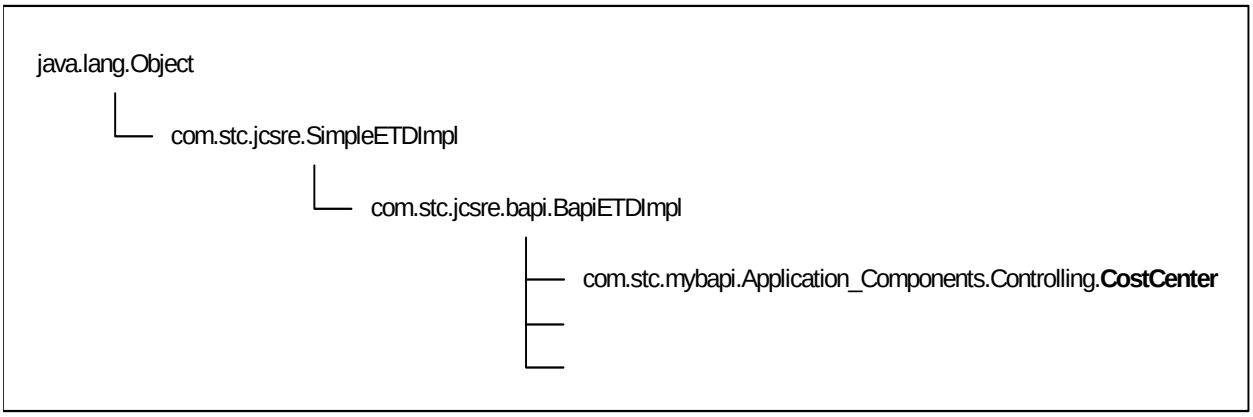
Throws

None.

8.3 CostCenter Class

A class representing the **CostCenter** BAPI defined in the SAP R/3 BOR. This class implements **com.stc.jcsre.bapi.Bapi.ETD**.

Figure 112 CostCenter Class Hierarchy



Definition

```
public class CostCenter
```

Methods

8.3.1 Methods

getActivateMuliple

Description

Gets and returns a reference to the object representing the BAPI method **ActivateMultiple**.

Signature

```
getActiveMultiple()
```

Parameters

None.

Return Type

CostCenter.ActivateMultiple

Throws

None.

getChangeMultiple

Description

Gets and returns a reference to the object representing the BAPI method **ChangeMultiple**.

Signature

```
getChangeMultiple()
```

Parameters

None.

Return Type

CostCenter.ChangeMultiple

Throws

None.

getCheckMultiple

Description

Gets and returns a reference to the object representing the BAPI method **CheckMultiple**.

Signature

```
public getCheckMultiple()
```

Parameters

None.

Return Type

CostCenter.CheckMultiple

Throws

None.

getCreateMultiple

Description

Gets and returns a reference to the object representing the BAPI method **CreateMultiple**.

Signature

```
getCreateMultiple()
```

Parameters

None.

Return Type

CostCenter.CreateMultiple

Throws

None.

getDeleteMultiple

Description

Gets and returns a reference to the object representing the BAPI method **DeleteMultiple**.

Signature

```
getDeleteMultiple()
```

Parameters

None.

Return Type

CostCenter.DeleteMultiple

Throws

None.

getGetActivityPrices

Description

Gets and returns a reference to the object representing the BAPI method **GetActivityPrices**.

Signature

```
getGetActivityPrices()
```

Parameters

None.

Return Type

CostCenter.GetActivityPrices

Throws

None.

getGetActivityQuantities

Description

Gets and returns a reference to the object representing the BAPI method **GetActivityQuantities**.

Signature

```
getGetActivityQuantities()
```

Parameters

None.

Return Type

CostCenter.GetActivityQuantities

Throws

None.

getGetActivityTypes

Description

Gets and returns a reference to the object representing the BAPI method **GetActivityTypes**.

Signature

```
getGetActivityTypes()
```

Parameters

None.

Return Type

CostCenter.GetActivityTypes

Throws

None.

getGetDetail

Description

Gets and returns a reference to the object representing the BAPI method **GetDetail**.

Signature

```
getGetDetail()
```

Parameters

None.

Return Type

CostCenter.GetDetail

Throws

None.

getGetDetail1

Description

Gets and returns a reference to the object representing the BAPI method **GetDetail1**.

Signature

```
getGetDetail1()
```

Parameters

None.

Return Type

CostCenter.GetDetail1

Throws

None.

getGetList

Description

Gets and returns a reference to the object representing the BAPI method **GetList**.

Signature

```
getGetList()
```

Parameters

None.

Return Type

CostCenter.GetList

Throws

None.

getGetList1

Description

Gets and returns a reference to the object representing the BAPI method **GetList1**.

Signature

```
getGetList1()
```

Parameters

None.

Return Type

CostCenter.GetList

Throws

None.

Reset

Description

Resets the data content of the entire BAPI object; returns **true** if successfully reset. This method is specified by **reset** in the interface **com.stc.jcsre.ETD**.

Parameters

None.

Return Type

boolean

Throws

None.

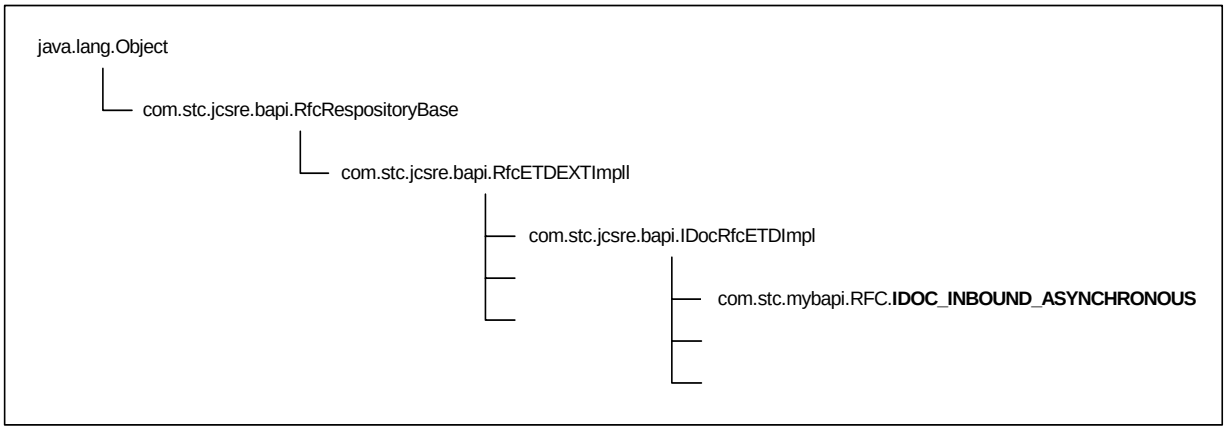
Overrides

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

8.4 IDOC_INBOUND_ASYNCHRONOUS Class

A class representing the IDOC_INBOUND_ASYNCHRONOUS RFC-enabled function defined in SAP R/3. This class implements **com.stc.jcsre.bapi.IDocRfcETD**.

Figure 113 IDOC_INBOUND_ASYNCHRONOUS Class Hierarchy



Definition

```
public class IDOC_INBOUND_ASYNCHRONOUS
```

Methods

8.4.1 Methods

getIDOC_CONTROL_REC_40

Description

Gets and returns the entire **com.sap.mw.jco.JCo.Table** object for the IDOC_CONTROL_REC_40 table parameter.

Signature

```
getIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

com.sap.mw.jco.JCo.Table

Throws

None.

getIDOC_CONTROL_REC_40

Description

Gets and returns a reference to a single row within the IDOC_CONTROL_REC_40 table parameter.

Signature

```
getIDOC_CONTROL_REC_40(i)
```

Parameters

Name	Type	Description
i	int	The zero-based index to the row.

Return Type

IDOC_INBOUND_ASYNCHRONOUS.IDOC_CONTROL_REC_40

Throws

None.

clearIDOC_CONTROL_REC_40

Description

Clears or resets the data content of the IDOC_CONTROL_REC_40 table parameter.

Signature

```
clearIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

void

Throws

None.

resetIDOC_CONTROL_REC_40

Description

Clears or resets the data content of the IDOC_CONTROL_REC_40 table parameter.

Signature

```
resetIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

void

Throws

None.

deleteIDOC_CONTROL_REC_40

Description

Deletes a single row within the IDOC_CONTROL_REC_40 table parameter.

Signature

```
deleteIDOC_CONTROL_REC_40(i)
```

Parameters

Name	Type	Description
i	int	The zero-based index to the row.

Return Type

void

Throws

None.

hasIDOC_CONTROL_REC_40

Description

Determines if there is a non-empty IDOC_CONTROL_REC_40 table parameter; returns **true** if the table parameter has rows.

Signature

```
hasIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

boolean

Throws

None.

countIDOC_CONTROL_REC_40

Description

Counts and returns the number of rows in the IDOC_CONTROL_REC_40 table parameter.

Signature

```
countIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

int

Throws

None.

reinitializeIDOC_CONTROL_REC_40

Description

Re-initializes the table parameter administration for the ETD.

Signature

```
reinitializedIDOC_CONTROL_REC_40()
```

Parameters

None.

Return Type

void

Throws

None.

getIDOC_DATA_REC_40

Description

Gets and returns the entire **com.sap.mw.jco.JCo.Table** object for the IDOC_DATA_REC_40 table parameter.

Signature

```
getIDOC_DATA_REC_40()
```

Parameters

None.

Return Type

com.sap.mw.jco.JCo.Table

Throws

None.

getIDOC_DATA_REC_40

Description

Gets and returns a reference to a single row within the IDOC_DATA_REC_40 table parameter.

Signature

```
getIDOC_DATA_REC_40(i)
```

Parameters

Name	Type	Description
i	int	The zero-based index to the row.

Return Type

IDOC_INBOUND_ASYNCHRONOUS.IDOC_DATA_REC_40

Throws

None.

clearIDOC_DATA_REC_40

Description

Clears or resets the data content of the IDOC_DATA_REC_40 table parameter.

Signature

```
clearIDOC_DATA_REC_40
```

Parameters

None.

Return Type

void

Throws

None.

resetIDOC_DATA_REC_40

Description

Resets or clears the data content of the IDOC_DATA_REC_40 table parameter.

Signature

```
resetIDOC_DATA_REC_40(i)
```

Parameters

Name	Type	Description
i	int	The zero-based index to the row.

Return Type

void

Throws

None.

deleteIDOC_DATA_REC_40

Description

Deletes a single row within the IDOC_DATA_REC_40 table parameter.

Signature

```
deleteIDOC_DATA_REC_40(i)
```

Parameters

Name	Type	Description
i	int	The zero-based index to the row.

Return Type

void

Throws

None.

hasIDOC_DATA_REC_40

Description

Determines if there is a non-empty IDOC_DATA_REC_40 table parameter; returns **true** if the table parameter has rows.

Signature

```
hasIDOC_DATA_REC_40()
```

Parameters

None.

Return Type

boolean

Throws

None.

countIDOC_DATA_REC_40

Description

Counts and returns the number of rows in IDOC_DATA_REC_40 table parameter.

Signature

```
countIDOC_DATA_REC_40()
```

Parameters

None.

Return Type

int

Throws

None.

reinitializeIDOC_DATA_REC_40

Description

Re-initializes the table parameter administration for the ETD.

Signature

```
reinitializeIDOC_DATA_REC_40()
```

Parameters

None.

Return Type

void

Throws

None.

reset

Description

Resets the data content of the RFC object; returns **true** if successfully reset. This method is specified by **reset** in the interface **com.stc.jcsre.ETD**.

Note: *This method is fast, but may fragment memory.*

Signature

```
reset()
```

Parameters

None.

Return Type

boolean

Throws

None.

Overrides

reset in class **com.stc.jcsre.bapi.RfcETDExtImpl**

reset

Description

Resets the data content of the RFC object; returns **true** if successfully reset.

Signature

```
reset(speed)
```

Parameters

Name	Type	Description
speed	boolean	Use **true** for fast reset, **false** to conserve memory.

Return Type

boolean

Throws

None.

re-initialize

Description

Re-initializes the BAPI/RFC ETD.

Signature

```
reinitialize()
```

Parameters

None.

Return Type

void

Throws

None.

registerBapi

Description

Registers the ABAP/4 function name of this BAPI/RFC as a service for the RFC server.

Signature

```
registerBapi()
```

Parameters

None.

Return Type

void

Throws

com.stc.jcsre.EBobConnectionException

unregisterBapi

Description

Unregisters the ABAP/4 function name of this BAPI/RFC as a service for the RFC server.

Signature

```
unregisterBapi()
```

Parameters

None.

Return Type

void

Throws

com.stc.jcsre.EBobConnectionException

isBapiCalled

Description

Determines whether a call from a remote R/3 is directed to this BAPI/RFC method; returns **true** if R/3 has called IDOC_INBOUND_ASYNCHRONOUS.

Signature

```
isBapiCalled()
```

Parameters

None.

Return Type

boolean

Throws

None.

convertTablesToIDoc

Description

Combines the control record table and data record table of an IDOC_INBOUND_ASYNCHRONOUS RFM into a single BLOB, returning a BLOB suitable for marshaling into an IDoc ETD. This method is specified by **convertTablesToIDoc** in the interface **com.stc.jcsre.bapi.IDocRfcETD**.

Signature

```
convertTablesToIDoc()
```

Parameters

None.

Return Type

`byte[]`

Throws

None.

Overrides:

convertTablesToIDoc in class **com.stc.jcsre.bapi.IDOCRfcETDImpl**

Index

A

B

C

D

E

G

H

I

J

W