SeeBeyond™ eBusiness Integration Suite

e*Gate Integrator Collaboration Services Reference Guide

Release 4.5.2

SEEBEYOND™

Version 20020304125213.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter introduces you to this reference guide, its general purpose and scope, and its organization. It also provides sources of related documentation and information.

1.1 Purpose and Scope

SeeBeyond Technology Corporation™ (SeeBeyond™) provides Collaboration Services as part of the SeeBeyond eBusiness Integration™ suite. This document describes each Collaboration Service and discusses how to select and implement the service in a production environment.

This guide explains the following:

- Monk and Pass Through Collaboration Services
- Java Collaboration Service
- C Collaboration Service

Important: *Any operation explanations given here are generic, for reference purposes only, and do not necessarily address the specifics of setting up individual Collaboration Services.*

This document does not contain information on software installation and system administration procedures (see **"Supporting Documents" on page 10**).

1.2 Intended Audience

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system. This person must also have expert-level knowledge of Windows NT/Windows 2000 and UNIX operations and administration and to be thoroughly familiar with Windows-style GUI operations. Use of a language-specific Collaboration Service (Monk, C, or Java) requires familiarity with the appropriate language.

1.3 Organization of Information

This document is organized topically as follows:

- **Chapter 1 "Introduction" on page 7** — Gives a general preview of this document, its purpose, scope, and organization.
- **Chapter 2 "Requirements for Supported Services" on page 11** — Provides an overview of the system requirements for the Collaboration Services that the e*Gate system supports.
- **Chapter 3 "Monk and Pass Through Collaboration Services" on page 13** — Describes the Monk and Pass Through Collaboration Services, including the Monk-related services.
- **Chapter 4 "Java Collaboration Service (JCS)" on page 15** — Describes the Java Collaboration Service and provides in-depth information on how to use it
- **Chapter 5 "C Collaboration Service" on page 27** — Explains how the C Collaboration Service enables the developer to utilize the C and C++ programming languages to write a Dynamic Link Library (**.dll**) file.

In addition there is one appendix:

- **Appendix A "The Java Collaboration Service Prior to 4.5" on page 44** — Describes how to manually code Business Code Logic to use Java Collaboration Service and how to manually promote the Collaboration.

1.4 Writing Conventions

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

Hypertext Links

When you are using this guide online, cross-references are also hypertext links and appear in **blue text** as shown below. Click the **blue text** to jump to the section.

For information on these and related topics, see **"Parameter, Function, and Command Names" on page 9**.

Command Line

Text to be typed at the command line is displayed in a special font as shown below.

```
java -jar ValidationBuilder.jar
```

Variables within a command line are set in the same font and bold italic as shown below.

```
stcregutil -rh host-name -rs schema-name -un user-name
-up password -ef output-directory
```

Code and Samples

Computer code and samples (including printouts) on a separate line or lines are set in Courier as shown below.

```
Configuration for BOB_Promotion
```

However, when these elements (or portions of them) or variables representing several possible elements appear within ordinary text, they are set in *italics* as shown below.

path and *file-name* are the path and file name specified as arguments to **-fr** in the **stcregutil** command line.

Notes and Cautions

Points of particular interest or significance to the reader are introduced with ***Note***, ***Caution***, or ***Important***, and the text is displayed in *italics*, for example:

Note: *The Actions menu is only available when a Properties window is displayed.*

User Input

The names of items in the user interface such as icons or buttons that you click or select appear in **bold** as shown below.

Click **Apply** to save, or **OK** to save and close.

File Names and Paths

When names of files are given in the text, they appear in **bold** as shown below.

Use a text editor to open the **ValidationBuilder.properties** file.

When file paths and drive designations are used, with or without the file name, they appear in **bold** as shown below.

In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.

Parameter, Function, and Command Names

When names of parameters, functions, and commands are given in the body of the text, they appear in **bold** as follows:

The default parameter **localhost** is normally only used for testing.

The Monk function **iq-put** places an Event into an IQ.

You can use the **stccb** utility to start the Control Broker.

1.5 Supporting Documents

The following SeeBeyond documents provide additional information about the e*Gate Integrator system as explained in this guide:

- *Creating an End-to-End Scenario with e*Gate Integrator*
- *e*Gate Integrator Alert Agent User's Guide*
- *e*Gate Integrator Alert and Log File Reference Guide*
- *e*Gate Integrator Installation Guide*
- *e*Gate Integrator Intelligent Queue Services Reference Guide*
- *e*Gate Integrator SNMP Agent User's Guide*
- *e*Gate Integrator System Administration and Operations Guide*
- *e*Gate Integrator User's Guide*
- *SeeBeyond eBusiness Integration Suite Primer*
- *SeeBeyond eBusiness Integration Suite Deployment Guide*
- *Monk Developer's Reference*
- *Standard e*Way Intelligent Adapter User's Guide*
- *Working with Collaboration IDs*
- *XML Toolkit*

See the *SeeBeyond eBusiness Integration Suite Primer* for a complete list of e*Gate-related documentation. You can also refer to the appropriate Microsoft Windows or UNIX documents, if necessary.

Note: *For information on how to use a specific add-on product (for example, an e*Way Intelligent Adapter), see the user's guide for that product.*

1.6 SeeBeyond Web Site

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

http://www.SeeBeyond.com/

Chapter 2

Requirements for Supported Services

Collaboration Services are the libraries that provide the low-level facilities by which Collaborations execute Collaboration Rules.

2.1 Supported Collaboration Services

The Collaboration Services currently supported are:

- C Collaboration Service
- Java Collaboration Service (JCS)
- Monk
- Monk ID
- Pass Through
- Route Table
- XSLT (available with the XML Toolkit add-on; for information on the XSLT Collaboration Service, see the *XML Toolkit*).

The Collaboration Services are automatically installed when you install an e*Gate Participating Host. For information about installing e*Gate, see the *e*Gate Integrator Installation Guide*.

2.2 Requirements

2.2.1 System Requirements

Most of the Collaboration Services have no requirements above those required by a standard e*Gate installation.

- All Collaboration Services require an e*Gate Participating Host version 4.5 or later.
- For information on downloading the Java 2 SDK from **http://java.sun.com/j2se** and using it in conjunction with e*Gate, see the *e*Gate Integrator Installation Guide*.

2.2.2 Important Requirements for the Java 2 SDK on UNIX Systems

- Do not move **Java 2 SDK** to any other location. It must remain where it was installed by the installation process. Upon installation, the location of the **Java 2 SDK** was entered into the operating system's Online Database Management (ODM). Changing the location prevents the proper execution of the Java JNI DLL needed by the JCS.
- The user environment on the Participating Host must have the "dynamic load library" search path environment variable (actual names vary according to the OS) set appropriately to include all directories of the Java 2 SDK installation that contain shared libraries (extensions vary according to OS). See the table below for more information.

Table 1 Java 2 SDK DLL Search Path Environment Variables

OS	DLL Search Path Environment Variable	Extension
Solaris 2.6, 7, or 8	LD_LIBRARY_PATH	.so
HP-UX 11.0 or 11i	SHLIB_PATH	.sl
AIX 4.3.3	LIBPATH	.a
Compaq *Tru64* V4.0F or 5.0A	LD_LIBRARY_PATH	.so
Red Hat Linux 6.2	LD_LIBRARY_PATH	.so

For AIX Participating Hosts only:

In the event that certain PTFs are not installed the LIBPATH environment variable must be set to the following:

- The **jre/bin** directory first followed by the **jre/bin/classic** directory, followed by the directories of other software as needed.

  For example, if Java 2 SDK 1.3 was installed under **/usr/java_dev2**, then (for Bourne Shell, Korn Shell):

  ```
  LIBPATH=/usr/java_dev2/jre/bin:/usr/java_dev2/jre/bin/
  classic:$LIBPATH
  ```
- Add this line into the **egateclient.sh** file, *immediately* prior to the **export LIBPATH** statement.

  For C-shell users:

  ```
  setenv LIBPATH /usr/java_dev2/jre/bin:/usr/java_dev2/jre/bin/
  classic:`printenv LIBPATH`
  ```

  should be added after the current statements that set LIBPATH.

This intervention is necessary because the Java 1.2.2 JNI DLL will cause a core unless the LIBPATH is set as such.

Chapter 3

Monk and Pass Through Collaboration Services

This chapter describes the Monk and Pass Through Collaboration Services, including the Monk-related services.

Note: *The Java Pass Through class, **STCJavaPassThrough.class**, uses the Java Collaboration Service (JCS), not the Pass Through Collaboration Service. See **"Creating Java Collaboration Rules Components" on page 16**.*

3.1 Overview: Monk and Pass Through Services

The Monk-related Collaboration Services are:

- Monk
- Monk ID
- Route Table

In addition, e*Gate provides the Pass Through Collaboration Service. The rest of this chapter explains these services.

3.2 Monk Collaboration Service

The Monk Collaboration Service enables the developer to apply business logic or develop other e*Gate components using SeeBeyond's Monk language.

Monk files can be written using any editor able to create text files (such as **Notepad** or **vi**), or using the e*Gate Collaboration Rules Editor in the Enterprise Manager.

SeeBeyond recommends that most Monk files be given the **.monk** extension. Files created by the Monk Collaboration Rules Editor that are used to transform data within a Collaboration are by default assigned the **.tsc** extension; do not confuse this with the standard extensions for Monk ETDs (**.ssc**) and Java-enabled ETDs (**.xsc**).

3.3 Monk ID Collaboration Service

The Monk ID Collaboration Service enables the developer to execute Collaboration-ID Rules. In versions of e*Gate prior to release 4.0, these rules were most commonly used to validate inbound Events. We recommend that all such validation be performed within a standard Collaboration Rule, and that you only use Collaboration-ID rules when backwards compatibility with prior versions of e*Gate is required.

Collaboration-ID rules are normally created with the e*Gate Collaboration-ID Editor, although they can also be created or modified with text files. The default extension is **.isc**.

3.4 Route Table Collaboration Service

The Route Table Collaboration Service is reserved for users who are upgrading from versions of e*Gate version 3.x. See the *e*Gate Integrator Upgrade Guide*.

3.5 Pass Through Collaboration Service

Note: *Do not use the Pass Through Collaboration Service to communicate with e*Way Connections; use the Java Pass Through class,* ***STCJavaPassThrough.class****, instead. See* **procedure on page 16**.

The Pass Through Collaboration Service provides a means to copy input Events to output Events, leaving the Event contents unchanged. The Service simply performs a byte-for-byte copy for all data that it processes.

Note: *No Collaboration Rules are required to execute the Pass Through Collaboration Service.*

Chapter 4

Java Collaboration Service (JCS)

This chapter describes the e*Gate Java Collaboration Service and how to use it.

4.1 What is the 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 is instantiated and uses the associated Java Collaboration Rules class to accomplish the data transformation.

Unlike the Monk Collaboration Service, which allows only one-to-one Collaboration between Events, the Java Collaboration Service allows many-to-many Collaborations.

Note: *See* **Chapter 1** *for requirements specific to the Java 2 SDK.*

Note: *It is possible, but not recommended, to avoid the Java Collaboration Rules Editor and manually create* ***.class*** *files that use the Java Collaboration Service. For instructions on how to accomplish this, see* **Appendix A***.*

4.2 How to use the Java Collaboration Service

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.

Important: *Before creating a Java Collaboration, you must have created the Java-enabled ETDs (****.xsc*** *files) used by the Collaboration. For information on creating a Java ETD, refer to the material on the Java ETD Editor in the* ***e*Gate Integrator User's Guide****, or refer to the online help for the Java ETD Editor.*

The following procedures recapitulate material in the *e*Gate Integrator User's Guide*.

4.2.1 Creating Java Collaboration Rules Components

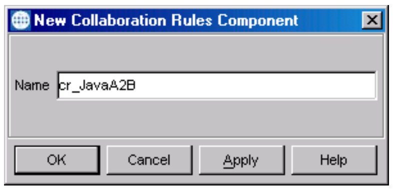
In the general case, when you create a new Collaboration Rule, you must specify inbound and outbound Event Type instances and the rules for transforming the data between them; see **procedure on page 17**.

However, a simple Java Collaboration Rule is presupplied: the Java Pass Through rule. Like the Pass Through Collaboration Service, the Java Pass Through rule transports data without transforming it. Unlike the Pass Through Collaboration Service, you can use the Java Pass Through rule to communicate with e*Way Connections. You are not required (or permitted) to specify instance names or initialization strings.

To create a Java Pass Through Collaboration Rule

1. Use e*Gate Enterprise Manager to create and name a new Collaboration Rules component. See Figure 1.

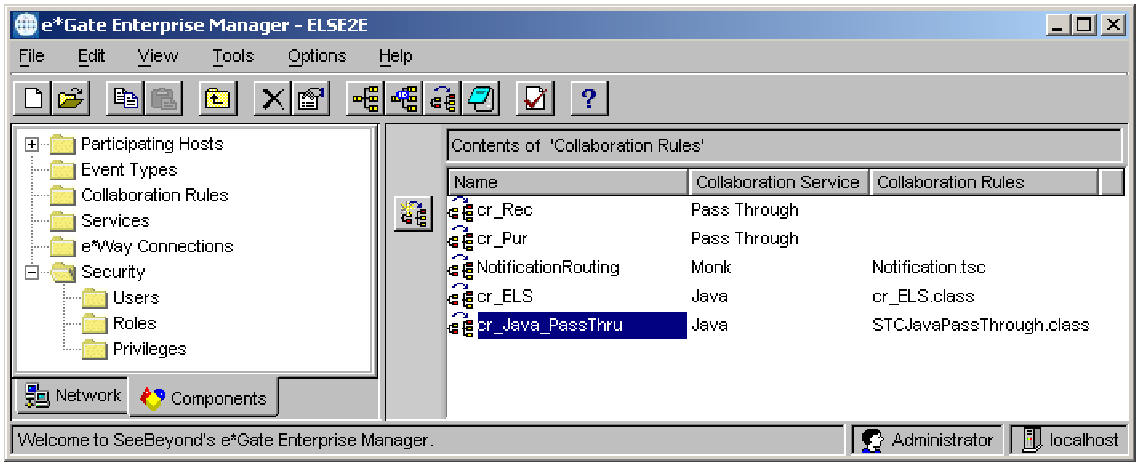
Figure 1 New Collaboration Rules Component



Name cr_JavaA2B

2. Edit the properties of the new Collaboration Rule.
3. In the **Properties** dialog: In the **General** tab, Collaboration Rules area, click **Find**.
4. Navigate to the **collaboration_rules\STCLibrary** folder.
5. Click **STCJavaPassThrough.class** and click **Select**.
6. In the **Properties** dialog box, click **OK** to save your changes and close the dialog. See Figure 2.

Figure 2 STCJavaPassThrough.class - Java Pass Through Collaboration Rule



Contents of 'Collaboration Rules'

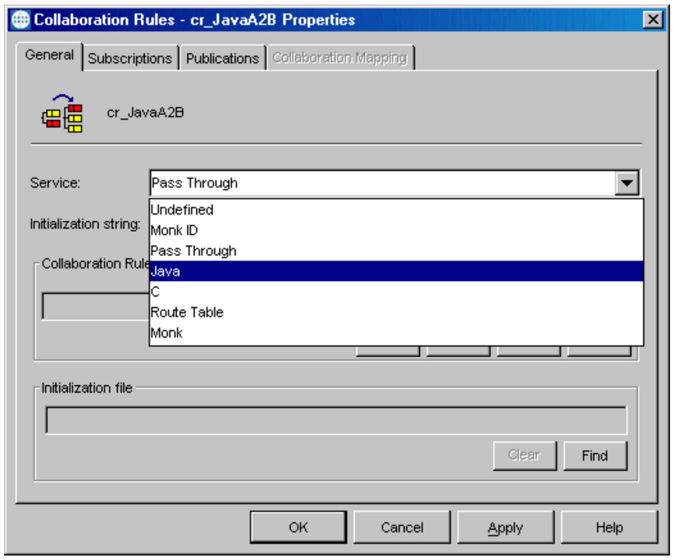
Name	Collaboration Service	Collaboration Rules
cr_Rec	Pass Through	
cr_Pur	Pass Through	
NotificationRouting	Monk	Notification.tsc
cr_ELS	Java	cr_ELS.class
cr_Java_PassThru	Java	STCJavaPassThrough.class

Welcome to SeeBeyond's e*Gate Enterprise Manager.

To create a Java Collaboration Rules component

1. Use e*Gate Enterprise Manager to create and name a new Collaboration Rules component. See **Figure 1 on page 16**.
2. Edit the properties of the new Collaboration Rule. If necessary, on the **General** tab, select **Java** as the Collaboration Service. See Figure 3.

Figure 3 Selecting the Java Collaboration Service

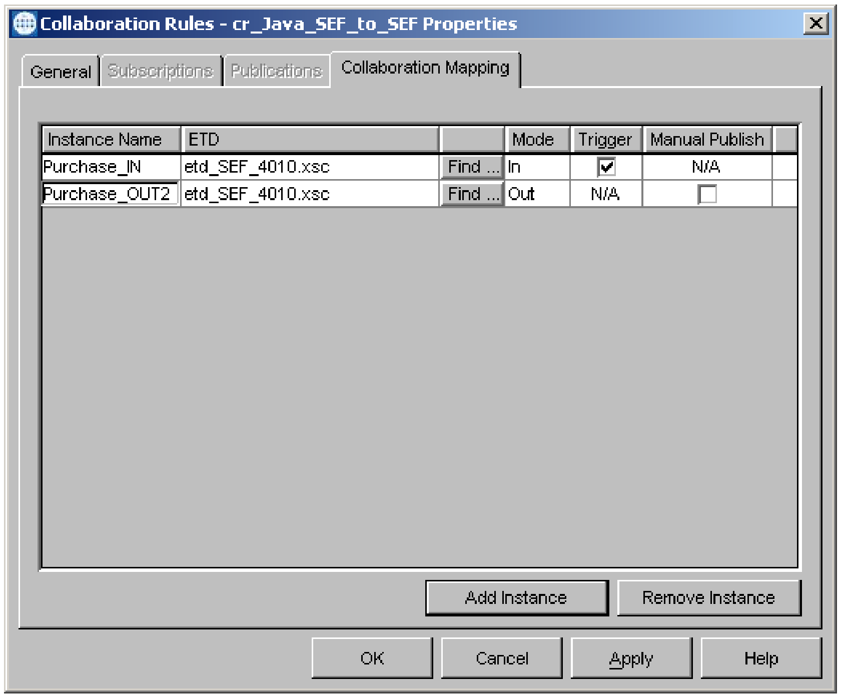


3. Click the **Collaboration Mapping** tab.
4. Click **Add Instance** to add a new instance.
5. Enter an **Instance Name** for the instance.

   The **Instance Name** will be used by the Collaboration Rules Editor to identify the source and destination Events.
6. Click **Find** to display a list of ETD files (**.xsc** files), and then select the source ETD.

   The name of the ETD is displayed in the **ETD** field.
7. In the **Mode** list, click **In** or **In/Out**.
8. Optionally, repeat steps 4 through 7 to create additional source Event instances.
9. As necessary, select the **Trigger** check box for one or more inbound Events.
10. Repeat steps 4 through 6 for each destination instance.
11. In the **Mode** list, click **Out** or **In/Out** for each destination instance.
12. You can select the **Manual Publish** check box for zero or more outbound Events.
13. Click **Apply** to save the changes. See Figure 4.

Figure 4 Collaboration Mapping



Instance Name	ETD		Mode	Trigger	Manual Publish
Purchase_IN	etd_SEF_4010.xsc	Find ...	In	✓	N/A
Purchase_OUT2	etd_SEF_4010.xsc	Find ...	Out	N/A	☐

14 Click the **General** tab.

15 Optionally, you can enter an initialization string to override certain run-time settings.

16 Click **New** to create a new Java Collaboration Rule.

The Java Collaboration Rule Editor starts.

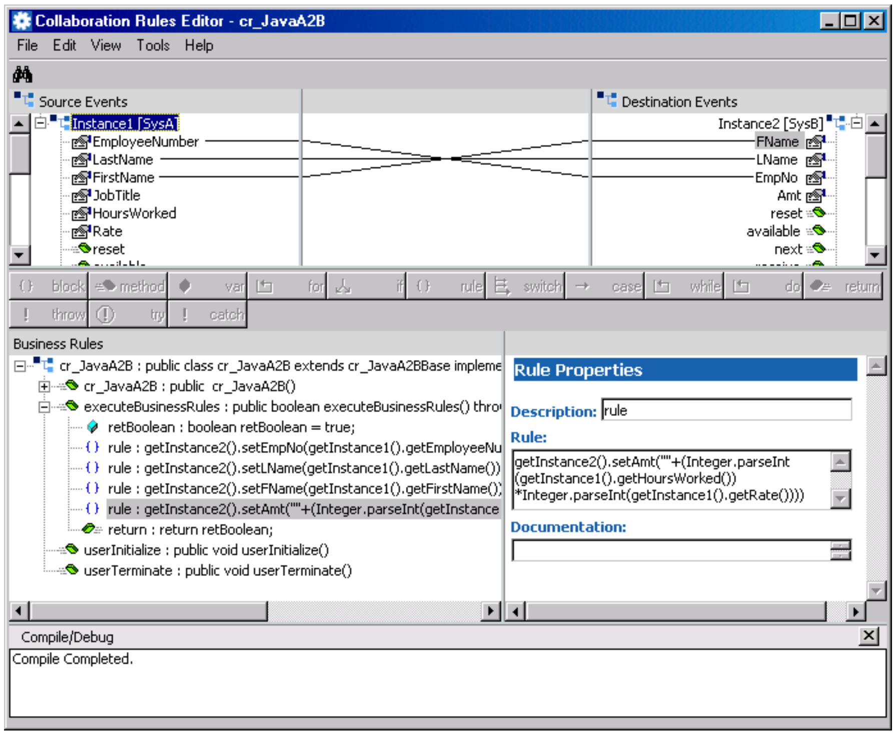
17 Using the Editor, add the required business logic (Java code) for this rule to the **executeBusinessRules()** method.

This process is made simpler and more robust by the GUI, which allows you to:

- Drag a node into areas of the Properties pane to generate **get()** methods.
- Drag a node onto another node to generate **get()**/**set()** methods.
- Right-click a node to view its properties.
- Right-click a pane to gain access to external Java packages and their methods.

A snapshot of a Collaboration Rule as seen through the Editor is shown in Figure 5. For information on using the Editor, refer to the Chapter 7 in the *e*Gate Integrator User's Guide*, or refer to the online help for the Java Collaboration Rules Editor.

Figure 5 Java Collaboration Rules Editor


Collaboration Rules Editor - cr_JavaA2B
File Edit View Tools Help
Source Events
Destination Events
block method var for if rule switch case while do return
throw try catch
Business Rules
Compile/Debug

Source Events:
- Instance1 [SysA]
  - EmployeeNumber
  - LastName
  - FirstName
  - JobTitle
  - HoursWorked
  - Rate
  - reset

Destination Events:
- Instance2 [SysB]
  - FName
  - LName
  - EmpNo
  - Amt
  - reset
  - available
  - next

Business Rules:

```
cr_JavaA2B : public class cr_JavaA2B extends cr_JavaA2BBase impleme
  cr_JavaA2B : public cr_JavaA2B()
  executeBusinessRules : public boolean executeBusinessRules() thro
    retBoolean : boolean retBoolean = true;
    rule : getInstance2().setEmpNo(getInstance1().getEmployeeNu
    rule : getInstance2().setLName(getInstance1().getLastName())
    rule : getInstance2().setFName(getInstance1().getFirstName()
    rule : getInstance2().setAmt(""+(Integer.parseInt(getInstance
    return : return retBoolean;
  userInitialize : public void userInitialize()
  userTerminate : public void userTerminate()
```

Rule Properties

Description: rule

Rule:

```
getInstance2().setAmt(""+(Integer.parseInt
(getInstance1().getHoursWorked())
*Integer.parseInt(getInstance1().getRate())))
```

Documentation:

Compile Completed.

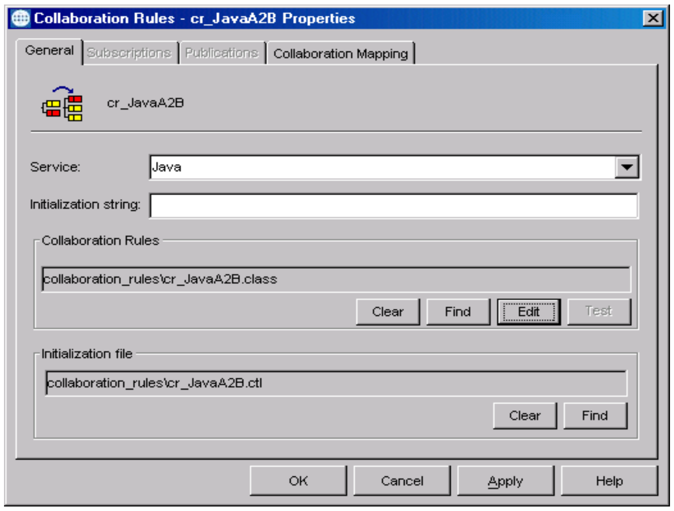
18 As you work on it, save and compile the Collaboration Rules class.

19 After it compiles cleanly, promote it and exit the Editor.

In the **Properties** dialog box for the Collaboration Rule, on the **General** tab, the Collaboration Rules **.class** file is entered in the **Collaboration Rules** box and the corresponding control file is entered in the **Initialization file** area.

The Collaboration Rule now uses the newly created Java class to perform the required data transportation and transformation. See Figure 6.

Figure 6 Collaboration Rules - Properties



cr_JavaA2B

Service: Java

Initialization string:

Collaboration Rules: collaboration_rules\cr_JavaA2B.class

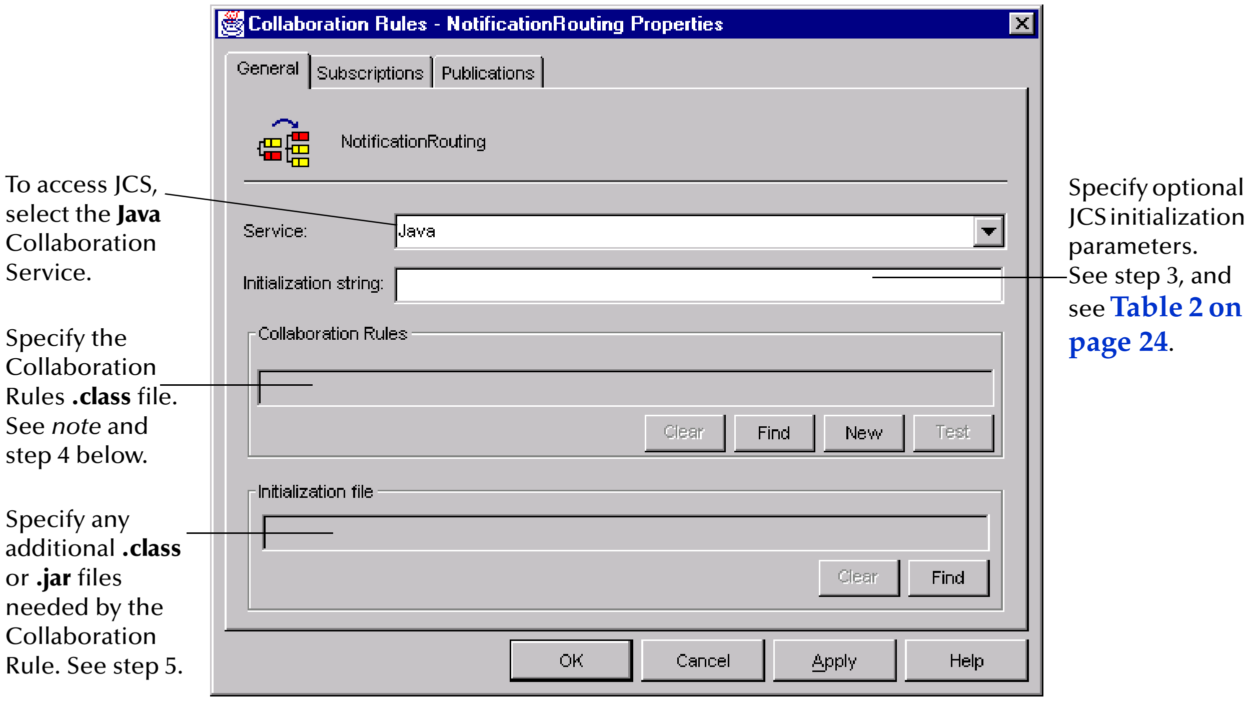
Initialization file: collaboration_rules\cr_JavaA2B.ctl

4.2.2 Implementing Java Collaboration Rule Components

To define a Collaboration Rule that uses the JCS

1. In the Enterprise Manager, define a Collaboration Rules component using the steps in **"Creating Java Collaboration Rules Components" on page 16**.
2. Set the Collaboration Rules properties as shown in Figure 7 below.

Figure 7 Collaboration Rules Properties Sheet


Collaboration Rules - NotificationRouting Properties
General
Subscriptions
Publications
NotificationRouting
Service:
Java
Initialization string:
Collaboration Rules
Clear
Find
New
Test
Initialization file
Clear
Find
OK
Cancel
Apply
Help
To access JCS, select the **Java** Collaboration Service.
Specify the Collaboration Rules **.class** file. See *note* and step 4 below.
Specify any additional **.class** or **.jar** files needed by the Collaboration Rule. See step 5.
Specify optional JCS initialization parameters. See step 3, and see **Table 2 on page 24**.

3. Enter any required JCS initialization parameters in the **Initialization string** box.

   For example:

   -jnidll *myjnidll* **-java1** *com.mystc.myProgram*

   Table 2 on page 24 provides a list of the JCS initialization parameters.

   For any parameter that contains embedded spaces, the entire parameter must be contained within doublequotes ("*parm name*"). If it is not, the JCS will not be able to locate the file specified and therefore will be unable to perform initialization.

   For example:

   -jnidll "C:\Program Files\JavaSoft\JRE\1.2\bin\classic\jvm.dll"

   Certain initialization parameters cause the Collaboration Rules pane to become available.

4. In the Collaboration Rules box, if required by the initialization parameter, enter the appropriate path and filename.

Note: *The class name may exist in a* ***.jar*** *or* ***.zip*** *file.*

5 If the Collaboration Rule requires an additional **.class** or **.jar** file, enter its name in the **Initialization file** text box. If more than one file is required, reference the necessary files in an e*Gate Registry control file, and specify the Registry control file in the **Initialization file** text box. For more information, see **"To commit the file new.jar to the classes/path within the e*Gate repository:" on page 22**.

When committing the Java class, if it is placed into a Java package, you must use the correct path location. This consists of: The **classes/** directory prepended to the package name, then converting all periods (**.**) to forward slashes (**/**). For example:

```
classes.com.stc.common.collabService.myClassFile (The class name)
classes/com/stc/common/collabService/myClassFile (The path to
commit the class)
```

4.2.3 Dealing With Long CLASSPATHs

In some instances, the classpath may exceed 255 characters. There are several possible ways to accommodate this.

- The JCS automatically uses the CLASSPATH environmental variable. The user can refer to all **.jar** files and directories here, allowing for a hardcoded maximum of 4096 characters for the classpath supplied to Java.
- If using one global CLASSPATH environmental variable for all JCS is not desirable, you can reference different environmental variables in the **`-classpath`**, **`-cp`**, or **`-jnidll`** options by enclosing the name with percent (**%**) characters. For example: `%YOURCLASSPATH%`.
- If it is also important that the JCS run on different Participating Hosts, then all **.jar**, **.zip**, and **.class** files can be checked into the e*Gate Registry and referenced from a Registry Control file. The Registry Control file can then be entered into the **Initialization File** text box of the **Collaboration Rules Properties** sheet.
  For example:

  FILE1.jar,classes,FILETYPE_BINTEXT

  FILE2.zip,classes,FILETYPE_BINTEXT

  FILE3.class,classes,FILETYPE_BINTEXT

  When the JCS processes this control file, it downloads the respective files and constructs this string as part of the JVM classpath variable:

  <EG SharedData>/classes/FILE1.jar;<EG SharedData>/classes/FILE2.zip

  The **<EG SharedData>/classes** directory is a standard part of the JVM classpath.

4.2.4 Committing Java Classes and .jar Files to the Registry

After the Java Collaboration Rules have been compiled, the resultant **.class** files must be committed to the e*Gate Registry under the **classes/** directory. Additionally, any other supporting Java classes must be compiled and stored in **.jar** files. These **.jar** files must also be committed to the e*Gate Registry under the **classes/** directory.

You normally commit files to the Registry using Enterprise Manager: On the **File** menu, click **Commit to Sandbox**. However, you can also commit files using the **stcregutil.exe** command-line utility.

The following example demonstrates that you can also commit files by running **stcregutil** with the **-fc** (file commit) flag. The example is printed on more than one line for clarity, but must be issued as a single command line.

To commit the file new.jar to the classes/path within the e*Gate repository:

1 Create a control (**.ctl**) file with a text editor giving it a name such as **myjar.ctl**. Each line within **myjar.ctl** should have the following format:

new.jar,classes,FILETYPE_BINTEXT

Note: *There must mot be any spaces before or after the commas (,).*

2 Run the **stcregutil** utility by typing the following at the command line:

```
stcregutil -rh registry -rs schema -un user-name
   -up password -fc classes -ctl myjar.ctl
```

For more information about the **stcregutil.exe** command-line utility, see the *e*Gate Integrator System Administration and Operations Guide*.

You can also commit one file at a time using other **File** menu options; for example, see below.. Additional information is available in the online help for Enterprise Manager.

Using the .ctl File to Download Entries from the Registry

If you want to use the **.ctl** file as a vehicle for downloading entries from the registry, you can use the special text editor provided within Enterprise Manager to make changes. If you place your changes at the end of the file and then immediately re-commit the **.ctl** file, your changes are preserved.

To make permanent edits to a .ctl file

1 In Enterprise Manager, on the **File** menu, click **Edit File**.

The **Open File** dialog box appears.

2 Set the **Files of type** to **All files** and then open the **collaboration_rules** folder.

3 Locate and open the **.ctl** file you want to edit.

4 Place your commands at the end of the file, *after* comment block beginning:

```
#USER DOWNLOADABLE ENTRIES
#
```

For example, after your edits, the file might look like this (emphasis added for greater clarity):

```
[...]
#USER DOWNLOADABLE ENTRIES

# Entries below this section will be preserved. Any entries
# found above this section will be overwritten when the
# collaboration rule is compiled.
#
# /--Next two lines added by pc 2002-02-29 per TR 98765 ---\
MyFile.jar,C:\ThisPath\ThatPath\Folder,FILETYPE_BINTEXT
MyWord.txt,C:\MyPath,FILETYPE_ASCII
# \----------------------------------------- End TR 98765 ---/
#
```

5 Exit the editor, saving your changes; when the system prompts you to Commit the file, answer **Yes**.

6 In response to the system prompt, navigate to the location where the file should be stored, and then save your changes.

4.3 Parameters for the JCS Initialization String

The following table lists all parameters and parameter values recognized by the Java Collaboration Service. All parameters are optional.

Table 2 JCS Initialization Parameters

Parameter	Value	Purpose
-classpath	An absolute path, or an environmental variable	Specifies the CLASSPATH that the JVM will use. If this parameter is not specified, the JCS will create an appropriate default CLASSPATH variable containing all **.jar** files, class directories, and any additional files declared as necessary. This parameter can accept reference to an environmental variable (for example: **-classpath %MYPATH%**). ***Caution:*** The -classpath parameter *completely overrides* the default CLASSPATH. Thus, if you specify -classpath without supplying all the required paths, or if some required paths are not available, the JCS will not run. If, instead, you want to add or suggest a classpath, use the -cp or appendenvcp parameters instead; see below.
-appendenvcp	String	Specifies any string or environment variable to **ap**pend the current CLASSPATH used by JCS. The **-appendenvcp** parameter is preferred to -classpath because it will not override any necessary paths.
-cp	String	Specifies any string or environment variable to **pre**pend the current CLASSPATH used by JCS. The **-cp** parameter is preferred to -classpath because it will not override any necessary paths.

Table 2 JCS Initialization Parameters (Continued)

Parameter	Value	Purpose		
-debug	Integer	Specifies a port number; a setting of **-debug 8000** is the default if nothing is specified. Allows you to run either JDB or e*Gate Java Debugger for tracing and finding errors in Collaboration Rules.		
-def	String	Defines a Java property using the following format: **-def *propertyname*** = ***value***.		
-jnidll	An absolute path, or an environmental variable	Specifies the location of the (Java Native Interface (JNI) dynamic-load library (DLL). The absolute path name of where the installed Java JNI DLL library is found. The JNI DLL name varies on different OS platforms:		
			Java 2	**Java 1**
		Windows 2000	jvm.dll	javai.dll
		Windows NT	jvm.dll	javai.dll
		Solaris 2.6, 7, or 8	libjvm.so	libjava.so
		HP-UX 11.0 or 11i	libjvm.sl	libjava.sl
		AIX 4.3.3	libjvm.a	libjava.a
		Compaq *Tru64*	libjvm.so	libjava.so
		Linux	libjvm.so	libjava.so
		The JNI DLL must be located on the Participating Host in the same directory in which the SDK or JRE installed it. The value assigned can contain a reference to an environment variable enclosed between % symbols (such as %**JREJNIDLL**%). Such variables can be used when multiple Participating Hosts are used on different platforms.		
-java1	None	Specifies that the version of the jnidll location is for Java version 1.1.7b. If unspecified, the jnidll specified is assumed to be for Java 2.		
		Windows	javai.dll	
		Solaris, Linux,	libjava.so	
		Compaq	libjava.so	
		HP	libjava.sl	
		AIX	libjava.a	

Table 2 JCS Initialization Parameters (Continued)

Parameter	Value	Purpose
-ldp	An absolute path, or an environmental variable	Overrides the default directory of the dll. A suitable library load path is configured automatically by the JCS. This parameter prepends the specified paths to the library load path used. For example you could use this parameter when Java code contains IDS-out-wrapper classes that need a specified library file.
-ms	Integer	Specifies initial heap size of the Java Virtual Machine (JVM) in bytes. The default size in bytes is 32000000. If larger data is to be processed, this parameter must be defined.
-mx	Integer	Specifies the maximum heap size for the Java Virtual Machine in bytes to control the maximum size limitation of the JVM.
-noclasgc	Expects no input	Disables class garbage collection. If this parameter is in place, no memory deallocation will take place for the JVM.
-nojit	Expects no input	Disables the just-in-time compiler.
-suspend	Expects no input	When **-suspend** is specified, the JVM waits for an attach to occur before executing the Collaboration Rule.
-verbose	Expects no input	Reports JVM information and all class loads.
-verbosegc	Expects no input	Enables **g**arbage **c**ollection console activity.
	Example: ▪ com.mystc. myProgram	If you indicate a path location as the last parameter in the **Initialization string** text box, it is unnecessary to indicate a Collaboration Rule in the **Collaboration Rules** text box.

Chapter 5

C Collaboration Service

The C Collaboration Service enables the developer to utilize the C and C++ programming languages to write a Dynamic Link Library (**.dll**) file. Selected via the GUI from the **Collaboration Rules** dialog box, e*Gate compiles and links the external source code to create the dynamic or shared library.

For example:

- You may already have a library or application written in C or C++ and want to make it accessible to your e*Gate applications.
- You want to implement a portion of time-critical code, written in C or C++ and have the e*Way call these functions.
- You may have application-specific problems that are better handled outside of the Monk programming environment.

5.1 Header File: HTRANSCC.h

This section contains the description of the header file **HTRANSCC.h**, which is used to pass a string in and out of the interface object.

The object types passed between the application and external include character blobs, wide character blob, long, booleans, characters, wide characters, double floating point numbers, external interface objects and vectors of these types.

The external interface object is used to implement the external interfaces. A structure is defined that contains a location where the user can place data for the object as well as functions that implement the interface.

```
#ifndef STCCCOLLAB_H
#define STCCCOLLAB_H

#include    "gendefs.h"
#include    "stcapis.h"
#include    "stctrans.h"

#ifdef __cplusplus
extern "C"
{
#endif

#define CEXT_VERSION "CEXTV1"

typedef void    *HTRANSCC;
```

```
//----------------------------------------------------------------
//  IQInitialTopic
//  --------------------------------------------------------------
//
//  Purpose:
//
//      provides access to the name of the Event Type that initiated
//      the current translation.
//
//      This is a C Collaboration equivalent of the iq-initial-topic
//      Monk function.
//
//      (For use within ccollab_translate)
//
//  --------------------------------------------------------------
//
//  Parameters:
//
//      pcszInitialTopic:
//                returns a null-terminated string containing the
//              Initial topic name. The string must be pre-allocated.
//
//     pvData:   passes the incoming pvData parameter through for all
//                calls. (For internal use.)
//
//      return:   if successful, this function will return TRUE.
//
//----------------------------------------------------------------
extern
BOOL
DLLEXP
APIDEF
IQInitialTopic (OUT char *pszInitialTopic,
                IN void *pvData);


//----------------------------------------------------------------
//  IQGet
//  --------------------------------------------------------------
//
//  Purpose:
//
//      retrieves and removes the next pending message from the
//      default IQ for the current Collaboration.
//
//      This is a C Collaboration equivalent of the iq-get Monk
//      function.
//
//      (For use within ccollab_translate.)
//
//  --------------------------------------------------------------
//
//  Parameters:
//
//      pcszInputTopic:
//                      the name of the Event Type to get.
//
//      hIQ:            currently not in use. Must be NULL.
//
//      pbMsgData:      the retrieved byte data. NULL if call failed.
//
//      pbMsgDataLen:   the length of pbMsgData.
```

```
//
//      pvData:   passes the incoming pvData parameter through for
//                all calls. (For internal use.)
//
//      return:   if successful, this function will return TRUE.
//
//-------------------------------------------------------------------
extern
DLLEXP
BOOL
APIDEF
IQGet (IN const char *pcszInputTopic,
       IN OUT HIQ hIQ,
       OUT char *pbMsgData,
       OUT DWORD *pbMsgDataLen,
       IN void *pvData);


//-------------------------------------------------------------------
//  IQInputTopics
//  -----------------------------------------------------------------
//
//  Purpose:
//
//      returns a comma-separated list of the names of all input
//      Event Types for the current Collaboration.
//
//      This is a C collaboration equivalent of the iq-input-topics
//      Monk function.
//
//      (For use within ccollab_translate.)
//
//  -----------------------------------------------------------------
//
//  Parameters:
//
//      pcszCVSInputTopics:
//                provides a list of comma-separated values of
//                all Event Types for the current collaboration.
//                This string must be pre-allocated.
//
//      pvData:   passes the incoming pvData parameter through for
//                all calls. (For internal use.)
//
//      return:   if successful, this function will return TRUE.
//
//-------------------------------------------------------------------
extern
BOOL
DLLEXP
APIDEF
IQInputTopics (OUT char *pszCSVInputTopics,
               IN void *pvData);


//-------------------------------------------------------------------
//  IQOutputTopics
//  -----------------------------------------------------------------
//
//  Purpose:
//
//      returns a comma-separated list of the names of all output
//      Event Types for the current Collaboration.
//
```

```
//      This is a C Collaboration equivalent of the iq-output-topics
//      Monk function.
//
//      (For use within ccollab_translate.)
//
//  ----------------------------------------------------------------
//
//  Parameters:
//
//      pcszCVSInputTopics:
//                provides a list of comma-separated values of
//                all Event Types for the current Collaboration.
//                This string must be pre-allocated.
//
//      pvData:   passes the incoming pvData parameter through for
//                all calls. (For internal use.)
//
//      return:   if successful, this function will return TRUE.
//
//------------------------------------------------------------------
extern
DLLEXP
BOOL
APIDEF
IQOutputTopics (OUT char *pszCSVOutputTopics,
                IN void *pvData);


//-------------------------------------------------------------------
---------
//  IQPut
//  ----------------------------------------------------------------
//
//  Purpose:
//
//      places an event on the output queue, but does not commit
//      it to the queue until the transformation function returns
//      successfully.
//
//      This is a C Collaboration equivalent of the iq-output-topics
//      Monk function.
//
//      (For use within ccollab_translate.)
//
//  ----------------------------------------------------------------
//
//  Parameters:
//
//      pcszOutputTopic:
//                  pass the name of the output topic to publish.
//
//      pbMsgData:
//                  pass the data to publish.
//
//      pcszCSVInputEventTypes:
//                  pass a comma-separated list of the input
//                  Event Types which were used to create this data.
//
//      dwPriority:
//                  priority to assign to the output Event.
//
//      dwMajorSeqNumber:
//                  major sequence number to assign.
//
```

```
//      dwMinorSeqNumber:
//                minor sequence number to assign.
//
//      pvData:   passes the incoming pvData parameter through for
//                all calls. (For internal use.)
//
//      return:   if successful, this function will return TRUE.
//
//-------------------------------------------------------------------
extern
BOOL
DLLEXP
APIDEF
IQPut (IN const char *pcszOutputTopic,
       IN const STC_BLOB *pbMsgData,
       IN const char *pcszCSVInputTopics,
       IN DWORD dwPriority,
       IN DWORD dwMajorSeqNumber,
       IN DWORD dwMinorSeqNumber,
       IN void *pvData);


//-------------------------------------------------------------------
//  ccollab_init
//  -----------------------------------------------------------------
//
//  Purpose:
//
//      Used to initialize the DLL
//
//      The handle that is optionally returned is passed into all
//      other functions.
//
//  -----------------------------------------------------------------
//
//  Parameters:
//
//     phCC:      if successful, a STC Session handle that is needed
//                 for all STC APIs.
//
//      pcszInitFile:
//                 if pcszInitFile[0] != 0x00, then this is
//                 the initialization file configured for the
//                 Collaboration.
//
//      pcszInitialization:
//                 if pcszInitialization[0] != 0x00, then this
//                 is the initialization string configured for
//                 the Collaboration.
//
//      dwFlags:    bit flags. Reserved for future use.
//
//      pvReserved: this param is reserved for future use and
//                 MUST be set to NULL.
//
//      return:     if successful, this function should return TRUE.
//                 If an error occurs, it should return FALSE and
//                 make a call to SETLASTERROR(x) where "x" is the
/                  GENERRO_xxx code defined in generror.h.
//
//-------------------------------------------------------------------
extern
DLLEXP
BOOL
```

```
APIDEF
ccollab_init(OUT HTRANSCC *phCC,
             IN const char *pcszInitFile,
             IN const char *pcszInitialization,
             IN DWORD dwFlags,
             IN OUT OPTIONAL void *pvReserved);


//------------------------------------------------------------------
//  ccollab_translate
//  ----------------------------------------------------------------
//
//  Purpose:
//
//      translate pInMsg to pReturnMsg.
//
//  ----------------------------------------------------------------
//
//  Parameters:
//
//      hCC:        The handle returned from ccollab_init.
//
//      pInMsg:     pointer to a blob that is the Event to translate.
//
//      pReturnMsg: the address of an STC_BLOB that, upon success,
//                 the implementation should fill out the cbData and
//                  pbData members with the translated Event.
//
//      dwFlags:    bit flags. Reserved for future use.
//
//      pvData:     This variable is required as a parameter to all
//                  IQ Service calls.
//
//      return:     if successful, this function should return TRUE.
//                  If an error occurs, it should return FALSE and
//                  make a call to SETLASTERROR(x) where "x" is the
/                   GENERRO_xxx code defined in generror.h.
//
//------------------------------------------------------------------
extern
DLLEXP
BOOL
APIDEF
ccollab_translate(IN HTRANSCC hCC,
                  IN STC_BLOB *pInMsg,
                  IN OUT STC_BLOB *pReturnMsg,
                  IN DWORD dwFlags,
                  IN OUT OPTIONAL void *pvData);


//------------------------------------------------------------------
//  ccollab_free
//  ----------------------------------------------------------------
//
//  Purpose:
//
//      free the memory allocated via the call to ccollab_translate
//      for the pReturnMsg blob.
//
//  ----------------------------------------------------------------
//
//  Parameters:
//
//      hCC:        The handle returned from ccollab_init
```

```
//
//      pReturnMsg: the address of an STC_BLOB that the pbData needs
//                 to be de-allocated.  This function should set the
//                  cbData = 0 and the pbData = NULL.
//
//      return:     if successful, this function should return TRUE.
//                  If an error occurs, it should return FALSE and
//                  make a call to SETLASTERROR(x) where "x" is the
/                   GENERRO_xxx code defined in generror.h.
//
//-----------------------------------------------------------------
extern
DLLEXP
BOOL
APIDEF
ccollab_free(IN HTRANSCC hCC,
             IN STC_BLOB *pReturnedMsg);


//-----------------------------------------------------------------
//  ccollab_term
//  ---------------------------------------------------------------
//
//  Purpose:
//
//      notification of termination and oportunity to clean up hCC.
//
//  ---------------------------------------------------------------
//
//  Parameters:
//
//      hCC:        The handle returned from ccollab_init
//
//      return:     if successful, this function should return TRUE.
//                  If an error occurs, it should return FALSE and
//                  make a call to SETLASTERROR(x) where "x" is the
/                   GENERRO_xxx code defined in generror.h.
//
//-----------------------------------------------------------------
extern
DLLEXP
BOOL
APIDEF
ccollab_term(IN HTRANSCC hCC);


//-----------------------------------------------------------------
//  ccollab_version
//  ---------------------------------------------------------------
//
//  Purpose:
//
//      signals intent to use enhanced C collaboration features
//      (such as IQ Service functions)
//
//-----------------------------------------------------------------
//
//  Parameters:
//
//      pszVersion: store CEXT_VERSION to enable features
//                  (pszVersion is pre-allocated)
//
//-----------------------------------------------------------------
extern
```

```
DLLEXP
void
APIDEF
ccollab_version(OUT char *pszVersion);


#ifdef __cplusplus
}
#endif

#endif // STCCCOLLAB_H
```

5.2 Developing the C Dynamic Link Library (.dll) File

The sample code in the **newcollab.c** (or **.cpp**) file shown below demonstrates the business logic the C Collaboration Service uses. The **.c** (or **.cpp**) file must be compiled externally into a **.dll** file.

```
#include "HTRANSCC.h"

BOOL

collab_init(OUT HTRANSCC *phCC,

        IN const char *pcszInitFile,

        IN const char *pcszInitString,

        IN DWORD dwFlags,

        IN OUT OPTIONAL void *pvReserved)


  return(TRUE);

BOOL

ccollab_translate(HTRANSCC hCC,

      STC_BLOB *sInBlob,

      STC_BLOB *sOutBlob,

      DWORD dwFlags, void *pvReserved)


sOutBlob->pbData = (BYTE *)malloc(sInBlob->cbData);
    if (!(sOutBlob->pbData))
    {
        return(FALSE);
    }
    sOutBlob->cbData = sInBlob->cbData;
    memcpy(sOutBlob->pbData, sInBlob->pbData, sInBlob->cbData);
    return(TRUE);
}

BOOL
```

```
collab_free(IN HTRANSCC hCC,

     IN STC_BLOB *pReturnedMsg)

    if (pReturnedMsg)
    {
        if (pReturnedMsg->pbData)
        {

            free(pReturnedMsg->pbData);
        }

        pReturnedMsg->pbData = NULL;
                pReturnedMsg->cbData = 0;
    }

    return(TRUE);


BOOL

collab_term (IN HTRANSCC hCC)

 return (TRUE);
```

Within the **ccollab_translate** function, you can implement any code you like to perform the business logic required by this Collaboration.

5.2.1 Monk IQ Functions That Do Not Support JMS IQs

The following Monk functions do not support JMS IQs:

- iq-get-header
- iq-mark-unusable
- iq-peek

5.3 The C Collaboration APIs

The next several pages list the e*Gate APIs available within the C Collaboration Service.

ccollab_free() on page 37

ccollab_init() on page 38

ccollab_term() on page 39

ccollab_translate() on page 40

ccollab_free()

Syntax

```
ccollab_free(IN HTRANSCC hCC,
             IN STC_BLOB *pReturnedMsg);
```

Description

ccollab_free() deallocates the memory associated with the **pReturnedMsg** in the **ccollab_translate()** call.

Parameters

Name	Type	Description
hCC	HTRANSC	The handle indicated by **ccollab_init()**.
pReturnedMsg	A pointer	A pointer to the message or blob associated with **ccollab_translate()**.

Return Value

Boolean: If successful, returns **true**; otherwise, returns **false**.

ccollab_init()

Syntax

```
ccollab_init(OUT HTRANSCC *phCC,
             IN const char *pcszInitFile,
             IN DWORD dwFlags,
             IN OUT OPTIONAL void *pvReserved);
```

Description

ccollab_init() is defined in the loadable extension library. It is called directly after loading the library. It initializes an interface object and returns it to the calling function.

Parameters

Name	Type	Description
phCC	Pointer	A pointer to the handle. Pass this function an address of an empty handle, and the function will return the handle for use by other functions
pcszInitFile	Zero-delimited string	A full path pointer to initialization file.
dwFlags	DWORD	Reserved; must be set to zero.
pvReserved	Void	Reserved; must be set to null.

Return Value

Boolean: If successful, returns **true**; otherwise, returns **false**.

ccollab_term()

Syntax

```
ccollab_term(IN HTRANSCC hCC);
```

Description

ccollab_term deallocates any memory associated with the initial **ccollab_init()** call.

Parameters

Name	Type	Description
hCC	HTRANSCC	The handle indicated by **ccollab_init**().

Return Value

Boolean: If successful, returns **true**; otherwise, returns **false**.

ccollab_translate()

Syntax

```
ccollab_translate(IN HTRANSCC hCC,
                  IN STC_BLOB *pInMsg,
                  OUT STC_BLOB *pReturnMsg,
                  IN DWORD dwFlags,
                  IN OUT OPTIONAL void *pvReserved);
```

Description

ccollab_translate() provides a pointer (pInMsg) string to external, accepts the message as a blob (pReturnMsg), and allocates memory as necessary.

Parameters

Name	Type	Description
hCC	HTRANSCC	The handle indicated by **ccollab_init**().
pInMsg	String	A read only pointer string.
pReturnMsg	String	A message, as a blob.
dwFlags	DWORD	Reserved; must be set to zero.
pvReserved	Void	Reserved; must be set to null.

Return Value

Boolean: If successful, returns **true**; otherwise, returns **false**.

5.4 Using the C Collaboration Service

Once the **.c** file has been compiled externally, the resultant **.dll** file must be committed to the run-time environment e*Gate Registry under the **bin/** directory.

You can commit files to the Registry either using the Enterprise Manager—on **theFile** menu, click **Commit to Sandbox**—or by using the **stcregutil.exe** command-line utility.

The example in this section demonstrates committing/retrieving files by using **stcregutil**, implementing the **-fr** and **-fc** commands. The example is printed on more than one line for clarity, but must be issued as a single command line.

To commit (import) the new .dll file to the bin/ path within the e*Gate Repository

1 Create a control (**.ctl**) file with a text editor, such as **mydll.ctl**. Each line must have the following format:

```
new.dll,path_location,FILETYPE_BINTEXT
```

Note: *There must not be any spaces before or after the commas (“,”).*

2 Run the **stcregutil** utility by typing the following at the command line:

```
stcregutil -rh registry -rs schema -un user-name -up password -fc
path_location -ctl mydll.ctl
```

- **registry** — The registry name to which to commit the file.
- **schema** — The schema name to which to commit the file.
- **user-name** — The user name.
- **password** — The password.
- **path_location —** The path location.
- **mydll.ctl** — The name of the **.ctl** file being committed.

For more information about the **stcregutil.exe** command-line utility, see the *e*Gate Integrator System Administration and Operations Guide*. You can also retrieve/commit a file using the Enterprise Manager’s **File** menu options. See the Enterprise Manager’s online Help system for more information.

5.5 C Collaboration Rules and the Enterprise Manager

After you have committed your C **.dll** file to the Registry, an e*Gate Collaboration Rule must be defined using the Collaboration Rules Editor.

To create a C Collaboration Service

1 Commit the **.dll** or **.ctl** file to the e*Gate Registry.
2 Define the Event Types to which the C Collaboration will subscribe and publish.
3 Create the Collaboration Rules that use the C Collaboration Service.

4. Configure a Collaboration to use the C Collaboration Rule, and configure a BOB or e*Way to execute this Collaboration. See the next section for additional details.

Note: *You cannot execute the C* ***.dll*** *file within a function called by the communications component of an e*Way.*

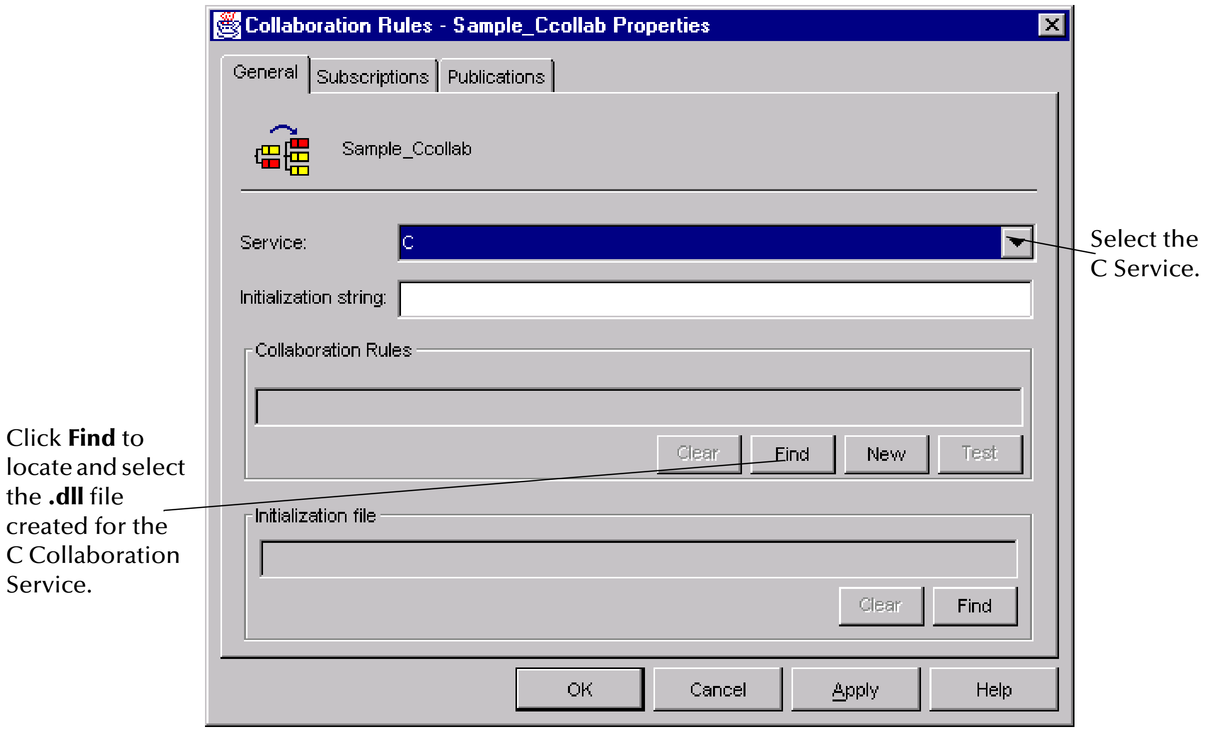
5. Configure any other e*Gate components as necessary to create a working schema.
6. Test the schema, making any correction as necessary to the e*Gate configuration or to any Collaboration Rules.
7. After the C **.dll** has been successfully tested, promote it to Run time. Use either the Enterprise Manager's **Promote to Runtime** command (a **File** menu option), or the **stcregutil.exe** command-line utility.

5.6 Implementing the C Collaboration Rule

To define a Collaboration Rule that uses the C Collaboration Service:

1. In the Enterprise Manager, define a Collaboration Rules component (see the Enterprise Manager's help system if you need assistance).
2. Set the Collaboration Rules properties as shown in Figure 8 below.

Figure 8 Collaboration Rules Properties Sheet


Collaboration Rules - Sample_Ccollab Properties

Sample_Ccollab

Service: C

Initialization string:

3 Click **Find** to select the name of the **.dll** file created for use with this Collaboration.

4 Configure the **Subscriptions** and **Publications** tabs as you would for any other Collaboration Rule.

5 Click **OK** to return to the Enterprise Manager.

Appendix A

The Java Collaboration Service Prior to 4.5

The Java Collaboration Service in versions of e*Gate Integrator before e*Gate Integrator Release 4.5 required you to manually code all Java code. This implementation continues to be supported, but is not recommended.

This chapter describes the prior implementation, using an extended example.

A.1 Developing the Java Business Logic Class

In the sample code in this example, **FileJCollab.java**, is a Java class you have created. The **.class** file must be imported into the schema in which the Java Collaboration Rule runs.

A.1.1 Sample Java Business Logic

Java Business Logic Classes use the following basic format as illustrated by the following sample. Each method created or defined for use with the Java Collaboration Service (JCS) must implement the **JCollaborator** class.

```
package com.stc.common.collabService;

/**
 *  A sample class to illustrate implementation of the JCollaborator
 *  interface.  A flat file is considered as the "external" system.
 *
 */

//  Java specific package imports

import java.io.*;

//  e*Gate specific package imports

import com.stc.common.collabService.*;

public class FileJCollab implements JCollaborator
{
  FileOutputStream fos = null;

  //  ----------------------------------------------------------------------

  /**
   *  Zero-argument constructor is needed (Java will provide one if not
   *  defined, but it's better to be explicit).
   *
   */
  public FileJCollab()
  {
    super();
  }

  //  ----------------------------------------------------------------------

  /**
   *  Called by the Java Collaboration Service (JCS) to inform this e*Gate
   *  collaboration that it has been loaded into the e*Gate system.  The
   *  applet can perform connection to externals as necessary here.
   *
   *  @exception com.stc.common.collabService.CollabConnException  thrown if
   *                            problem encountered with a connection
   *
   */
  public void initialize() throws CollabConnException
  {
    try
    {
      fos = new FileOutputStream(new File(System.getProperty("user.home"),
                                          "FileJCollab.txt"));
      fos.write("initialize(): we're here!".getBytes());
      fos.write(System.getProperty("line.separator").getBytes());
      fos.flush();
    }
    catch (IOException e)
    {
      throw new CollabConnException(e.getMessage());
    }
  }

  //  ------------------------------------------------------------------------

  /**
   *  Called by the JCS to translate an e*Gate collaboration subscribed event
   *  given as a byte blob.
   *
   *  @param    inputEvent      input event data given as a byte array
   *  @return                   output translated event as a byte array
   *  @exception com.stc.common.collabService.CollabConnException  thrown if
   *                            problem encountered with a connection
   *  @exception com.stc.common.collabService.CollabDataException  thrown if
   *                            problem encountered with data translation
   *
   */
  public byte[] translate(byte[] inputEvent)
              throws CollabConnException, CollabDataException
  {
    String inputString = new String(inputEvent);
    String outputString = inputString.toUpperCase();

    inputString = null;

    try
    {
      fos.write(inputEvent);
      fos.write(System.getProperty("line.separator").getBytes());
      fos.flush();
    }
    catch (IOException e)
    {
      throw new CollabConnException(e.getMessage());
    }

    return (outputString.getBytes());
  }

  //  ------------------------------------------------------------------------

  /**
   *  Called by the JCS to inform this e*Gate collaboration that it is
   *  being reclaimed and that it should destroy any resources that it
   *  has allocated.
   *
   */
  public void terminate()
  {
    try
    {
      fos.write("terminate(): we're done!".getBytes());
      fos.write(System.getProperty("line.separator").getBytes());
      fos.close();
    }
    catch (IOException e)
    {
      //  Since we're leaving, we don't care about errors
    }
  }
}
```

Note: *The above Java Business Logic Class transforms all data to upper case.*

Within the **translate** method, you can implement any code you like to perform the business logic required by this Collaboration. Compile your Java Business Logic Class using an IDE or Java's **javac** compiler, it must implement the **JCollaborator** interface.

A.1.2 Sample Java Class encode.java

The following class file incorporates the required Java Business Logic to shift characters one bit to the right.

```
//  Java specific package imports
import java.io.*;

//  e*Gate specific package imports
import com.stc.common.collabService.*;

//The encode class implements the JCollaborator interface (mandatory)

public class encode implements JCollaborator
{
      private String COPYRIGHT=
       "\nCopyright (c) 2001, SeeBeyond Technology Corporation, " +
       "All Rights Reserved\n";

    private String RCS_ID = COPYRIGHT + "$Id: $";
    public encode ()
    {
    }

    private String encodeStr( ByteArrayInputStream bais )
    {
      ByteArrayOutputStream baos = new ByteArrayOutputStream( 10 );
      try
      {
          int ascii = 1;
          while ( ascii != 0 )
          {
            ascii = (int)bais.read() + 1;
            if ( 256 == ascii )
            {
              baos.write( 0 );
            }
            else if ( 0 != ascii )
            {
              baos.write( ascii );
            }
          }
      }
      catch ( Exception e )
      {
        e.printStackTrace();
        System.err.println( "Caught exception in encodeStr '" + e.getMessage() + "'" );
      }
      return baos.toString();
    }

      public void initialize() throws CollabConnException
      {
            try
            {
          System.err.println( "Initialize function" );
            }
            catch ( Exception e )
            {
            e.printStackTrace();
            System.err.println( "Exception thrown in initialize:" + e.getMessage() );
                  throw new CollabConnException( e.getMessage() );
            }
      }

      public byte[] translate( byte[] inputEvent ) throws CollabConnException, CollabDataException
      {
        String outputString = "";
        ByteArrayInputStream tempar = new ByteArrayInputStream( inputEvent );
            try
            {
            outputString = encodeStr( tempar );
            }
            catch ( Exception e)
            {
            e.printStackTrace();
            System.err.println( "Exception thrown in translate:" + e.getMessage() );
                  throw new CollabDataException( e.getMessage() );
            }
            return ( outputString.getBytes() );
         }
      public void terminate()
      {

            try
            {
          System.err.println( "Terminate function" );
            }
            catch ( Exception e )
            {
            e.printStackTrace();
            System.err.println( "Exception thrown in terminate:" + e.getMessage() );
            }
      }

      public static void main( String[] args )
      {
      encode totest = new encode();

      try
        {
        System.out.println( new String( totest.translate( "beginning test 
\n\n\"abc\"\nend.".getBytes() ) ) );
        }
        catch ( Exception e )
        {

        }
      }

 }

 /* STC_LOG
  ******************************************************************************
  ** $Log: $
  ******************************************************************************
  ** STC_LOG */
```

Note: *Since the JCS does not support* `system.out.println`*, the above code uses* `print.err.println`*. If the unsupported method is invoked, no print statement will result.*

A.1.3 Sample Java Class decode.java

The following class file encorporates the required Java Business Logic to shift characters on bit to the left.

```
//  Java specific package imports
import java.io.*;

//  e*Gate specific package imports
import com.stc.common.collabService.*;

//The decode class implements the JCollaborator Interface (mandatory)
public class decode implements JCollaborator
{
      private String COPYRIGHT=
       "\nCopyright (c) 2001, SeeBeyond Technology Corporation, " +
       "All Rights Reserved\n";

    private String RCS_ID = COPYRIGHT + "$Id: $";
    public decode ()
    {
    }

    private String decodeStr( ByteArrayInputStream bais )
    {
      ByteArrayOutputStream baos = new ByteArrayOutputStream( 10 );
      try
      {
          int ascii = 1;
          while ( ascii >= 0 )
          {
            ascii = (int)bais.read();
            if ( 0 == ascii )
            {
              baos.write( 255 );
            }
            else if ( -1 != ascii )
            {
              baos.write( --ascii );
            }
          }
      }
      catch ( Exception e )
      {
        e.printStackTrace();
        System.err.println( "Caught exception in decodeStr '" + e.getMessage() + "'" );
      }
      return baos.toString();
    }

      public void initialize() throws CollabConnException
      {
            try
            {
          System.err.println( "Initialize function" );
            }
            catch ( Exception e )
            {
            e.printStackTrace();
                  throw new CollabConnException( e.getMessage() );
            }
      }

      public byte[] translate( byte[] inputEvent ) throws CollabConnException, CollabDataException
      {
        String outputString = "";
        ByteArrayInputStream tempstream = new ByteArrayInputStream( inputEvent );
            try
            {
            outputString = decodeStr( tempstream );
            }
            catch ( Exception e)
            {
            e.printStackTrace();
            System.err.println( "Exception thrown in translate:" + e.getMessage() );
                  throw new CollabDataException( e.getMessage() );
            }
            return ( outputString.getBytes() );
         }

      public void terminate()
      {

            try
            {
          System.err.println( "Terminate function" );
            }
            catch ( Exception e )
            {
            e.printStackTrace();
            }
      }

      public static void main( String[] args )
      {
      decode totest = new decode();

      try
        {
        System.out.println( new String( totest.translate( "cfhjoojoh!uftu!\013\013#bcd#\013foe/
".getBytes() ) ) );
        }
        catch ( Exception e )
        {

        }
      }

 }

 /* STC_LOG
  ********************************************************************************
  ** $Log: $
  ********************************************************************************
  ** STC_LOG */
```

A.2 Using the Java Collaboration Service

Once the Java class has been compiled by the external IDE, the resultant class files must be committed to the e*Gate Registry under the **classes/** directory. Additionally, any other supporting Java classes must be compiled and stored in **.jar** files. These **.jar** files must also be committed to the e*Gate Registry under the **classes/** directory.

You normally commit files to the Registry using Enterprise Manager: On the **File** menu, click **Commit to Sandbox**. However, you can also commit files using the **stcregutil.exe** command-line utility.

The following example demonstrates that you can also commit files by running **stcregutil** with the **-fc** (file commit) flag. The example is printed on more than one line for clarity, but must be issued as a single command line.

To commit the file new.jar to the classes/path within the e*Gate repository:

1. Create a control (**.ctl**) file with a text editor giving it a name such as **myjar.ctl**. Each line within **myjar.ctl** should have the following format:

   new.jar,classes,FILETYPE_BINTEXT

Note: *There must mot be any spaces before or after the commas (,).*

2. Run the **stcregutil** utility by typing the following at the command line:

   ```
   stcregutil -rh registry -rs schema -un user-name
      -up password -fc classes -ctl myjar.ctl
   ```

   For more information about the **stcregutil.exe** command-line utility, see the *e*Gate Integrator System Administration and Operations Guide*.

You can also retrieve or commit one file at a time using other **File** menu options. For more information, see the online help for Enterprise Manager.

A.3 Java Collaboration Service Methods

In order for the JCS to utilize your Java class, which performs the requisite business logic, the class must implement the SeeBeyond Java Interface:

com.stc.common.collabService.JCollaborator

The **JCollaborator** Interface prescribes implementation for the following methods:

initialize() on page 50

terminate() on page 50

translate() on page 50

initialize()

Syntax

```
void initialize()
```

Description

initialize() initializes the Java collaboration class and can perform functions such as connecting to externals as necessary.

Parameters

None.

Return Value

None.

Throws

com.stc.common.collabService.CollabConnException: indicates a problem encountered with a connection.

terminate()

Syntax

```
void terminate()
```

Description

terminate() notifies the Collaboration that it is no longer in use and that any connection type resources allocated to that Collaboration should be destroyed.

Parameters

None.

Return Value

None.

translate()

Syntax

```
byte[] translate(byte[] inputEvent);
```

Description

translate() translates a subscribed e*Gate Collaboration Event which must be input as a byte blob.

Parameters

Name	Type	Description
inputEvent	byte array	The Event to translate

Return Value

byte array

Throws

com.stc.common.collabService.CollabConnException: indicates a problem encountered with a connection.

com.stc.common.collabService.CollabDataException: indicates a problem encountered with data translation.

Additional Information

The byte array must be in the form of UTF-8 encoded characters (similar to the ASCII seven-bit characters, where the character values are between 1 and 127 and are represented as the same).

Index