SeeBeyond ICAN Suite

e*Gate Integrator Collaboration Services Reference Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the introduction, general purpose and scope, and organization of the *e*Gate Integrator Collaboration Services Reference Guide*. 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 guide describes each Collaboration Service and discusses how to select and implement the service in a production environment.

This guide explains the following Collaboration Services:

- Monk
- Pass Through
- Java
- C

Important: *Any operational explanations in this guide are generic, for reference purposes only, and do not necessarily address the specifics of configuring individual Collaboration Services.*

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 Integrator system. This person must also have expert-level knowledge of Windows XP/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 has five chapters and one appendix:

- **Chapter 1 "Introduction" on page 7**—Provides a general preview of this document, its purpose, scope, and organization.
- **Chapter 2 "Requirements for Supported Collaboration Services" on page 9**—Provides an overview of the system requirements for the Collaboration Services that the e*Gate Integrator system supports.
- **Chapter 3 "Monk-Related and Pass Through Collaboration Services" on page 12**—Describes the Monk and Pass Through Collaboration Services, including the Monk-related services.
- **Chapter 4 "Java Collaboration Service (JCS)" on page 14**—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.

1.4 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 Collaboration Services

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

2.1 Supported Collaboration Services

e*Gate Integrator supports seven Collaboration Services:

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

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 beyond those required by a standard e*Gate Integrator installation.

- All Collaboration Services require an e*Gate Integrator Participating Host version 5.0 SRE or later.
- For information on downloading 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 Java 2 SDK Requirements on UNIX Systems

Important:

- Do not move Java 2 SDK to any other location. It must remain where it was installed by the SeeBeyond eBusiness Integration Suite Setup program.

  The installation process enters the Java 2 SDK location into the UNIX Object Data Manager (ODM). Changing the location prevents the proper execution of the Java JNI dynamic-link library (DLL) needed by the JCS.
- The user environment on the Participating Host must have the DLL search path environment variable set to include *all* Java 2 SDK directories that contain shared libraries.

  The variable names and shared library file extensions of these vary, according to the operating system, as shown in Table 1.

Table 1 Java 2 SDK DLL Search Path Environment Variables

On these operating systems . . .	The DLL search path environment variable is . . .	And the file extension is . . .
Compaq *Tru64* V4.0F or 5.0A Solaris 2.6, 7, or 8 Red Hat Linux 6.2	LD_LIBRARY_PATH	.so
HP-UX 11.0 or 11i	SHLIB_PATH	.sl
AIX 4.3.3	LIBPATH	.a

For AIX Participating Hosts Only:

If certain Program Temporary Fixes (PTFs) are not installed, then the LIBPATH environment variable must be set to:

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

If, for example, Java 2 SDK 1.3 is installed under **/usr/java_dev2**, then:

- For the Bourne or Korn shell, add this line into the **egateclient.sh** file, *immediately before* the **export LIBPATH** statement:

```
LIBPATH=/usr/java_dev2/jre/bin:/usr/java_dev2/jre/bin/
classic:$LIBPATH
```

- For the C shell, add this line *immediately after* the statements that set LIBPATH:

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

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

Chapter 3

Monk-Related and Pass Through Collaboration Services

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

The two Monk-related Collaboration Services are:

- Monk ID Collaboration Service
- Route Table Collaboration Service

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 15***.*

3.1 Monk Collaboration Service

The Monk Collaboration Service supports the development of business logic and other e*Gate Integrator components using the proprietary Monk language.

Monk files can be created with any text editor such as Notepad or vi, or by using the e*Gate Collaboration Rules Editor in the Schema Designer.

Most Monk files should be given the .monk extension.

Files created by the Monk Collaboration Rules Editor, which are used to transform data within a Collaboration, are assigned the .tsc extension by default. Do not confuse these with the standard extensions for Monk ETDs (.ssc) and for Java-enabled ETDs (.xsc).

3.2 Monk ID Collaboration Service

The Monk ID Collaboration Service supports the execution of Collaboration-ID Rules.

In versions of e*Gate Integrator earlier than release 4.0, Collaboration-ID Rules were commonly used to validate inbound Events.

In release 4.0 and later:

- Perform all such validation within a standard Collaboration Rule.
- Use Collaboration-ID Rules only for backward compatibility with prior versions of e*Gate Integrator.

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 file extension is .isc.

3.3 Route Table Collaboration Service

The Route Table Collaboration Service is used for upgrading from e*Gate version 3.*x*, as described in the *e*Gate Integrator Upgrade Guide*.

3.4 Pass Through Collaboration Service

The Pass Through Collaboration Service supports copying input Events to output Events, leaving the Event contents unchanged. It performs a byte-for-byte copy for all data that it processes, and requires no Collaboration Rules.

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* **"To create a Java Pass Through Collaboration Rules component" on page 15**.

Chapter 4

Java Collaboration Service (JCS)

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

4.1 Description

The Java Collaboration Service (JCS) supports using a Java class to implement the business logic that transforms Events as they move through the e*Gate Integrator. When data passes through e*Gate using a Java Collaboration, the JCS instantiates a Java Virtual Machine (JVM) and uses the associated Java Collaboration Rules class to accomplish the data transformation. (See **Chapter 1** for requirements specific to Java 2 SDK.)

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

4.2 Using the Java Collaboration Service

To use the Java Collaboration Service:

1. Create a Collaboration Rule and select Java as the service.
2. Using the Java Collaboration Rules Editor, add the rules and logic between the instances of previously defined Event Type Definitions (ETDs).
3. Compile the Collaboration Rules to create a Java Collaboration Rules class and all required support files.

The resulting Java class implements the data transformation logic.

Note: *Though it is not recommended, you can manually create .class files that use the Java Collaboration Service without using the Java Collaboration Rules Editor.*

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.*

4.2.1 Creating Java Collaboration Rules Components

When creating a new Collaboration Rules component, you must specify inbound and outbound Event Type instances and the rules for transforming the data between them. See **"To create a Java Collaboration Rules component" on page 16**.

The Java Pass Through Collaboration Rule, however is a predefined Collaboration Rules component that, like the Pass Through Collaboration Service, transports data without transforming it—there is no need to specify data transformation rules.

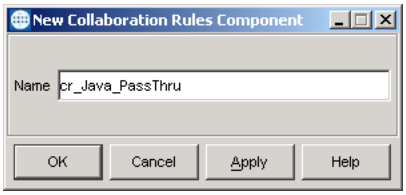
Unlike the Java Pass Through Collaboration Service, however, you can use the Java Pass Through Collaboration Rules component to communicate with e*Way Connections. You are neither required nor permitted to specify instance names or initialization strings.

The following procedures summarize the detailed procedures in the *e*Gate Integrator User's Guide*.

To create a Java Pass Through Collaboration Rules component

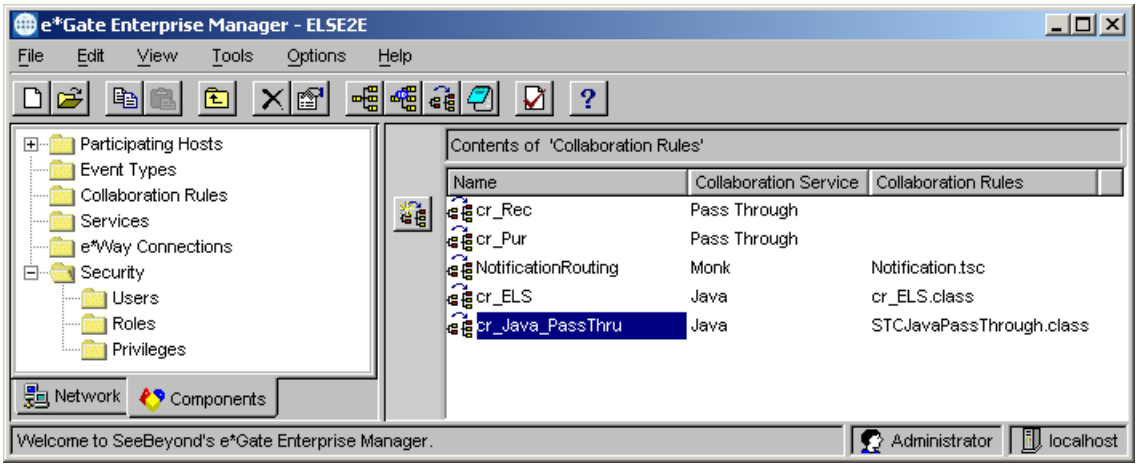
1. Use e*Gate Schema Designer to create and name a new Collaboration Rules component, as shown in Figure 1.

Figure 1 Creating a New Collaboration Rules Component



2. Edit the properties of the new Collaboration Rules component.
3. In the **Collaboration Rules - Properties** dialog box, click **Find** in the Collaboration Rules part of the **General** tab.
4. Navigate to the **collaboration_rules\STCLibrary** folder.
5. Click **STCJavaPassThrough.class** and click **Select**.
6. In the **Properties** dialog box, click **OK** to add **STCJavaPassThrough.class** as the Collaboration Rules as shown in Figure 2.

Figure 2 Adding the STCJavaPassThrough.class Collaboration Rules

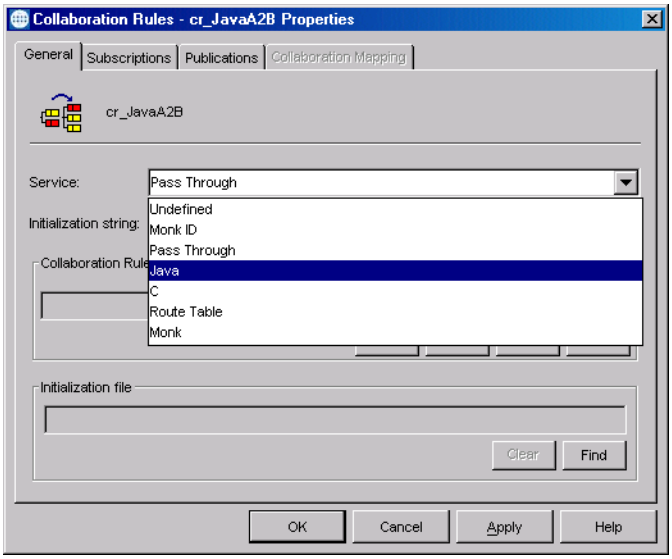


To create a Java Collaboration Rules component

1 Use e*Gate Schema Designer to create and name a new Collaboration Rules component. See **Figure 1 on page 15**.

2 Edit the properties of the new Collaboration Rules component. If necessary, select **Java** as the Collaboration Service on the **General** tab as shown in Figure 3.

Figure 3 Selecting the Java Collaboration Service

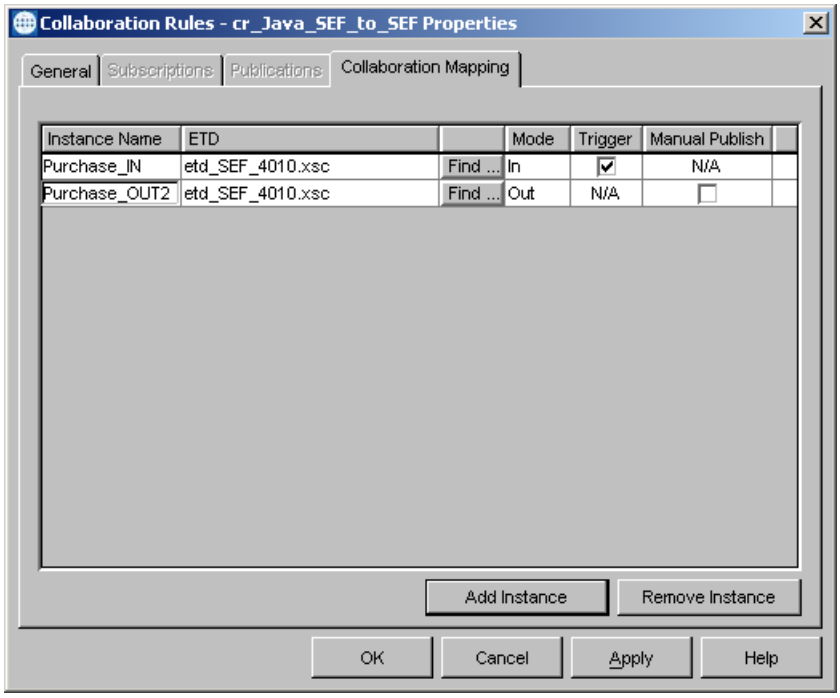


3 Open the **Collaboration Mapping** tab and click the **Add Instance** command button to add a new instance (see Figure 4).

4 Enter a name in the **Instance Name** field.

The Collaboration Rules Editor uses the Instance Name to identify the source and destination Events.

Figure 4 Adding Collaboration Instances



5 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.

6 In the **Mode** list, click **In** or **In/Out**.

7 Optionally, repeat steps 3 through 6 to create additional source Event instances.

8 As necessary, select the **Trigger** check box for one or more inbound Events.

9 Repeat steps 3 through 5 for each destination instance.

10 In the **Mode** list, click **Out** or **In/Out** for each destination instance.

11 You can select the **Manual Publish** check box for zero or more outbound Events.

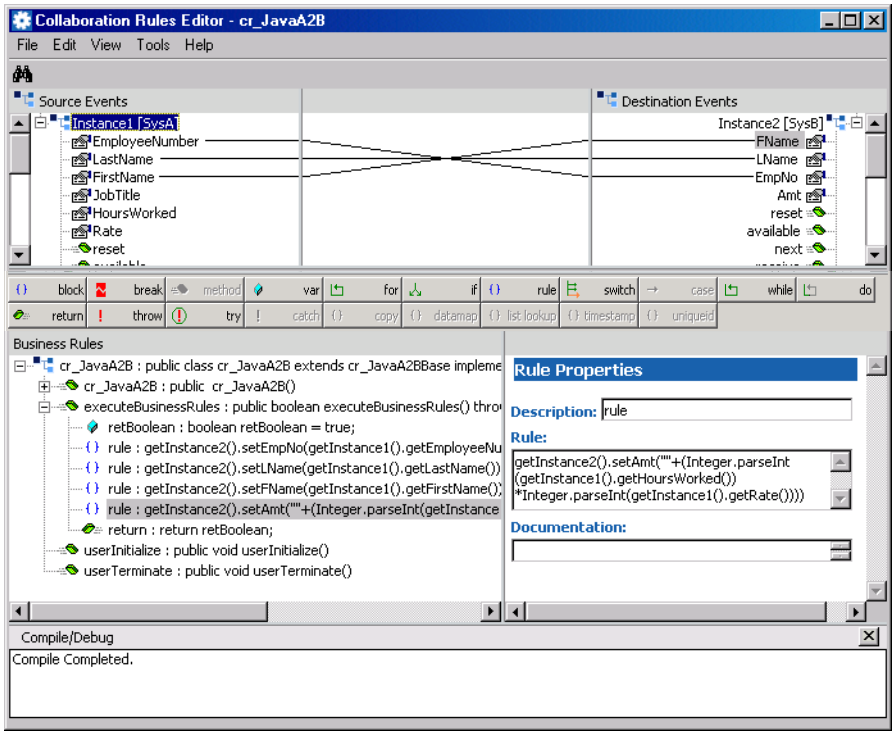
12 Click **Apply** to save the changes.

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

14 Click the **General** tab.

15 Click the **New** command button in the Collaboration Rules part of the **General** tab to open the Collaboration Rules Editor window (Figure 5).

Figure 5 Adding the Business Logic (Java Code)



16 Use the Collaboration Rules Editor to add the required business logic (Java code) for this rule to the **executeBusinessRules()** method.

The Collaboration Rules Editor 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

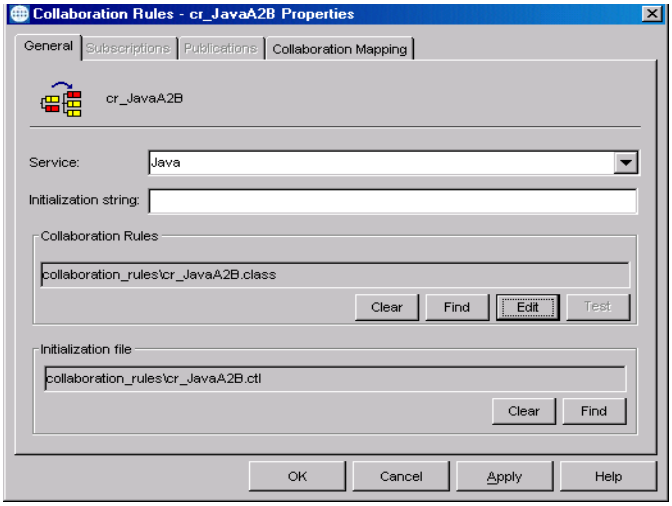
17 For information about using the Collaboration Rules Editor, use its online help or refer to the Chapter 7 of the *e*Gate Integrator User's Guide*.

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

19 After the Collaboration Rules class compiles cleanly, promote it and close the Collaboration Rules Editor to enter the .class and the corresponding control file (.ctl) in the Collaboration Rules and Initialization file fields of the **Collaboration Rules - Properties** dialog box, as shown in Figure 6.

20 These are the files that the Collaboration Rules component uses to perform the required data transportation and transformation functions.

Figure 6 Collaboration Rules - Properties Dialog Box



4.2.2 Implementing Java Collaboration Rule Components

To define a Collaboration Rule that uses the JCS

1. Use e*Gate Schema Designer to create and name a new Collaboration Rules component. See **Figure 1 on page 15**.
2. Edit the properties of the new Collaboration Rule. If necessary, select **Java** as the Collaboration Service on the **General** tab as shown in **Figure 3 on page 16**.
3. Enter any required JCS initialization parameters in the **Initialization string** box.

   For example:

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

   Table 2 on page 23 provides a list of the JCS initialization parameters.

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

   For example:

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

Note: *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** 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** box.

For more information, see **"To commit the file new.jar to the classes/path within the e*Gate repository" on page 21**.

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, with all periods (**.**) converted to forward slashes (**/**).

The path for this class name, for example:

```
com.stc.common.collabService.myClassFile
```

Is:

```
classes/com/stc/common/collabService/myClassFile
```

4.2.3 Dealing With Long CLASSPATH Parameters

In some instances, the classpath parameter can exceed 255 characters. There are several ways to accommodate this:

- The JCS automatically uses the CLASSPATH environmental variable. You can refer to all .jar files and directories here, allowing for a hard-coded 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 the JCS must also run on different Participating Hosts, then you can check all .jar, .zip, and .class files into the e*Gate Registry and reference them from a Registry Control file. The Registry Control file can then be entered into the **Initialization file** text box of the **Collaboration Rules - Properties** dialog box. 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 compiling the Java Collaboration Rules, you must:

- Commit the resultant .class files to the e*Gate Registry under the **classes/** directory.
- Compile and store any other supporting Java classes 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 Schema Designer by clicking **Commit to Sandbox** on the **File** menu. However, you can also commit files using the **stcregutil.exe** command-line utility.

The following example shows how you can also commit files by running stcregutil with the -fc (file commit) flag.

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 the **myjar.ctl** file should have the following format, with no spaces before of after the comma (,) characters:

   ```
   new.jar,classes,FILETYPE_BINTEXT
   ```

2. Run the stcregutil utility by typing the following—on a single line—at the command prompt:

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

Where . . .	Is the . . .
registry	Registry name to which to commit the file.
schema	Schema name to which to commit the file.
user_name	User name
password	Password.
myjar	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 commit one file at a time using other **File** menu options, as described in the following subsection.

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 Schema Designer to make changes. If you place your changes at the end of the file and then immediately recommit the .ctl file, your changes are preserved.

To make permanent edits to a .ctl file

1. In Schema Designer, on the **File** menu, click **Edit File** to display the **Open file** dialog box.
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, and answer **Yes** when the system prompts you to Commit the file.
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
-appendenvcp	String	Specifies any string or environment variable to append the current CLASSPATH used by JCS. The **-appendenvcp** parameter is preferred over the **-classpath** parameter because it will not override any necessary paths.
-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. 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 **-appendenvcp** or **-cp** parameters.
-cp	String	Specifies any string or environment variable to prepend the current CLASSPATH used by JCS. The **-cp** parameter is preferred to **-classpath** because it will not override any necessary paths.
-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 or tracing and finding errors in Collaboration Rules.
-def	String	Defines a Java property using the following format: `-def propertyname = value`

Table 2 JCS Initialization Parameters (Continued)

<table>
<tr><th>Parameter</th><th>Value</th><th>Purpose</th></tr>
<tr><td>-java1</td><td>None</td><td>Specifies that the version of the jnidll location is for Java version 1.1.7b.

If unspecified, the jnidll specified is assumed to before Java 2:
<table><tr><td>Windows</td><td>javai.dll</td></tr><tr><td>Solaris, Linux, Compaq</td><td>libjava.so</td></tr><tr><td>HP</td><td>libjava.sl</td></tr><tr><td>AIX</td><td>libjava.a</td></tr></table></td></tr>
<tr><td>-jnidll</td><td>An absolute path, or an environmental variable</td><td>Specifies the location of the (Java Native Interface (JNI) dynamic-link library (DLL).

The absolute path name of where the installed Java JNI DLL is located varies on different OS platforms:
<table><tr><th>OS Platform</th><th>Java 2</th><th>Java 1</th></tr><tr><td>Windows 2000</td><td>jvm.dll</td><td>avai.dll</td></tr><tr><td>Solaris 2.6, 7, or 8
Compaq Tru64
Linux</td><td>libjvm.so</td><td>libjava.so</td></tr><tr><td>HP-UX 11.0 or 11i</td><td>libjvm.sl</td><td>libjava.sl</td></tr><tr><td>AIX 4.3.3</td><td>libjvm.a</td><td>libjava.a</td></tr></table>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.</td></tr>
<tr><td>-jvmprops</td><td>A file name, including a path relative to the Registry</td><td>Specifies the location of a single file that contains all of the JVM property options. See the subsection “Using the Java Virtual Machine (JVM) Properties File” in the e*Gate Integrator User’s Guide for information about creating and using a JVM properties file.</td></tr>
</table>

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-link path is configured automatically by the JCS. This parameter prepends the specified paths to the link-library path used. You could, for example, use this parameter when Java code contains IDS-out-wrapper classes that need a specified link-library file.
-ms	Integer	Specifies the initial/starting heap size of the Java Virtual Machine (JVM) in bytes. The default value is 32000000. You must use the **-ms** parameter to increase the starting heap size if your process passes messages larger than 32,000,000 bytes. ***Caution:*** The **-ms** parameter impacts the performance of your JCS. Inappropriate settings can produce Java OutOfMemoryError conditions, and setting the **-ms** value greater than the **-mx** value produces an incompatible-size error message. Consult the Java 2 SDK reference documentation and evaluate the system performance implications before setting the **-ms** and **-mx** parameters.
-mx	Integer	Specifies the maximum heap size of the Java Virtual Machine (JVM) in bytes. The default value is 64000000. ***Caution:*** The **-mx** parameter impacts the performance of your JCS. Inappropriate settings can produce Java OutOfMemoryError conditions, and setting the **-mx** value less than the **-ms** value produces an incompatible-size error message.\ Consult the Java 2 SDK reference documentation and evaluate the system performance implications before setting the **-ms** and **-mx** parameters.
-noclasgc	None	Disables class garbage collection. If this parameter is in place, no memory deallocation will take place for the JVM.
-nojit	None	Disables the just-in-time compiler.
-suspend	None	When **-suspend** is specified, the JVM waits for an attach to occur before executing the Collaboration Rules component.
-verbose	None	Reports JVM information and all class loads.
-verbosegc	None	Enables garbage collection console activity.

Table 2 JCS Initialization Parameters (Continued)

Parameter	Value	Purpose
None—A valid path name, entered as the last value in the **Initialization string** box, for example: com.mystc.myProgram		Specifies the location of the Collaboration Rules file. When you use the last value of the initialization string to specify the Collaboration Rules file, you do not need to indicate a Collaboration Rules component in the **Collaboration Rules** box.

Chapter 5

C Collaboration Service

The C Collaboration Service supports using the C and C++ programming languages to write dynamic-link library (DLL) files. When selected via the GUI from the **Collaboration Rules** dialog box, e*Gate Integrator 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 Integrator applications.
- You may want to implement a portion of time-critical code, written in C or C++ , and have an e*Way call these functions.
- You may have application-specific situations that are better handled outside the Monk programming environment.

5.1 Header File: HTRANSCC.h

This section describes the **HTRANSCC.h** header file that passes a string into and out of the interface object.

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

The external interface object implements the external interfaces. A structure is defined that provides a location for object data and 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 following **newcollab.c** (or .cpp) file shows 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

ccolab_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

This subsection describes the e*Gate Integrator APIs that are available within the C Collaboration Service:

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	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 the 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

After 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 using either the:

- Schema Designer (click **Commit to Sandbox** on the **File** menu)
- **stcregutil.exe** command-line utility

The example in this section shows how to commit/retrieve files by using **stcregutil**, implementing the **-fr** and **-fc** commands.

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

1. Use a text editor to create a control (.ctl) file, such as **mydll.ctl**. Each line in this file must have the following format, with no spaces before or after the comma (,) characters:

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

2. Run the **stcregutil** utility by typing the following—on a single line—at the command prompt:

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

Where . . .	Is the . . .
registry	Registry name to which to commit the file.
schema	Schema name to which to commit the file.
user_name	User name
password	Password.
path_location	Path location.
mydll	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 Schema Designer's **File** menu options. See the Schema Designer online Help system for more information.

5.5 C Collaboration Rules and the Schema Designer

After you have committed your C .dll file to the Registry, you must use the Collaboration Rules Editor to define an e*Gate Collaboration Rules component.

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, as described in the following subsection.

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

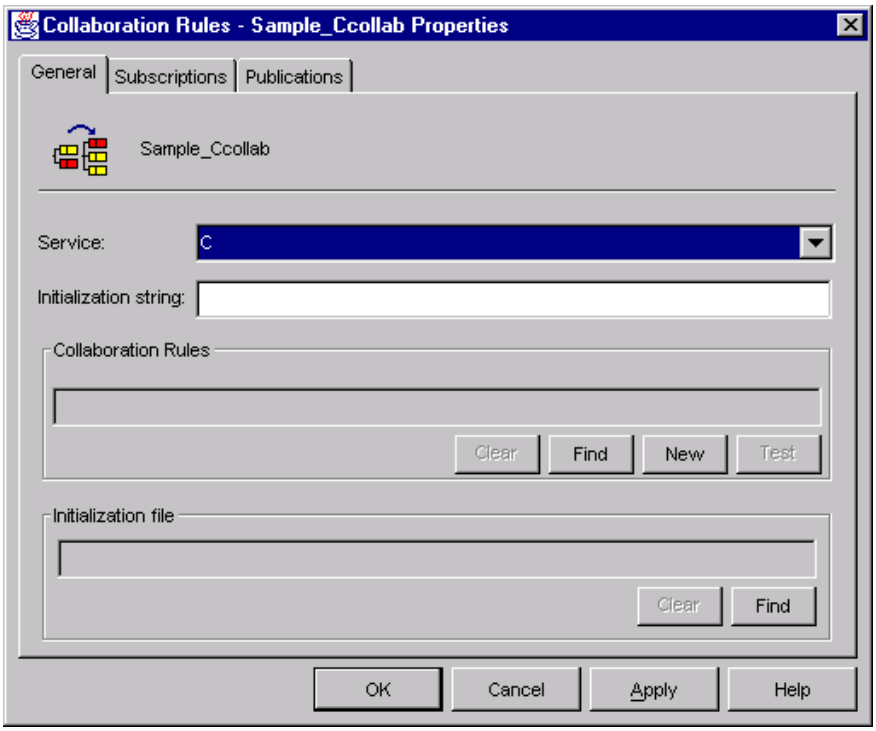
5. Configure any other e*Gate Integrator components as necessary to create a working schema.
6. Test the schema, making any correction as necessary to the e*Gate Integrator configuration or to any Collaboration Rules components.
7. After successfully testing the C .dll, promote it to the run-time environment, using either the:
   - **Promote to Runtime** command on the Schema Designer **File** menu
   - **stcregutil.exe** command-line utility

5.6 Implementing the C Collaboration Rule

To define a Collaboration Rule that uses the C Collaboration Service

1. Use e*Gate Schema Designer to create and name a new Collaboration Rules component. See **Figure 1 on page 15**.
2. Edit the properties of the new Collaboration Rule, selecting **C** as the Collaboration Service on the **General** tab of the **Collaboration Rules - Property** dialog box, as shown in Figure 7.
3. Click **Find** in the Collaboration Rules part of the dialog box to select the name of the .dll file that you created to use with this Collaboration.
4. Configure the **Subscriptions** and **Publications** tabs as you would for any other Collaboration Rules component.
5. Click **OK** to close the **Collaboration Rules - Properties** dialog box and return to the Schema Designer.

Figure 7 Selecting the C Collaboration Service



Index

Symbols

A

C

D

F

H

I

J

L

M

N

O

P

R

S

V