SeeBeyond ICAN Suite

e*Way Intelligent Adapter for COM/DCOM User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Version 20050406084157.

Contents

Chapter 1

Introduction

The e*Way Intelligent Adapter for COM/DCOM enables the e*Gate system to exchange data with server side COM/DCOM-enabled applications and components. This document describes how the Java™-enabled COM/DCOM e*Way is installed, configured, and implemented.

1.1 COM/DCOM Overview

The Microsoft™ *Component Object Model* (COM) is a component software architecture that allows applications and systems to be built using separate components. COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE (Object Linking and Embedding). OLE services span various aspects of component software, including compound documents, custom controls, inter-application scripting, data transfer, and other software interactions. Using COM allows software objects to be reused for a variety of applications. Because of its binary standard, COM allows any two components to communicate regardless of the language in which they were written.

The Microsoft *Distributed Component Object Model* (DCOM) is an extension of COM, and supports communication among objects residing on different computers; LANs, WANs, and the Internet. With DCOM, these software objects can be reused over a distributed environment.

COM objects or components are individual modular software routines that can be reused within applications. COM objects are reusable compiled binary objects, as opposed to reusable sections of code. The COM objects create *handles* that provide access to the COM-enabled applications.

The COM/DCOM e*Way enables configurable transparent data exchange with COM/DCOM enabled applications and components.

1.2 Intended Reader

The reader of this guide is presumed:

- to be a developer or system administrator with responsibility for maintaining the e*Gate system.

- to have a working knowledge of Windows operations and administration.
- to be thoroughly familiar with COM/DCOM, Java, and Windows-style GUI operations.

1.3 Supported Operating Systems

The Java-enabled COM/DCOM e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003

1.4 System Requirements

To use the COM/DCOM e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection
- Additional disk space for e*Way executable, configuration, library, and script files. The disk space is required on both the Participating and the Registry Host. Additional disk space is required to process and queue the data that this e*Way processes; the amount necessary varies based on the type and size of the data being processed, and any external applications performing the processing.

The client components of the external systems with which the e*Way interfaces, have their own requirements; see those systems' documentation for more details.

Note: *Open and review the **Readme.txt** for the COM/DCOM e*Way regarding any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM at setup\addons\ewims.*

1.4.1. External Application Requirements

The COM/DCOM e*Way requires an existing COM/DCOM-compliant application or component.

Chapter 2

Installation

This chapter contains pre-installation requirements and the procedures for installing the Java-enabled COM/DCOM e*Way. In addition, a list of installed files is provided, along with the directories where these files are located.

2.1 Windows Installation

2.1.1. Pre-installation

- Quit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this e*Way.

2.1.2. Installation Procedure

To install the COM/DCOM e*Way on Windows

1. Log in as an Administrator to the workstation on which you are installing the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's Autorun feature is enabled, the setup application launches automatically; skip ahead to step 4. Otherwise, use Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application launches. Follow the installation instructions until you come to the **Please choose the product to install** dialog box.
5. Select **e*Gate Integrator**, then click **Next**.
6. Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.
7. Clear the check boxes for all selections except **Add-ons**, and then click **Next**.
8. Follow the on-screen instructions until you come to the **Select Components** dialog box.

9 Highlight (but do not check) **e*Ways**, and then click the **Change** button. The **SelectSub-components** dialog box appears.

10 Select the **COM/DCOM e*Way**. Click the continue button to return to the **Select Components** dialog box, then click **Next**.

11 Follow the rest of the on-screen instructions to install the Java-enabled COM/DCOM e*Way. Be sure to install the e*Way files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory. ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.***

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

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

2.2 Files/Directories Created by the Installation

The Java-enabled COM/DCOM e*Way installation process installs the following files within the e*Gate "client" directory tree. Files are installed within the "egate/client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files Created by the Installation

e*Gate Directory	File(s)
bin\	stccombuilder.exe stccomjavabuilder.exe stccomutil.dll
bin\java	gnu-regexp-1.1.1.jar jcscomp.jar stcjintegra.jar xerces.jar
classes\	ewscom.jar ewmscom.jar stcjcs.jar
configs\MSCOM\	com.def
etd\	com.ctl comwizard.ctl stcewmscom

Chapter 3

Multi-Mode e*Way Configuration

This chapter describes how to configure the Multi-Mode e*Way.

3.1 Multi-Mode e*Way

Multi-Mode e*Way properties are set using the Schema Designer.

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

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

Multi-Mode e*Way Configuration Parameters

The Multi-Mode e*Way configuration parameters are arranged in the following sections:

- **JVM Settings** on page 10
- **General Settings** on page 14

3.1.1. JVM Settings

The JVM Settings control basic Java Virtual Machine settings.

- **JNI DLL Absolute Pathname** on page 11
- **CLASSPATH Prepend** on page 12

JNI DLL Absolute Pathname

Description

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

Required Values

A valid pathname.

Additional Information

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

OS	Java 2 JNI DLL Name
Windows	jvm.dll

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

```
%MY_JNIDLL%
```

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

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

CLASSPATH Prepend

Description

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

Required Values

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

Additional Information

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

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

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

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

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

Required Values

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

Additional Information

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

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

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

Required Values

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

Initial Heap Size

Description

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

Required Values

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

Maximum Heap Size

Description

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

Required Values

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

Maximum Stack Size for Native Threads

Description

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

Required Values

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

Maximum Stack Size for JVM Threads

Description

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

Required Values

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

Disable JIT

Description

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

Required Values

YES or **NO**.

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

Remote Debugging port number

Description

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

Required Values

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

Suspend option for debugging

Description

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

Required Values

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

Auxiliary JVM Configuration File

Description

Specifies an auxiliary JVM configuration file for additional parameters.

Required Values

The location of the auxiliary JVM configuration file.

3.1.2. General Settings

For more information on the General Settings configuration parameters see the *e*Gate Integrator User's Guide*. The General Settings section contains the following parameters:

- **Rollback Wait Interval** on page 14
- **Standard IQ FIFO** on page 15

Rollback Wait Interval

Description

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

Required Values

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

Standard IQ FIFO

Description

Specifies whether the highest priority messages from all STC_Standard IQs will be delivered in the first-in-first-out (FIFO) order.

Required Values

Select **YES** or **NO**. YES indicates that the e*Way will retrieve messages from all STC_Standard IQs in the first-in-first-out (FIFO) order. NO indicates that this feature is disabled. NO is the configured default.

Chapter 4

e*Way Connection Configuration

This chapter describes how to configure the COM/DCOM e*Way Connection.

4.1 Configuring e*Way Connections

e*Way Connections are set using the Schema Designer.

To create and configure e*Way Connections

1. In the Schema Designer's Component editor, select the **e*Way Connections** folder.
2. On the palette, click on the icon to create a new **e*Way Connection**.
3. The **New e*Way Connection Component** dialog box opens, enter a name for the **e*Way Connection**. Click **OK**.
4. Double-click on the new **e*Way Connection**. For this example, the connection has been defined as **com_connector**.
5. The **e*Way Connection Properties** dialog box opens.
6. From the **e*Way Connection Type** drop-down box, select **COM/DCOM**.
7. Enter the **Event Type "get"** interval in the dialog box provided. The configured default is 100 milliseconds. The "get interval is the intervening period at which the e*Way connection is polled.
8. From the **e*Way Connection Configuration File**, click **New** to create a new Configuration File for this e*Way Connection. (To use an existing file, click **Find**.)

The COM/DCOM e*Way Connection configuration parameters are organized into the following sections:

4.1.1. Connector

This section contains a set of top level parameters.

Type

Description

String-set. Specifies the connector type. Retain the default (COM) value for COM connections.

Required Values

COM. The value defaults to **COM**.

Class

Description

String-set. Specifies the class name of the COM/DCOM connector object.

Required Values

Retain the default value. The default is **com.stc.eways.com.ComEWayConnector**.

4.1.2. COM/DCOM Configuration

This section contains a set of top level parameters.

ProgID

Description

String-set. Specifies the programmatic ID that identifies the component to be used (for example: Excel.Application). Changes to this parameter in the ETD Editor Properties take presidence over the e*Way Connection parameter.

The ProgID is a Windows registry entry that uniquely identifies a program or a COM object. Unlike a Globally Unique Identifier (GUID), the ProgID is a humanly-readable alphanumeric string. The ProgID consists of three parts in this format:

```
vendor.component.version
```

The *vendor* parameter is the control's library, the *component*, it's class, and the *version* number, which is optional. If no version is given, the latest version of the application object is assumed. Each parameter is separated by periods and no spaces; for example the Visual Basic Command Button object's ProgID is:

```
VB.CommandButton
```

The ProgID for most objects can be determined by checking, for example, the Visual Basic Object Browser. The Windows registry entry reads as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\ProgID =
```

Required Values

A valid programmatic ID to be used by the component.

Hostname

Description

String-set. Specifies the name the remote server on which the DCOM object resides. Changes to this parameter in the ETD Editor Properties take presidence over the e*Way Connection parameter.

Required Values

A valid host name.

Note: *Hostname is only relevant to DCOM components (that is .EXEs). Do not use the Hostname value for .DLLs.*

Chapter 5

Implementation

This chapter includes information pertinent to implementing the Java-enabled COM/DCOM e*Way in a production environment. Information on implementing the sample schema is included.

The following assumptions are applicable to this implementation:

- The COM/DCOM e*Way has been successfully installed.
- The executable and the configuration files have been appropriately assigned.
- All necessary .jar files are accessible.
- The user has a working understanding of COM/DCOM concepts.

5.1 Implementation Notes

The COM/DCOM e*Way supports the following data types:

Table 2 Supported Data Types

OLE Data Types	Description
VT_I2	2-byte signed int
VT_I4	4-byte signed int
VT_R4	4-byte real
VT_R8	8-byte real
VT_BSTR	Binary string
VT_DISPATCH	IDispatch (1 dimentional array only)
VT_ERROR	4-byte error code
VT_BOOL	Boolean
VT_UI1	Unsigned char
VT_DATE	The standard COM DATE data type (8-byte real).

Table 2 Supported Data Types (Continued)

OLE Data Types	Description
VT_SAFEARRAY VT_I2 VT_I4 VT_R4 VT_R8 VT_BSTR VT_DISPATCH VT_ERROR VT_BOOL VT_UI1 VT_DATE VT_VARIANT	1 dimensional array (VT_VARIANT can support a 2 dimensional array)

All of the above types are supported for In and Out parameters.

The COM/DCOM e*Way does not supports the following data types:

Table 3 Non-Supported Data Types

OLE Data Types	Description
VT_CY	currency

Important: ***The code processes one dimensional SAFEARRAYs only.*** *The number of elements in all rows are expected to be equal. VT_SAFEARRAY has specific, limited functionality as used in the COM/DCOM e*Way.*

5.2 Considerations

- Currently the Java e*Way Intelligent Adapter for COM/DCOM does not support accessing a range of cells from Excel. Each cells must be accessed one by one.
- The method names used for created custom Com objects must not match a method name on the Java Object class. Duplicate names will result in compile-time errors indicating that a name collision has occurred. It is also recommended that you avoid methods names that are similar to those of any other Java object that may be used in your Collaboration.

5.3 COM/DCOM Sample Implementation Overview

During installation, the host and Control Broker are automatically created and configured. The default name of each is the name of the host on which you are installing the e*Gate Schema Designer GUI. To complete the implementation of the Java-enabled COM/DCOM e*Way, do the following:

- Make sure that the **Control Broker** is activated.

- In the e*Gate Schema Designer, define and configure the following as necessary:
  - **Trigger e*Way** using **stcewfile.exe**.
  - The **Multi-Mode e*Way** component as described in **Chapter 3**.
  - **Event Type Definitions** used to package the data to be exchanged with the external system.
  - **Collaboration Rules** to process Events.
  - The **e*Way Connection** as described in **Chapter 4**.
  - **Collaborations**, to be associated with each e*Way component, to apply the required Collaboration Rules.
  - The destination to which data is published.

The following sections describe how to define and associate each of the above components. The sample schema provided on the Installation CD-ROM is, for the most part, complete, once it is imported. The e*Way component implementation is provided for the purpose of explaining how each of these components are created manually.

5.3.1. Importing the Sample Schema

The first task in deploying the sample implementation is to create a new schema name. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install the COM/DCOM e*Way, do the following:

1. Start the e*Gate Schema Designer GUI.
2. When prompted to log in, select the host specified during installation, and enter the password.
3. When prompted to select a schema, click **New**.
4. Enter a name for the new schema. In this case, enter **ExcelSample**, or any appropriate name as desired.
5. Select **Create from export**, locate the **ExcelSample.zip** on the CD, and click **Open**.

The e*Gate Schema Designer opens to the new schema. It is also necessary to create the file **test.xls** in C:\temp. To create this file do the following:

1. Open Microsoft Excel.
2. Create a new worksheet called **test.xls**.
3. Save this file to the **c:\temp** directory and close **Excel**.

5.4 Creating the Sample Schema Manually

The following sections explain how to manually configure the e*Way components to create the same sample schema that is created when the sample is imported.

5.5 Creating Event Type Definitions

An ETD is a structural representation of an Event (that is, the blueprint of an Event).The COM/DCOM e*Way installation includes the COM Event Type Definition Wizard to assist in the quick creation for COM specific ETDs. In addition, if you have imported the sample schema, you have existing .xsc files that can be opened in the ETD Editor.

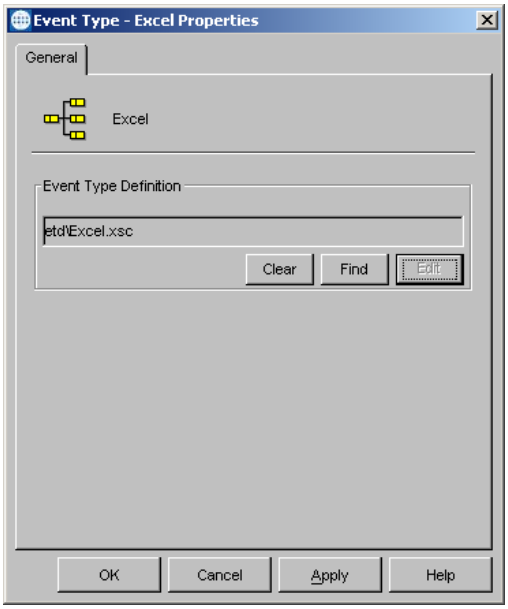
In COM, the Event Type corresponds to the object being used (it's methods). These methods are exposed in the Collaboration to allow the user to get and set the properties or call one or more methods. In addition, methods are made available to get and set the objects properties.

5.5.1. Creating an Event Type Definition Using an Existing .xsc

For the purpose of this example, the following procedure shows how to edit an **Event Type** from an existing **.**xsc file using **com.xsc** as the input file.

1. Select the **Event Types** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click the **Create a New Event Type** button.
3. Enter the name of the **Event Type** in the **New Event Type Component** window, then click **OK** (for this sample the Event Type is defined as **Excel**).
4. Double-click the new **Event Type** to edit its properties. The Event Type Properties window opens (see Figure 1).

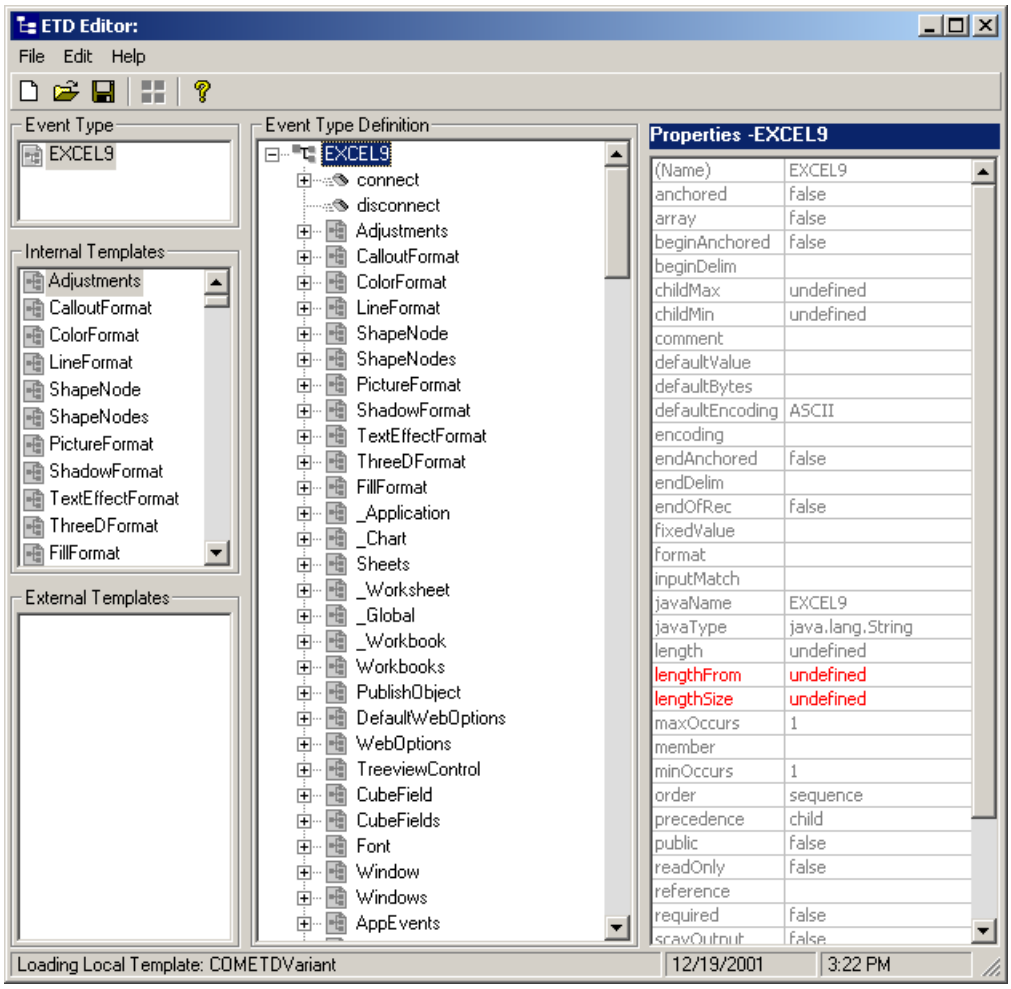
Figure 1 Event Type - Properties



5. Click **Find** under the Event Type Definition field. Find and select **Excel.xsc**.

6 Click the **Edit** button to open the xsc file in the ETD Editor (see Figure 2). This is a large file and may take more than a few minutes to open.

Figure 2 Event Type Definition Editor - Excel



7 On the File menu, click **Compile and Save**. This should be done after any modifications are made to the file.

8 On the **File** menu, click **Promote to Run Time** to promote the file from the **sandbox** to run time. A dialog box opens listing the files that have been successfully promoted.

9 Close the ETD Editor. Click **OK** to close the Event Type Properties Dialog box.

5.5.2. Creating an Event Type Definition Using the COM ETD Wizard

The COM ETD Wizard is used to create a ETD structure specific to the COM, with corresponding nodes and methods. For the purpose of this example, the following procedure shows how to create an **Event Type Definition** (**ETD**) using the **COM ETD Wizard**. This ETD is not used in the sample schema included with the CD.

1 Select the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

2 On the palette, click the **Create a New Event Type** button.

3 Enter the name of the **Event Type** in the **New Event Type Component** window, then click **OK** (for this sample the Event Type is defined as **Excel**).

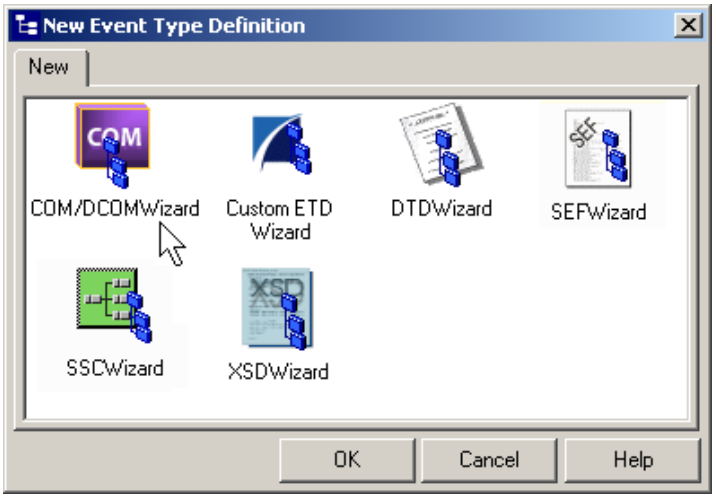
4 Double-click the new **Event Type** to edit its properties.

5 When the **Properties** window opens, click the **New** button. The ETD Editor opens.

6 Select **File**, **New** to open the New Event Type Definition window (see Figure 3).

7 Select the **COM/DCOMWizard**.

Figure 3 New Event Type Definition - COM/DCOMWizard



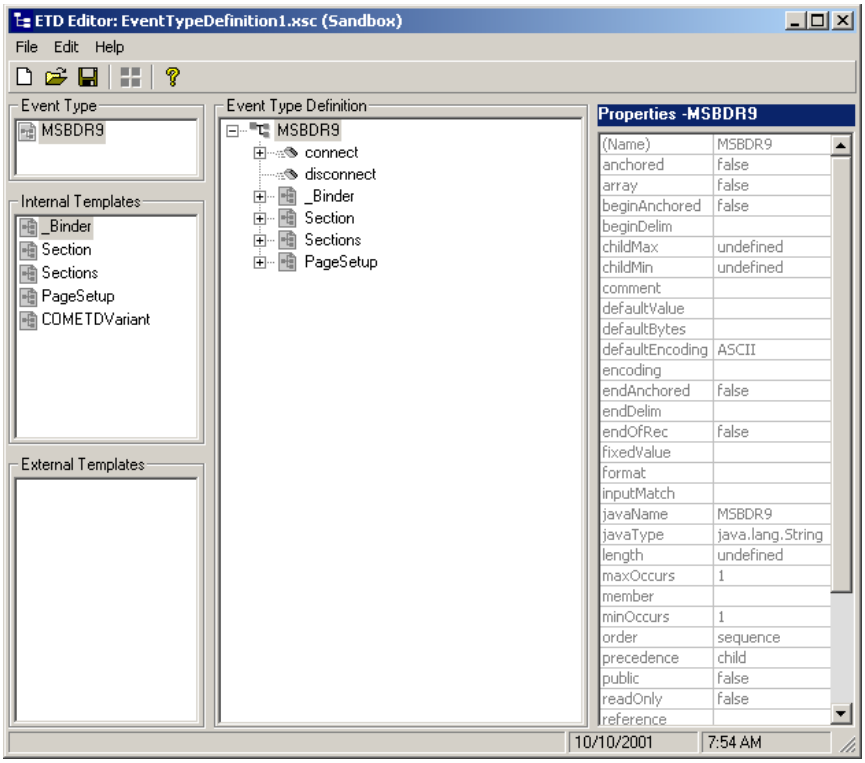
8 Enter a Package Name where the ETD builder can place all the generated Java classes associated with the created ETD (for this sample, **com.stc.ExcelSample**).

9 Enter the **COM Type** library file from which the **.xsc** file is created. TypeLib files are *.tlb, *.olb, *.dll, *.ocx, and *.exe files. For the purpose of this sample the selected file is Excel9.olb at C:\Program Files\Microsoft Office\Office. Click **OK** and **Finish**.

Note: *COM type library files describe the methods exposed from an automation compatible component. Com type library typically have the file extension .tlb or .olb. However, most components embed the type library file in the DLL, OCX or EXE that houses the component.*

10 The **EDT Editor** generates an Event Type structured for the specific COM/DCOM application file (seeFigure 4). The ETD is read-only, allowing for no further configuration from the ETD Editor.

Figure 4 Event Type Definition Editor



11 From the File menu click **Compile and Save**.

12 Once the file has compiled, from the **File** menu, click **Promote to Run Time**. The file is promoted to the run time environment.

13 Close the ETD Editor.

5.6 Creating and Configuring the Component e*Ways

e*Ways connect with external systems to poll or send data. They also transform and route data. Multi-Mode e*Ways are used to run Java Collaborations that utilize e*Way Connections to send and receive Events to and from multiple external systems.

To Create the Trigger e*Way

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Ways.

3 Select the **Control Broker** that manages the new e*Ways.

4 On the palette, click the **Create a New e*Way** button.

5 Enter the name of the new e*Way (in this sample, **trigger**), then click **OK**.

6 Right-click the new e*Way and select **Properties** to edit its properties.

7 The e*Way Properties window opens. Click the **Find** button beneath the **Executable File** field, and select **stcewfile.exe** as the executable file.

8 Under the **Configuration File** field, click the **New** button. The Edit Settings window opens. Select the settings as displayed in Table 4 for this configuration file.

Table 4 Configuration Parameters for the trigger e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	YES
AllowOutgoing	NO
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	C:\Indata (input file folder)
InputFileExtension	*.fin (input file extension)
PollMilliseconds	1000
Remove EOL	YES
MultipleRecordsPerFile	YES
MaxBytesPerLine	4096
BytesPerLineIsFixed	NO
File Records Per eGate Event	1
Performance Testing	Default

9 After selecting the desired parameters, save the **configuration** file (as "**trigger.cfg**").

10 From the **File** menu, click **Promote to Run Time**. This closes the **.**cfg file.

11 In the e*Way - Properties window, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each e*Way you configure.

A Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts after abnormal termination or due to scheduling, and so forth.

B Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

C Use **Security** to view or set privilege assignments.

12 Select **OK** to close the e*Way Properties window.

To Create the COM/DCOM e*Way (Multi-Mode)

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Way.

3 Select the **Control Broker** that manages the new e*Way.

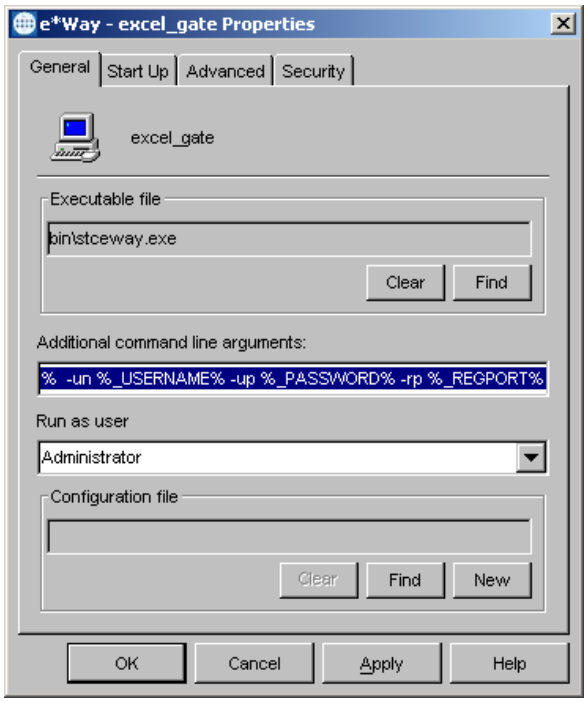
4 On the palette, click the **Create a New e*Way** button.

5 Enter the name of the new e*Way (in this sample, **excel_gate**), then click **OK**.

6 Right-click the new e*Way and select **Properties** to edit its properties.

7 When the e*Way Properties window opens, click the **Find** button beneath the **Executable File** field, and select **stceway.exe** as the executable file (see Figure 5).

Figure 5 Multi-Mode e*Way Properties



8 To edit the JVM Settings, select **New** under Configuration file. For information on setting the parameters for the Multi-Mode e*Way see **Multi-Mode e*Way Configuration** on page 10.

9 Save the **.cfg** file, and promote to run time to close the Edit Settings window.

10 In the e*Way Properties window, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each.

- A Use the **Startup** tab to specify whether the e*Way starts automatically, restarts after abnormal termination or due to scheduling, etc.
- B Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.
- C Use **Security** to view or set privilege assignments.

11 Click **OK** to close e*Way Properties window.

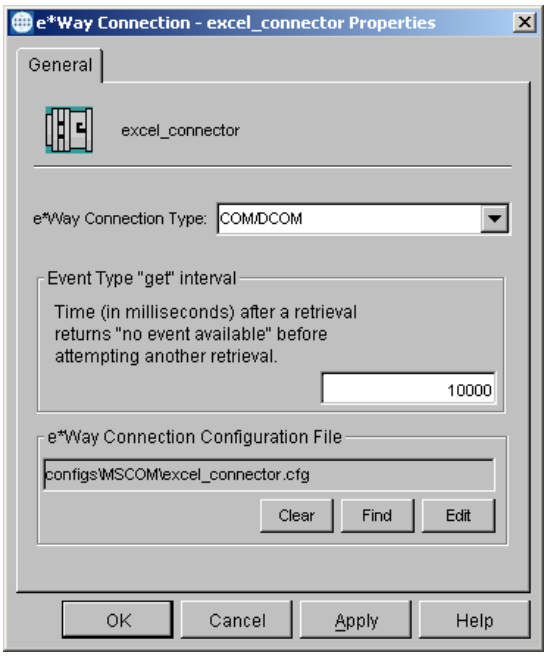
5.7 Creating the e*Way Connection

The e*Way Connection configuration file contains connection information along with the information needed to communicate using COM/DCOM.

5.7.1. To Create and Configure a New e*Way Connection

1. Select the **e*Way Connection** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click the **Create a New e*Way Connection** button.
3. Enter the name of the e*Way Connection (for this sample, **excel_connector**), then click **OK.**
4. Double-click the new e*Way Connection to edit its properties.
5. The e*Way Connection Properties window opens. Select **COM/DCOM** from the **e*Way Connection Type** drop-down menu (see Figure 6).
6. Enter the **Event Type "get" interval** in the dialog box provided. 10000 milliseconds is the configured default. The "get interval is the intervening period at which, when subscribed to, the e*Way connection is polled.

Figure 6 e*Way Connection Properties



7. Under e*Way Connection Configuration File, click the **New** button.
8. The e*Way Connection editor opens. Select the necessary parameters. For information on the COM/DCOM e*Way Connection parameters, see **Configuring e*Way Connections** on page 16.
9. Save the .cfg file. From the **File** menu, click **Promote to Run Time**. The Settings Editor closes.
10. Close the e*Way Connection Properties dialog box.

5.8 Creating Intelligent Queues (IQs)

The next step is to create and associate Intelligent Queues (IQs). IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, and handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

To Create and Modify an Intelligent Queue for the COM/DCOM e*Way

1. Select the Navigator's **Components** tab.
2. Open the host on which you want to create the IQ.
3. Open a **Control Broker**.
4. Select an **IQ Manager**.
5. On the palette, click the **Create a New IQ** button.
6. Enter the name of the new **IQ** (in this sample, **test_iq**), then click **OK.**
7. Double-click the new **IQ** to edit its properties.
8. On the **General** tab, specify the **Service** and the **Event Type Get Interval**.

   The **STC_Standard** IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service DLLs may be created.

   The default **Event Type Get Interval** of 100 Milliseconds is satisfactory for the purposes of this initial implementation.
9. On the **Advanced** tab, make sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.
10. Click **OK** to close the **IQ Properties** window

5.9 Creating the Collaboration Rules

The next step is to create the Collaboration Rules that extract and process selected information from the source Event Type defined above, according to its associated Collaboration Service. The Default Editor can be set to either **Monk** or **Java**.

From the Schema Designer Task Bar, select **Options** and click **Default Editor**. Make sure editor is set to **Java**.

The sample schema calls for the creation of two Collaboration Rules files.

- **trigger_collab** (Pass Through)
- **excel_gate_collab** (Java)

To Create the Pass Through Collaboration Rules

1 Select the Navigator's **Components** tab in the e*Gate Schema Designer.

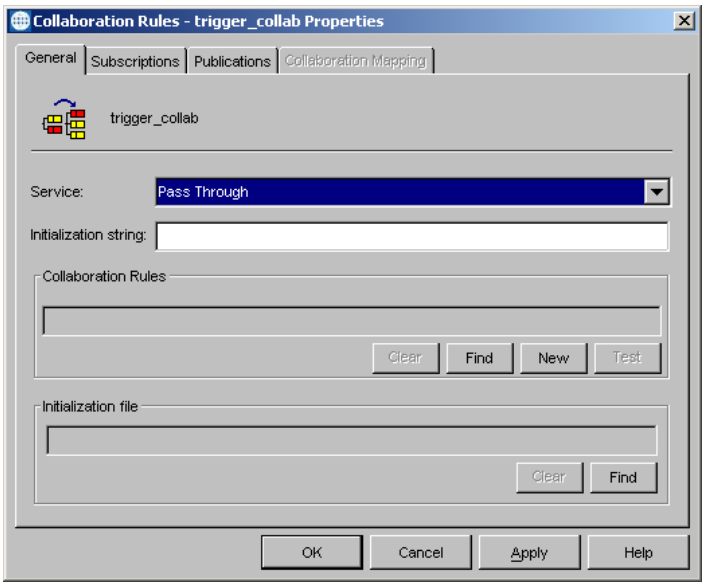
2 In the Navigator, select the **Collaboration Rules** folder.

3 On the palette, click the **Create New Collaboration Rules** button.

4 Enter the name of the new Collaboration Rule Component (for this sample **trigger_collab**), then click **OK**.

5 Double-click the new Collaboration Rules Component. The **Collaboration Rules Properties** dialog box opens (seeFigure 7).

Figure 7 Collaboration Properties - Pass Through

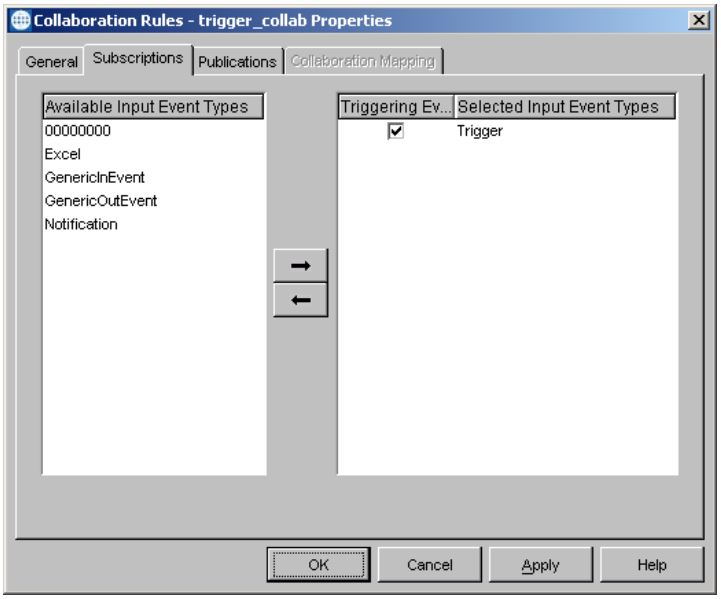


6 The **Service** field defaults to **Pass Through**.

7 Go to the **Publications** tab. Select **Trigger** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. Make sure the **Default** option is be enabled.

8 Go to the **Subscriptions** tab. Select **Trigger** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. Make sure the checkbox under **Triggering Event** is selected (see Figure 8).

Figure 8 Collaboration Properties - Subscriptions



9 Click **OK** to close the **Collaboration Rules Properties** dialog box.

To Create the Java Collaboration Rules

1 Select the Navigator's **Components** tab in the e*Gate Schema Designer.

2 In the **Navigator**, select the **Collaboration Rules** folder.

3 On the palette, click the **Create New Collaboration Rules** button.

4 Enter the name of the new Collaboration Rule, then click **OK** (for this sample, **excel_gate_collab**).

5 Double-click the new Collaboration Rules Component to edit its properties. The **Collaboration Rules Properties** dialog box opens.

6 From the **Service** field drop-down list box, select **Java**. The **Collaboration Mapping** tab is now enabled, and the **Subscriptions** and **Publications** tabs are disabled.

7 In the **Initialization string** field, enter any required initialization string that the Collaboration Service may require. This field can be left blank.

8 Select the **Collaboration Mapping** tab (see Figure 9).

9 Using the **Add Instance** button, create instances to coincide with the Event Types. For this sample, do the following:

A In the **Instance Name** column, enter **trigger** for the instance name.

B Click **Find**, navigate to **Trigger.xsc**, and double-click to select. **Input.xsc** is added to the **ETD** column for this instance.

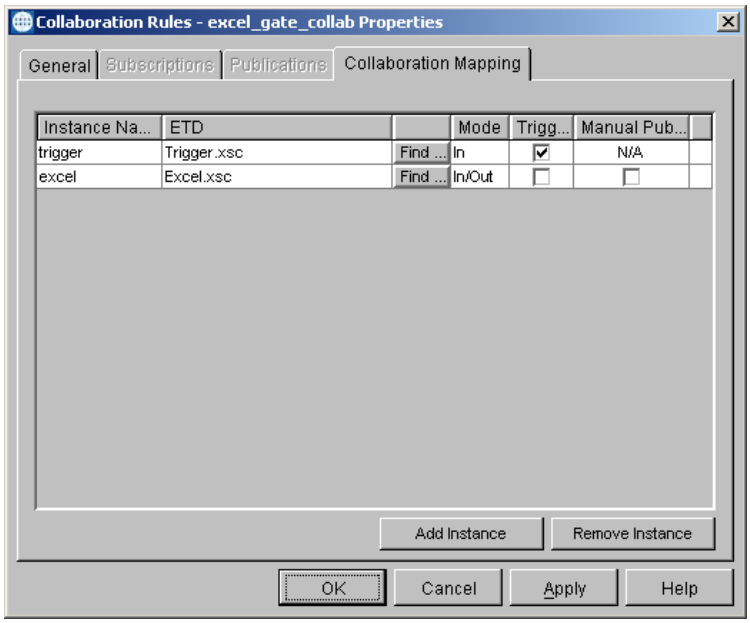
C In the **Mode** column, select **In** from the drop–down menu available.

D In the **Trigger** column, select the box to enable trigger mechanism.

10 Repeat steps 9–13 using the following values:

- Instance Name — **excel**
- ETD — **Excel.xsc**
- Mode — **In/Out**
- Trigger — clear
- Manual Publish – clear

Figure 9 Collaboration Rules - Collaboration Mapping Properties



11 Select the **General** tab, under the Collaboration Rule box, select **New** to open the **Collaboration Rules Editor**. Click **OK** to close the **Collaboration Rules** dialog box or click **New** or **Edit** under the Collaboration Rules field to open the Collaboration Rules Editor.

5.9.1. Using the Collaboration Rules Editor

This section contains an example of how the business logic was defined for the sample schema using the Java Collaboration Rules Editor. The required logic is defined by selecting the desired command and dragging and dropping the nodes (Source or Event) into the corresponding Properties Box, or onto the corresponding node. The resulting code is displayed in the Business Rules pane of the Collaboration Rules Editor.

Each new rule is created by clicking the **rule** button on the Business Rules toolbar. For additional information on using the Java Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*. The **excel_gate_collab** Business Rules are created as follows:

1 To open the Collaboration Rules Editor to the **excel_gate_collab** Collaboration Rules, click **New** or **Edit** under the **Collaboration Rules** field in the Properties dialog box (see above section.) The Collaboration Rules Editor opens. Expand the

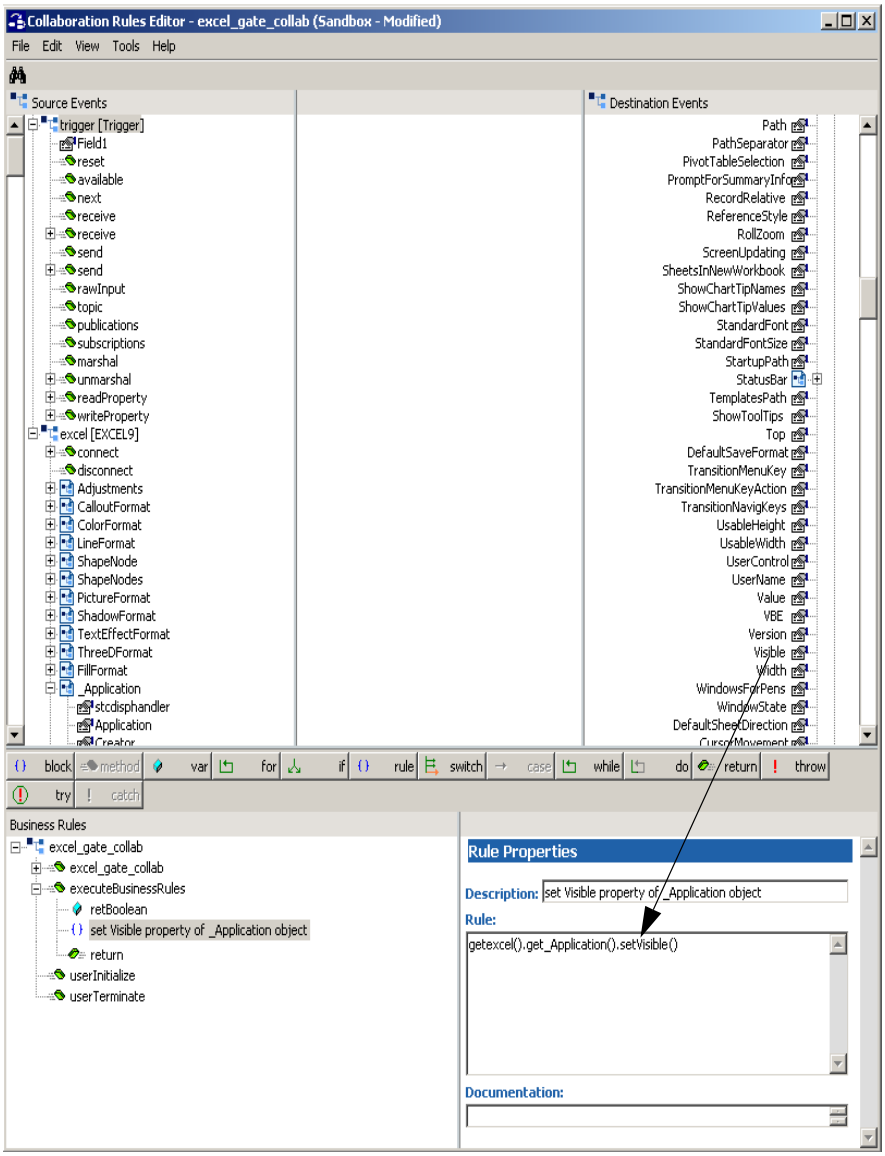
Editor to full size for optimum viewing and expand the **Source** and **Destination** Events as well.

2 Select **retBoolean** in the **Business Rules** pane. All of the user-defined Business Rules are added as part of this method.

3 The **set Visible property of _Application object** rule is created by dragging Visible, under excel, Application, on the Destination Events command node, into the Rule Properties, Rule field (see Figure 10). Place the curser in the last set of parentheses and enter **true** as the value. This creates the following code:

```
getexcel().get_Application().setVisible(true)
```

Figure 10 Collaboration Rules Editor



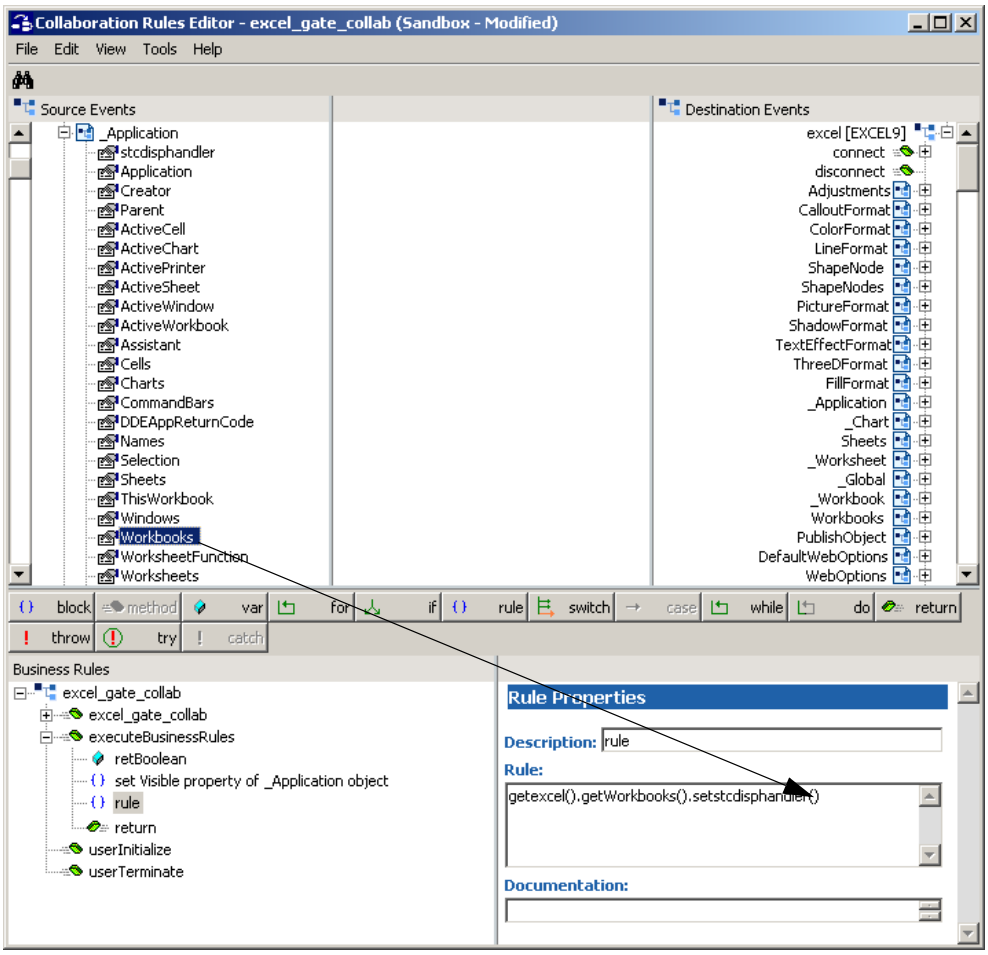
4 Type "**set Visible property of _Application object**" in the Rule Properties, Description field. This replaces "rule" as the expression for this rule in the Business Rules pane.

5 To create the **set 'Workbooks' handle** rule, drag **stcdisphandler** under excel, Workbooks, on the Destination Events command node into the Rule Properties, Rule field. From the Source Events command node, select **Workbooks** under excel, Application. Drag-and-drop it into the last set of parentheses in the Rule Properties, Rule field (see Figure 11), to create the following code:

```
getexcel().getWorkbooks().setstcdisphandler(getexcel().get_Applica
tion().getWorkbooks())
```

stcdisphandler is the node where dispatch handles are stored.

Figure 11 Collaboration Rules Editor



6 To create the **set Filename argument for 'Open'** rule, drag **Filename**, under excel, Workbooks, Open, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and type **"c\\temp\\test.xls"** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().setFilename("c\\temp\\test.xls
")
```

7 To create the **set UpdateLinks for 'Open'** rule, drag **error**, under excel, Workbooks, Open, UpdateLinks, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getUpdateLinks().seterror(-
2147352572)
```

Note: A VT-error of -2147352572 indicates that the parameters of the function are unset.

8 To create the **set ReadOnly for 'Open'** rule, drag **error**, under excel, Workbooks, Open, ReadOnly, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getReadOnly().seterror(-
2147352572)
```

9 To create the **set Format for 'Open'** rule, drag **error**, under excel, Workbooks, Open, Format, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getFormat().seterror(-
2147352572)
```

10 The **set Password for 'Open'** rule is created by dragging **error**, under excel, Workbooks, Open, Password, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getPassword().seterror(-
2147352572)
```

11 The **set WriteResPassword for 'Open'** rule is created by dragging **error**, under excel, Workbooks, Open, WriteResPassword, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getWriteResPassword().seterror
(-2147352572)
```

12 To create the **set IgnoreReadOnlyRecommended for 'Open'** rule, drag **error**, under excel, Workbooks, Open, IgnoreReadOnlyRecommended, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getIgnoreReadOnlyRecommended()
.seterror(-2147352572)
```

13 To create the **set Origin for 'Open'** rule, drag **error**, under excel, Workbooks, Open, Origin, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getOrigin().seterror(-
2147352572)
```

14 The **set Delimiter for 'Open'** rule is created by dragging **error**, under excel, Workbooks, Open, Delimiter, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getDelimiter().seterror(-
2147352572)
```

15 The **set Editable for 'Open'** rule is created by dragging **error**, under excel, Workbooks, Open, Editable, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getEditable().seterror(-
2147352572)
```

16 To create the **set Notify for 'Open'** rule, drag **error**, under excel, Workbooks, Open, Notify, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getNotify().seterror(-
2147352572)
```

17 To create the **set Converter for 'Open'** rule, drag **error**, under excel, Workbooks, Open, Converter, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getConverter().seterror(-
2147352572)
```

18 The **set AddToMru for 'Open'** rule is created by dragging **error**, under excel, Workbooks, Open, AddToMru, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **-2147352572** as the value. This creates the following code:

```
getexcel().getWorkbooks().getOpen().getAddToMru().seterror(-
2147352572)
```

19 The **invoke 'Open' to get _Workbook** rule is created by dragging **invoke**, under excel, Workbooks, Open, on the Source Events command node, into the Rule Properties, Rule field to create the following code:

```
getexcel().getWorkbooks().getOpen().invoke()
```

20 The **set _Workbook handle** rule is created by dragging **stcdisphandler**, under excel, Workbook, on the Source Events command node, into the Rule Properties, Rule field. Drag **retval**, under excel, Workbooks, Open, on the Source Events command node, into the last set of parentheses to create the following code:

```
getexcel().get_Workbook().setstcdisphandler(getexcel().getWorkbook
s().getOpen().getretval())
```

21 The **set Worksheets handle** rule is created by dragging **stcdisphandler**, under excel, Sheets, on the Source Events command node, into the Rule Properties, Rule field. Drag **Worksheets**, under excel, Workbook, on the Source Events command node, into the last set of parentheses to create the following code:

```
getexcel().getSheets().setstcdisphandler(getexcel().get_Workbook()
.getWorksheets())
```

22 To create the **set variant parameter of property 'Item' to 1** rule, drag **int**, under excel, Sheets, propget_Item, Index, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and type 1 to create the following code:

```
getexcel().getSheets().getpropget_Item().getIndex().setint(1)
```

23 The **invoke PROPERTYGET of 'Item'** rule is created by dragging the **invoke** method, under excel, Sheets, propget_Item, on the Destination Events command node, into the Rule Properties, Rule field. This creates the following code:

```
getexcel().getSheets().getpropget_Item().invoke()
```

24 The **set _Worksheet handle** rule is created by dragging the **stcdisphandler**, under excel, Worksheet, on the Destination Events command node, into the Rule Properties, Rule field. Drag **retval**, under excel, Sheets, propget_Item, on the Source Events command node, into the last set of parentheses to create the following code:

```
getexcel().get_Worksheet().setstcdisphandler(getexcel().getSheets(
).getpropget_Item().getretval())
```

25 The **set 'Cell1' of Range** rule is created by dragging **String**, under excel, Worksheet, propget_Range, Cell1, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and enter **A1** as the value to create the following code:

```
getexcel().get_Worksheet().getpropget_Range().getCell1().setstring
("A1")
```

26 The **set 'Cell2' of Range** rule is created by dragging **String**, under excel, Worksheet, propget_Range, Cell2, on the Destination Events command node, into the Rule Properties, Rule field. Place the cursor in the last set of parentheses and type **A1** as the value to create the following code:

```
getexcel().get_Worksheet().getpropget_Range().getCell2().setstring
("A1")
```

27 To create the **invoke 'Range'** rule, drag the **invoke** method, under excel, Worksheet, propget_Range, on the Destination Events command node, into the Rule Properties, Rule field. This creates the following code:

```
getexcel().get_Worksheet().getpropget_Range().invoke()
```

28 To create the **set Range handle** rule, drag the **stcdisphandler** method, under excel, Range, on the Destination Events command node, into the Rule Properties, Rule field. Drag **retval**, under excel, Worksheet, propget_Range, on the Source Events command node, into the last set of parentheses to create the following code:

```
getexcel().getRange().setstcdisphandler(getexcel().get_Worksheet()
.getpropget_Range().getretval())
```

29 The **output value from cell A1** rule is created by typing the following code, into the Rule Properties, Rule field:

```
System.err.println("Cell(A1): " +
((COMETDVariant)getexcel().getRange().getValue()).getValue())
```

30 The **Variant** variable is created by clicking the variant (var) button on the Business Rules toolbar. A variable appears in the Business Rules under the last-selected rule. In the Variable Properties window, enter **Variant** as the description, **variant** as the name, select **com.stc.eways.com.COMETDVariant** as the type, and enter **new COMETDVariant("test value")** as the initial value.

31 To create the **change Cell 'A1'** rule, drag **Value**, under excel, Range, on the Destination Events command node, into the Rule Properties, Rule field. Place the

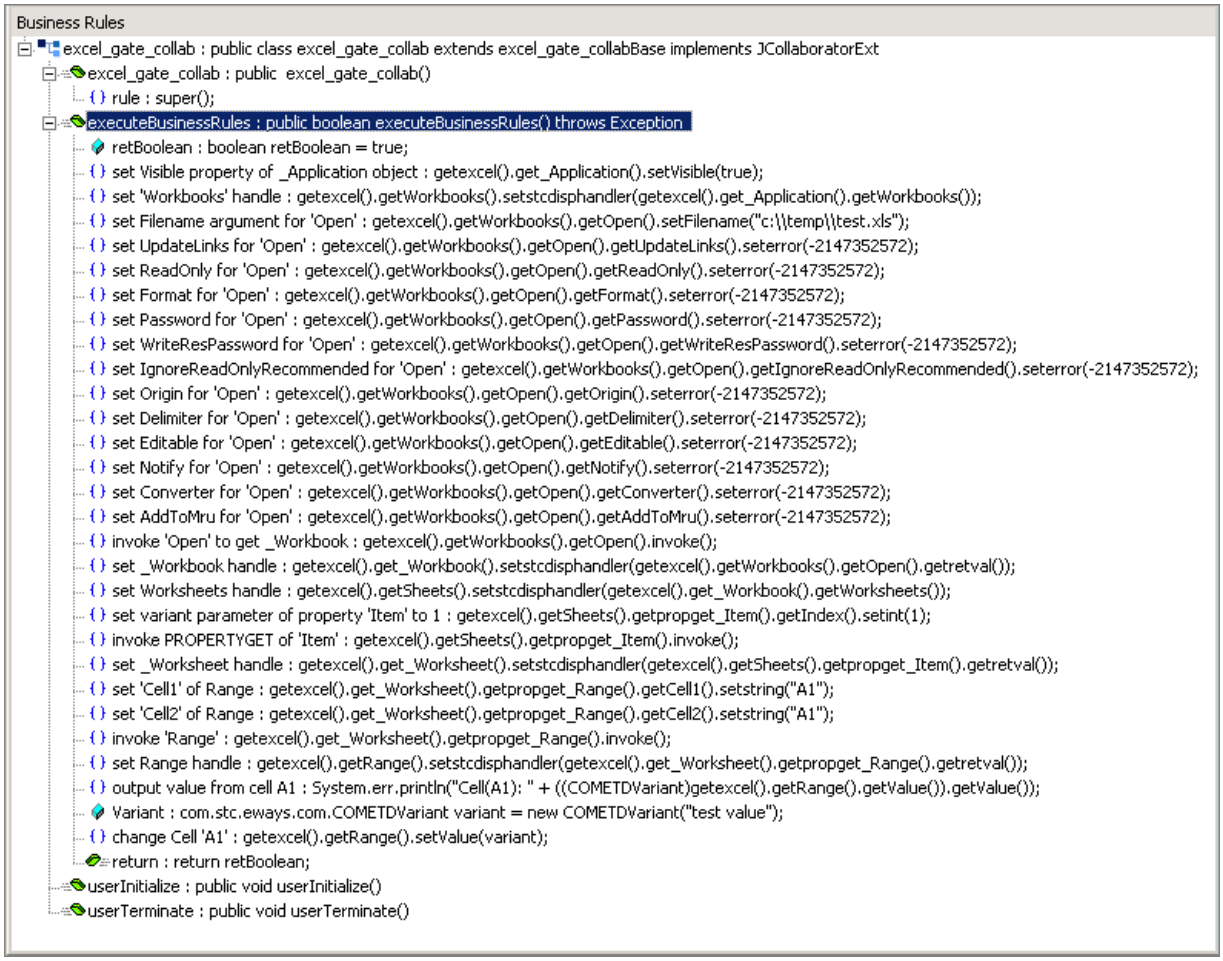
cursor in the last set of parentheses and type **variant** as the value to create the following code:

```
getexcel().getRange().setValue(variant)
```

32 When the business logic has been completed, as displayed in Figure 12, compile and save the Collaboration Rules. From the **File** menu, click **Compile**. The **Save** menu opens, provide a name for the **.**xpr file (for this sample, **excel_gate_collab.xpr**).

Note: *The .xpr file, excel_gate_collab.xpr is included with the sample schema on the Installation CD-ROM, and contains the completed version of this sample.*

Figure 12 Collaboration Rules - Complete



Note: *Business logic can also be created using the e*Ways low-level JNI layer. For more information see* **Java Classes and the Lower JNI Layer** *on page 48 and* **Creating Business Rules Using the e*Way's Low-level JNI Layer** *on page 48.*

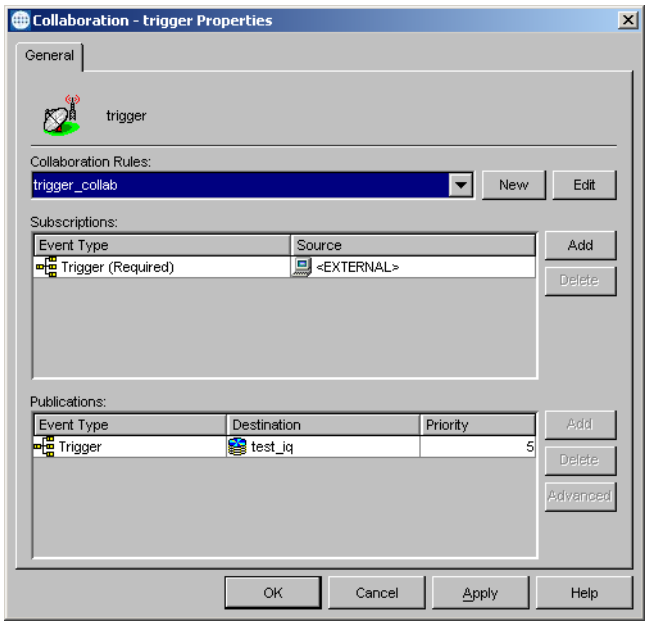
5.10 Creating Collaborations

Collaborations are the components that receive and process Event Types, and then forward the output to other e*Gate components or to an external directory. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a given source), and the Publisher, which distributes the transformed Event to a specified recipient.

To Create the trigger e*Way Collaboration

1. In the e*Gate Schema Designer, select the Navigator's **Components** tab.
2. Open the host on which you want to create the Collaboration, and select the **Control Broker.**
3. Select the **trigger** e*Way, and from the palette, click the **Create a New Collaboration** button.
4. Enter the name of the new Collaboration (for this sample, **trigger**), then click **OK**.
5. Double-click the new Collaboration to edit its properties. The Collaboration - trigger Properties dialog box appears (see Figure 13).

Figure 13 Collaboration Properties - trigger



6. From the Collaboration Rules list, select the Collaboration Rules file that you created previously (for this sample, **trigger_collab**).
7. In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration subscribes.
   A. From the **Event Type** list, select the **Event Type** that you previously defined as **Trigger**.

B Select the **Source** from the **Source** list (for this sample, **<External>**).

8 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration publishes.

A From the **Event Type** list, select the **Event Type** that you previously defined as **Trigger**.

B Select the publication destination from the **Destination** list (for this sample, **test_iq**).

C The Priority field defaults to **5**.

9 Click **OK** to close.

To Create the COM_Multi_Mode Collaboration

1 In the e*Gate Schema Designer, select the **Control Broker.**

2 Select the e*Way being assigned the Collaboration (for this sample, **excel_gate**), and from the palette, click the **Create a New Collaboration** button.

3 Enter the name of the new Collaboration, then click **OK** (for this sample, **excel_gate**).

4 Double-click the new Collaboration to edit its properties.

5 From the Collaboration Rules list, select the Collaboration Rules file that you created previously (for this sample, **excel_gate_collab**).

6 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration subscribes (see Figure 13).

A From the **Instance Name** list, select the Instance Name that you previously defined (for this sample, **trigger**).

B From the **Event Type** list, select the **Event Type** that you previously defined (for this sample, **Trigger**).

C Select the **Source** from the **Source** list (for this sample, **trigger**).

7 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration publishes (see Figure 14).

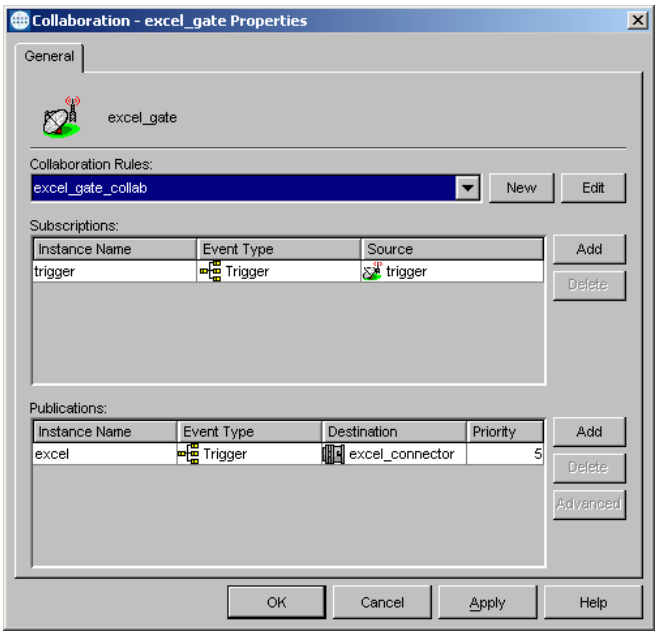
A From the **Instance Name** list, select the **Instance Name** that you previously defined as **excel**.

B From the **Event Types** list, select the **Event Type** that you previously defined (for this sample, **Trigger**).

C Select the publication destination from the **Destination** list (for this sample, **excel_connector**).

D The Priority field defaults to **5**.

Figure 14 Collaboration Properties

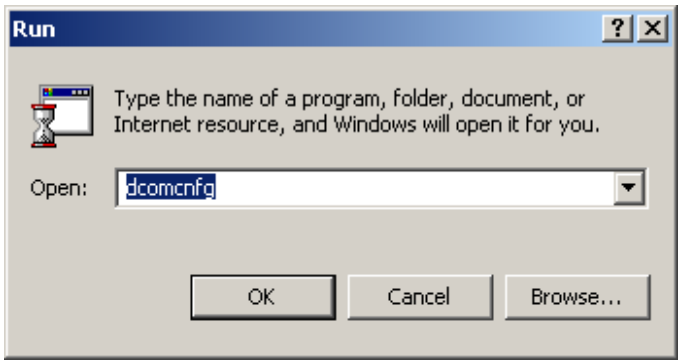


8 Click **OK** to close.

5.11 Configuring the DCOM Server

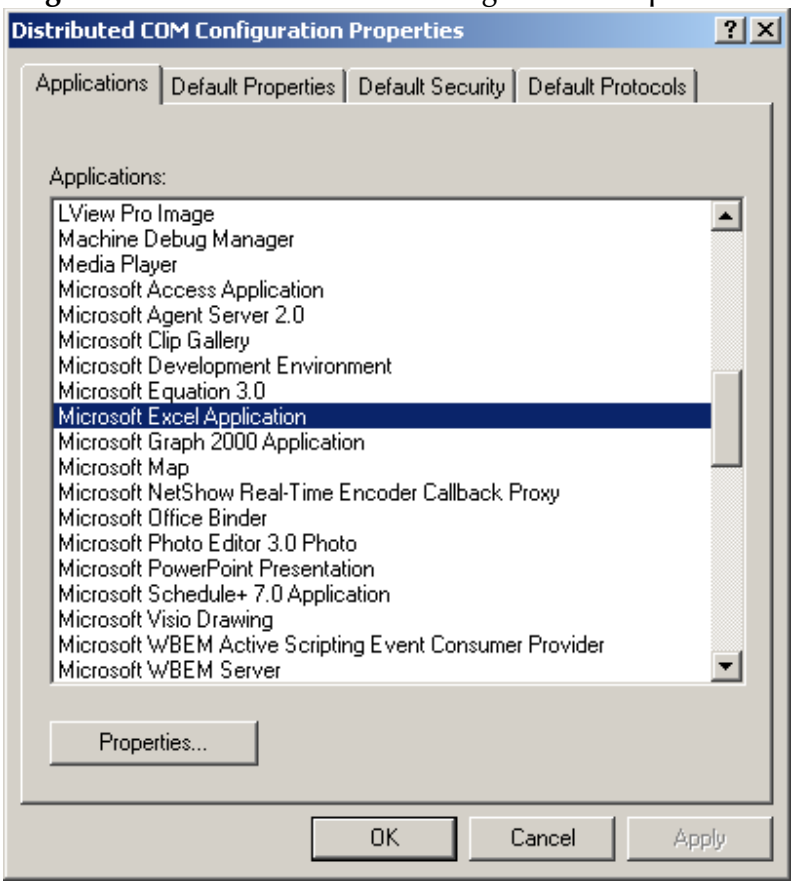
1 To configure the DCOM Server, select the Windows **Run** option. When the Run dialog box opens, type **dcomcnfg** in the **Open** field, and click **OK** (see Figure 15).

Figure 15 Run Dialog



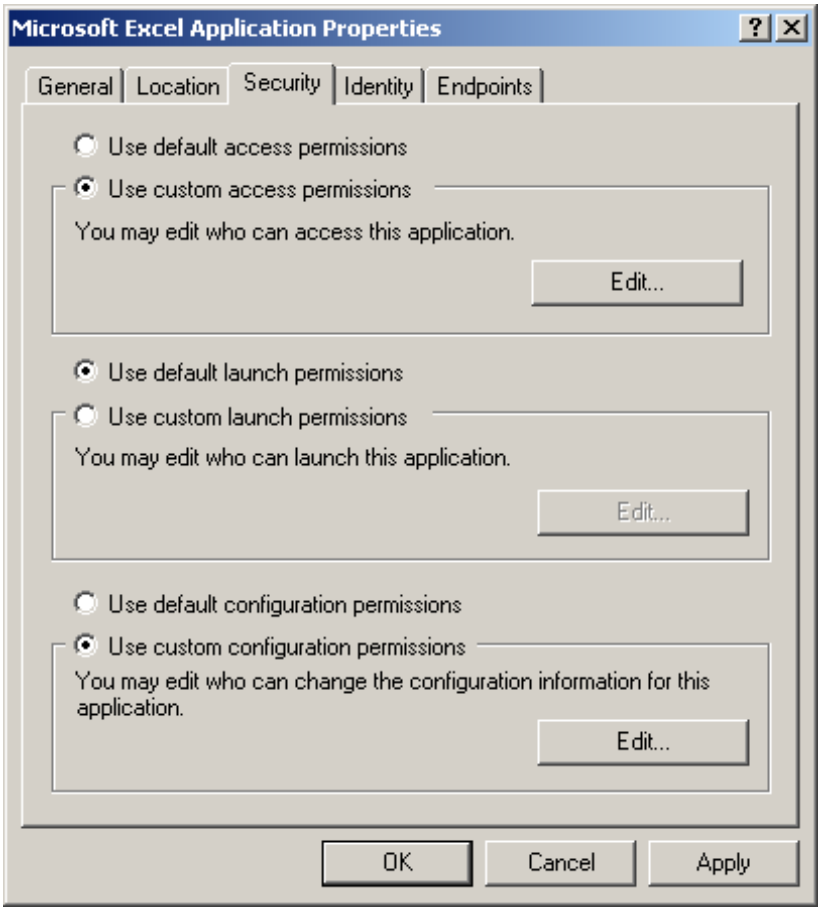
2 The Distributed COM Configuration Properties lists the registered applications. Select the **Microsoft Excel** application (see Figure 16). Click **Properties**.

Figure 16 Distributed COM Configuration Properties



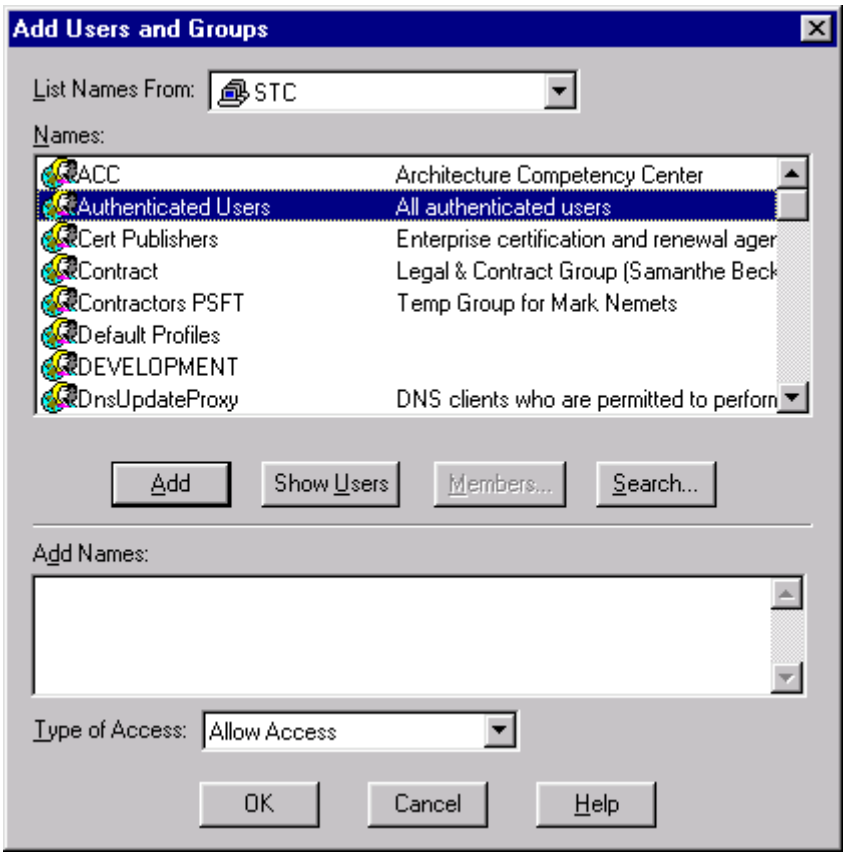
3 The Microsoft Excel Application Properties dialog box appears. Select the **Security** tab (see Figure 17).

Figure 17 Application Properties - Security Tab



4 Select the **Use custom access permission** option button and click **Edit**. The Add Users and Groups dialog box appears (see Figure 18).

Figure 18 Add Users and Groups - Access



5 Select the appropriate users to whom access is granted (for example, **Authenticated Users**), and click **Add**.

6 In the **Type of Access** field, select **Allow Access**, and click **OK**.

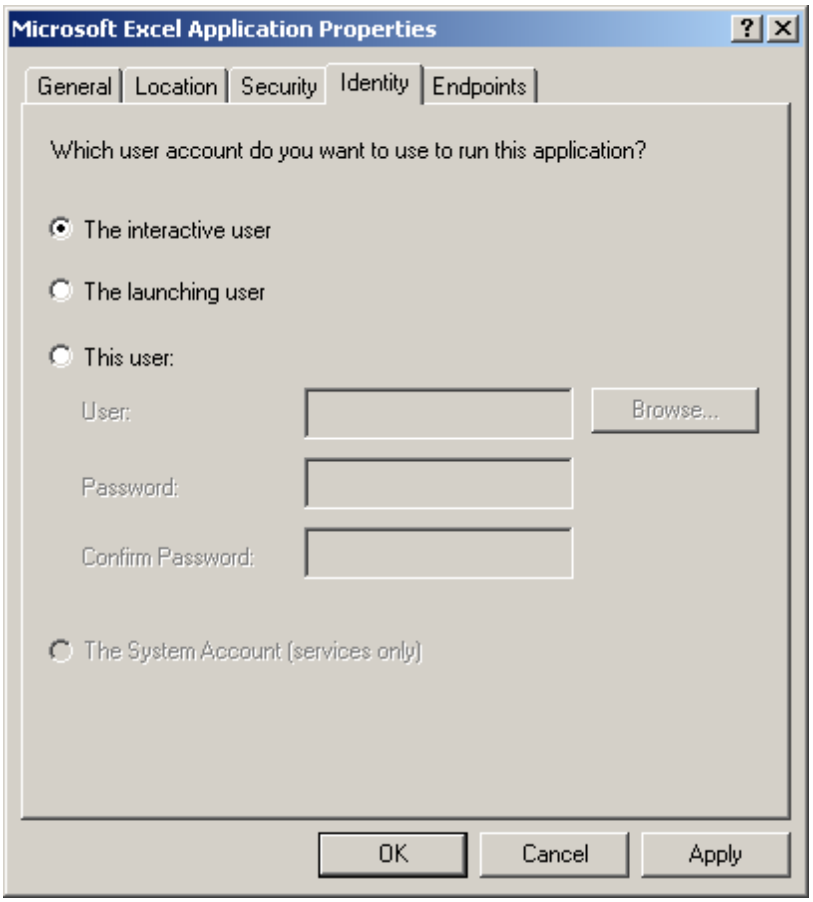
7 In the Application Properties window (see Figure 17), select the **Use custom launch permission** option button and click **Edit**.

8 Select the appropriate users to whom launch permission is granted (for example, **Authenticated Users**), and click **Add**.

9 In the **Type of Access** field, select **Allow Launch**, and click **OK**.

10 In the Application Properties window (see Figure 17), select the **Identity** tab (see Figure 19).

Figure 19 Application Properties - Identity Tab



11 Select **The interactive user** as the user account to use to run the application.

12 Click **OK**. The DCOM server is now configured for the sample schema.

5.12 Sample Schema

A sample of the COM/DCOM e*Way is included on the Installation CD. The sample demonstrates e*Gate's interaction with a COM/DCOM enabled application. The previous sections provided the basics for implementing the COM/DCOM e*Way. This section describes how to implement the COM/DCOM e*Way within a sample schema.

This implementation consists of one file-based e*Way, one Multi-Mode e*Way, two Event Type Definitions, two Collaboration Rules, one Intelligent Queues and two Collaborations, as follows:

- **trigger –** This e*Way receives input from an external source, applies pass through Collaboration Rules, and publish the information to an Intelligent Queue.
- **excel_gate** – This Multi-Mode e*Way applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic.
- **Excel** – This Event Type contains the methods and properties to be used to perform the necessary transformation.
- **Trigger** – This Event Type describes an Event that is input to the extended Java Collaboration Service.
- **trigger_collab** – This Collaboration Rule is associated with the **trigger** e*Way, and is used for receiving the input Event.
- **excel_gate_collab** – The Collaboration Rule is associated with the **excel_gate** Multi-Mode e*Way, and is used to perform the transformation process.
- **test_iq** – This Intelligent Queue is a STC_Standard IQ,.

5.12.1. Completing the Configuration of the ExcelSample Schema

The sample schema is nearly ready to use as it is presently configured. Check the e*Way Connection settings to make sure that they are appropriate for your specific system. Also do the following:

1. In the **Multi-Mode e*Way configuration**, **mscom config**, enter the ProgID for the host (see **ProgID** on page 17). For DCOM the Host name also needs to be entered (see **Hostname** on page 18).
2. Create two folders on the C: drive named DATA and INDATA.

5.12.2. Executing the Schema

To execute the COM/DCOM sample schema

1. Go to the command line prompt, and enter the following:

   ```
   stccb -rh hostname -rs schemaname -un username -up user password
   -ln hostname_cb
   ```

   Substitute ***hostname***, ***schemaname***, ***username*** and ***user password*** as appropriate.
2. Start the Schema Manager. Specify the server that contains the Control Broker you started in Step 1 above.

3 Select the COM/DCOM sample schema.

4 Verify that the Control Broker is connected. To do this, select and right-click the Control Broker in the Schema Manager, and select **Status**. (The message in the Control tab of the console will indicate command *succeeded* and status as *up*.)

5 Select the IQ Manager, ***hostname***_igmgr, then right-click and select **Start**. (This will already be started if **Start automatically** is selected in the IQ Manager properties.)

6 Select each of the e*Ways, right-click select **Start**. (These will already be started if **Start automatically** is selected in the e*Way's properties.)

7 To view the output, copy the output file (specified in the Outbound e*Way configuration file). Save to a convenient location, open.

Note: *Opening the destination file while the schema is running will cause errors.*

5.13 COM/DCOM Run-Time Exceptions

The e*Way Intelligent Adapter for COM/DCOM can be conceptually divided into two layers. The low-level JNI code that wraps the COM IDispatch interface and the higher-level ETD code that is generated by the builder.

At the lower level, all methods accessed via the IDispatch interface return an HRESULT error code. In C Programming Language, this type is a long. In Java, it is an integer. In general, a value of zero indicates success, greater than zero indicates a warning, and less than zero indicates an error.

The low level JNI code has always returned this HRESULT to the builder-generated code. However, the builder-generated ETD code does nothing with the HRESULT. It was possible, then, that the Collaboration could fail to create an instance of the component or a method on the component could fail, and the error may never be caught.

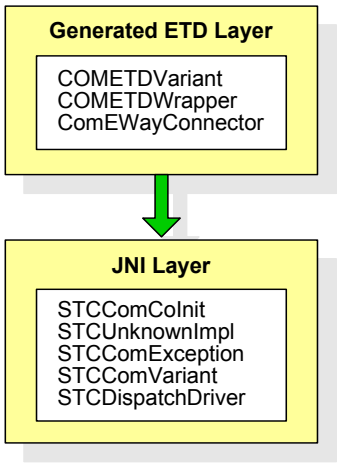
To provide the user with more control over this type of situation, an exception type, STCComException, has been added to the low level JNI code. This exception class is derived from the **java.lang.RuntimeException**. If a method fails on the component (or if the creation of the component fails) an exception of this type is thrown. The exception is passed up through the builder generated ETD code (because the ETD code does not catch it), up to the Collaboration code where the user can catch the exception if desired. Access to the underlying HRESULT is provided. The **getMessage** method has been overridden and provides a brief contextual string indicating the operation that failed.

For more information on the methods of the STCComException Class refer to the COM/DCOM e*Way Javadoc (see **COM/DCOM e*Way Javadoc** on page 52).

5.14 Java Classes and the Lower JNI Layer

The COM/DCOM Java classes can be thought of as a series of layers (see Figure 20). That is, an ETD that is generated using the ETD builder utilizes the **COMETDVariant**, **COMETDWrapper**, and **ComEWayConnector** classes. These classes, in turn, "wrap" the lower level JNI layer classes.

Figure 20 Java Class Layers



When business logic is created using the “drag and drop” feature in the Collaboration Editor, it uses the classes shown in the top layer. It is also possible, however, to drop down to the JNI layer in order to perform other specialized tasks.

5.15 Creating Business Rules Using the e*Way’s Low-level JNI Layer

Typically, Business Rules are created using the Collaboration Editor via features such as “drag and drop.” There may be times, however, when you need to “get a little closer to the metal.” This section discusses the **low-level JNI layer**, and provides an overview of how it is used to define business logic. It is assumed that the reader of this section is well-versed in COM and COM automation, and therefore, does not attempt to address the finer points of COM or any of the semantics of the IDispatch interface.

The two classes typically used are the **STCDispatchDriver** class and the **STCComVariant** class.

The **STCDispatchDriver** class is a shim layer (wrapper) for the **COM IDispatch interface** and supports calling **Invoke** on the underlying automation-compatible object. It also allows for getting and setting properties on the IDispatch object.

The **STCComVariant** class wraps the concept of the COM VARIANT. As in the COM VARIANT structure, the members of the **STCComVariant** class are public, to mimic the union in its COM counterpart. There is also the familiar "**vt**" member variable that

specifies the VARTYPE of the data currently being held, but with a slight deviation in respect to the VT_BYREF and VT_ARRAY specifiers. In the COM counterpart these are bitflags that are or'd in to the vt member. In the **STCComVariant** there are separate member variables to hold these.

Creating an instance of a COM object

To create an instance of a COM object, call create on the **STCDispatchDriver**, passing the ProgID for the object and the server name, if applicable.

The server name parameter can be **null**. It is only needed when the object is a DCOM object residing on a different server. For example:

```
STCDispatchDriver driver = new STCDispatchDriver();
int r = driver.create("Tester2.ComEWayTest", null);
System.out.println("create = " + Integer.toHexString(hr));
```

Here, we are creating an instance of an object identified by theProgID **Tester2.ComEWayTest**. The server is **null** since it is not applicable here. Though this **Tester2.ComEWayTest** does happen to be a DCOM server (that is, an executable), it resides on the same box, and the DCOM creation semantics use the local host as the default, unless it is registered to run on another server (via the dcomcnfg.exe utility).

Note: *The create method (and other methods on the **STCDispatchDriver**) can possibly throw an exception of type **STCComException**. So, technically the driver.create method should be wrapped within a try/catch block, but that is omitted here for clarity.*

Calling a method on the created object

In COM automation, all methods can be called by name. For example, assume that the **Tester2.ComEWayTest** object has a method declared in IDL as follows:

```
[id(1), helpstring("Add two longs and return the result")]
        HRESULT AddTwo([in] long a1, [in] long a2, [out,retval] long*
        result);
```

The procedure for calling this method would be as follows:

```
// Build up the call stack. Both arguments are of type long
// which is a vartype of VT_I4. Note that, as in IDispatch
// Invoke, the arguments are packed in the VARIANT args in
// reverse order.
STCComVariant arg1 = new STCComVariant(1, STCComVariant.VT_I4);
STCComVariant arg2 = new STCComVariant(2, STCComVariant.VT_I4);

STCComVariant[] args = new STCComVariant[2];
args[0] = arg2;
args[1] = arg1;

// This will be used to hold the return value
STCComVariant out = new STCComVariant();

// Invoke the AddTwo function
int r = driver.invoke("AddTwo", args, out);
System.out.println("invoke = " + Integer.toHexString(hr));
System.out.println("out.intValue = " + out.intValue);

// Release the reference count
driver.Release();
```

Single dimension arrays of most of the supported types are also supported. For example, the following assumes the object supported a method declared in IDL:

```
[id(2), helpstring("Add all the shorts in the short array and
return the result")]
    HRESULT AddShortSA([in] SAFEARRAY(short)* psa,
        [out,retval] long* result);
```

The procedure for calling this method would be as follows:

```
// Create an array to add
short[] arr = new short[10];
for (int n = 0; n < 10; n++)
    arr[n] = 10;

// Set up the array argument
STCComVariant arg1 = new STCComVariant();
arg1.pshortValue = arr;
arg1.vt = STCComVariant.VT_I2;
arg1.isArray(true);
arg1.setArrayDim(1);

// pack 'em up - reverse order
STCComVariant[] args = new STCComVariant[1];
args[0] = arg1;

STCComVariant out = new STCComVariant();
int r = driver.invoke("AddShortSA", args, out);
System.out.println("invoke = " + Integer.toHexString(hr));
System.out.println("out.intValue = " + out.intValue);
driver.Release();
```

Calling a method that returns another IDispatch object

These classes can also be used to call a method that returns another IDispatch object. This example may be somewhat contrived, but consider the following case: A COM object is able to return an instance of a component specified by a ProgID, as represented by the following IDL:

```
[id(20), helpstring("Get IDispatch specified by progid")]
    HRESULT GetApp([in] BSTR sProgID,
        [out,retval] IDispatch** ppOut);
```

The procedure for calling this method would be as follows:

```
// This is the ProgID of the application we want it to return
STCComVariant arg1 = new STCComVariant("Vim.Application");

// Pack up the arguments - reverse order
STCComVariant[] args = new STCComVariant[1];
args[0] = arg1;

STCComVariant out = new STCComVariant();
int r = driver.invoke("GetApp", args, out);
System.out.println("invoke = " + Integer.toHexString(hr));

// We're done with this driver now. The returned IDispatch,
// if invoke succeeded, is in out.dispValue
driver.Release();

// In this case we know that the application - gvim in this case -
// supports a method named SendKeys which takes a string
```

```
STCComVariant xarg1 = new STCComVariant("iHello gvim from
java!<ESC>");
STCComVariant[] xargs = new STCComVariant[1];
xargs[0] = xarg1;
STCComVariant xout = new STCComVariant();
r = out.dispValue.invoke("SendKeys", xargs, xout);
out.dispValue.Release();
```

Setting and getting properties

The procedure for setting a property or getting a property is very simple. The general procedure is as follows:

```
// set the property
String msg = new String("Some string property");
STCComVariant vMsg = new STCComVariant(msg);
r = driver.setProp("TheStringProperty", vMsg);

// now read it back
STCComVariant vMsgOut = new STCComVariant();
r = driver.getProp("TheStringProperty", vMsgOut);
```

Chapter 6

e*Way Classes and Methods

The COM/DCOM e*Way contains a number of Java methods that have been exposed to make it easier for the user to set information in the e*Way ETD Editor and to get information from it. These methods are contained in the following classes:

- The **COMETDVariant** Class
- The **COMETDWrapper** Class
- The **ComEWayConnector** Class
- The **STCComCoInit** Class
- The **STCComException** Class
- The **STCComVariant** Class
- The **STCDispatchDriver** Class
- The **STCUnknownImpl** Class

For more information on the Generated ETD Class layer and the low-level JNI layer see **Java Classes and the Lower JNI Layer** on page 48.

6.0.1. COM/DCOM e*Way Javadoc

For a complete list of the Java methods within the classes listed above, refer to the **Javadoc.**

Index

P

R

S

T

U

W