

SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Compaq NonStop CORBA Client User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050405222228.

Contents

Chapter 1

Introduction	7
Compaq NonStop CORBA Client e*Way: Overview	7
Intended Reader	7
General e*Way Operation	7
Event Type Definitions and Collaborations	8
ORB Implementation	8
NonStop Server Implementation	8
e*Way Configuration	9
e*Way Design Overview	9
ETD Configuration and e*Way Operation	9
e*Way Components	10
Supported CORBA Types	10
Supported Operating Systems	11
System Requirements	11
External System Requirements	12

Chapter 2

Installation	13
Installation on Windows Systems	13
Pre-installation	13
e*Way Installation Procedure	13
UNIX	14
Pre-installation	14
e*Way Installation Procedure	14
Files/Directories Created by the Installation	15
External System Requirements	15
Compaq NonStop™ Himalaya Server	15
Interoperable Location Service Daemon	16
Optional External System Information-Orbix 2000™	16

Chapter 3

Multi-Mode e*Way Configuration	17
Multi-Mode e*Way Properties	17
JVM Settings	18
JNI DLL Absolute Pathname	18
CLASSPATH Prepend	18
CLASSPATH Override	19
CLASSPATH Append From Environment Variable	19
Initial Heap Size	19
Maximum Heap Size	19
Maximum Stack Size for Native Threads	20
Maximum Stack Size for JVM Threads	20
Class Garbage Collection	20
Garbage Collection Activity Reporting	20
Asynchronous Garbage Collection	20
Report JVM Info and all Class Loads	21
Disable JIT	21
Remote debugging port number	21
Suspend Option for Debugging	21
General Settings	21
Rollback Wait Interval	21
Standard IQ FIFO	22

Chapter 4

e*Way Connection Configuration	23
Configuring e*Way Connections	23
Configuration Parameters	24
Object Configuration	24
Interoperable Object References (IOR)	24
URI Formats for CORBA Object References	24
Correct Parameters	25
Locate Object Using	27
IOR	27
corbaname host	28
corbaname port	28
corbaname name service	29
corbaname stringified name	29
corbaloc host	30
corbaloc port	30
corbaloc object key	30
file	31
http URL	31
ftp URL	31
Manual URI	32
Advanced Configuration	33
Additional Information	33

ORBClass	35
ORB Singleton Class	35
Additional ORB Properties	35
ORB Arguments	36
Connector Configuration	36
Type	36
Class	36
Property.Tag	36
Configuration Parameters for Orbix 2000™	37

Chapter 5

Implementation	38
Implementation Overview	38
Using the Compaq NonStop CORBA Client ETD Wizard	38
Basic Schema Implementation	44
CorbaHello Sample Schema: Overview	44
Schema Operation	45
Schema Setup	46
Schema Components	46
Sample Schema: General Implementation Steps	48
Before Schema Implementation	48
Importing a Schema	48
Running a Schema	49
CorbaArrays Sample Schema: Overview	49
Schema Operation	49
Schema Setup	50
Schema Components	51
ETD Structure	53
Configuration Nodes	55
Object Nodes	55
Building Collaboration Rules	55
Dynamic Configuration	56
In Case of a CORBA SystemException	57

Chapter 6

e*Way Java Methods	59
e*Way Methods and Classes: Overview	59
CorbaClientConfiguration Class	60
getOrb	60
setOrb	61
getInterfaceURI	61
setInterfaceURI	62
getOrbArguments	62
setOrbArguments	63
getOrbProperties	63
setProperties	63

Contents

CorbaClientETDBase Class	64
getConfiguration	64

Index	65
--------------	-----------

Introduction

This guide explains how to install, set up, and configure the SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for Compaq NonStop CORBA Client. This chapter provides an introduction to the guide and the e*Way.

1.1 Compaq NonStop CORBA Client e*Way: Overview

This guide provides a general introduction to the Compaq NonStop CORBA Client e*Way then explains how to install, configure, and operate it. The guide also explains how to implement the e*Way in a sample e*Gate Integrator environment. The rest of this section provides an overview of the e*Way and how it operates.

Note: CORBA stands for Common Object Request Broker Architecture, a standard for creating, distributing, and managing software objects.

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 high-level knowledge of Windows operations and administration; to be thoroughly familiar with CORBA and types of Object Request Brokers (ORBs); and to be thoroughly familiar with Windows-style operations.

1.3 General e*Way Operation

The Compaq NonStop CORBA Client e*Way allows users to integrate non-CORBA systems with CORBA server applications without the need for custom coding. The e*Way enables e*Gate to make requests to a CORBA service from within a collaboration.

CORBA Services and ORBs

A service is an implementation of one or more interfaces which are managed by an ORB. ORBs generally contain many services consisting of applications, custom components written in-house, and/or services installed with the ORB.

Each service in an ORB is described in an Interface Definition Language (.idl) file. These files are readable text files and usually consist of a module declaration and one or more interfaces.

1.3.1 Event Type Definitions and Collaborations

You can select any .idl file (that you provide) containing the interface that represents your desired service. The e*Gate Schema Designer allows you to create a corresponding e*Gate Event Type Definition (ETD). This feature is a convenient wizard that compiles and generates an .xsc file that makes up the ETD.

This ETD represents a CORBA interface and all the types and other interfaces referenced by the CORBA interface. One or more of these ETDs are then used in a Collaboration to enable it to make requests to the corresponding CORBA service. The ETD represents the methods and attributes on the CORBA service. The e*Way's operation in e*Gate allows it to act as a client to a CORBA server, using the services of a CORBA object from within the Collaboration.

Additionally, the ETD creation (via the wizard window; see ["ETD Wizard" on page 10](#)) uses its own .idl file-to-java compiler.

Note: For more information on ETDs and their use in e*Gate, see the *e*Gate Integrator User's Guide*.

1.3.2 ORB Implementation

The e*Way operates in conjunction with a default client ORB. Therefore, the e*Way's use and correct implementation require that an ORB run-time environment is installed and properly configured on at least one e*Gate Participating Host. This process is automatic when you install the e*Way.

The e*Way supports the use of different CORBA 2.3 compliant client ORBs.

Note: The client ORB of your choice must be installed and configured on the Participating Host. See ["Configuration Parameters for Orbix 2000™" on page 37](#).

1.3.3 NonStop Server Implementation

The Compaq NonStop CORBA Client e*Way interfaces with server objects on the NonStop CORBA system, supports connectivity to the Compaq NonStop ORB server, and supports the NonStop CORBA Name Service.

1.3.4 e*Way Configuration

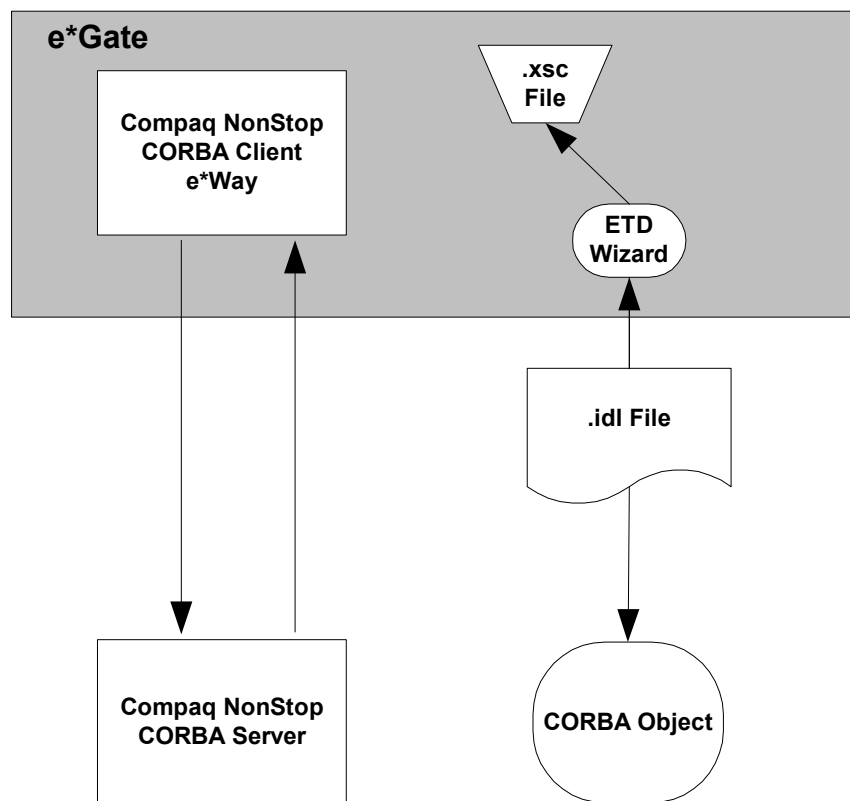
The configuration parameters for the Compaq NonStop CORBA Client e*Way, for convenience, allow you to set necessary parameters of operation. In turn, these e*Way parameters are adopted into the ETD's configuration.

See [Chapter 4](#) for details on how to set the Compaq NonStop CORBA Client e*Way configuration parameters.

1.3.5 e*Way Design Overview

Figure 1 shows a general diagram of the Compaq NonStop CORBA Client e*Way.

Figure 1 Compaq NonStop CORBA Client e*Way



1.3.6 ETD Configuration and e*Way Operation

The Compaq NonStop CORBA Client ETD contains Java methods that allow you to control the CORBA server. The e*Gate Schema Designer via its ETD Editor contains the wizard window that allows you an easy way to create an ETD from your **.idl** file.

ETD Wizard

Using the e*Gate ETD wizard, you select the `.idl` file containing the interface that represents the desired service. The wizard guides you through the steps of creating the ETD and handles all necessary operations required for creating the ETD.

Using the ETD generated by the wizard eliminates the need for manual coding. Using the Collaboration Editor, you can drag and drop ETD nodes to transfer the business logic into the appropriate e*Gate Collaboration Rules.

For complete information on how to use the wizard, see [“Using the Compaq NonStop CORBA Client ETD Wizard” on page 38](#).

1.3.7 e*Way Components

The Compaq NonStop CORBA Client e*Way is made up of the following components:

- The Multi-Mode e*Way executable file, `stceway.exe` (see [Chapter 3](#)).
- The file `stccorbaclient.jar`, the executable component.
- The file `stccorbaclientbuilder.jar` used by the ETD wizard.
- The file `stccommonbuilder.jar`, also used by the ETD wizard.
- Configuration files, which the e*Gate Schema Designer’s e*Way Editor uses to define configuration parameters.
- Additional files necessary for operation, as shown in [Table 1 on page 15](#) (shows a complete list of installed files).

1.3.8 Supported CORBA Types

The Compaq NonStop CORBA Client e*Way supports the following CORBA types:

- `short`
- `long`
- `unsigned short`
- `unsigned long`
- `float`
- `double`
- `char`
- `octet`
- `boolean`
- `string`
- `enum`
- `union`

- **sequence**
- **struct**
- **array**
- **typedef** (modifier)
- **const** (modifier)
- **interface** (construct)
- **module** (construct)
- **exception**

Types Not Supported

The e*Way does *not* support the following types:

- **object by value**
- **long double**
- **fixed**
- **wchar**
- **wstring**
- **any**

Note: The e*Way does not support *.idl* files with a **#pragma** prefix preprocessor directive.

1.4 Supported Operating Systems

The Compaq NonStop CORBA Client e*Way is available on the following operating systems:

- HP NonStop Server G06.19

1.5 System Requirements

To use the Compaq NonStop CORBA Client e*Way, you need to meet the following requirements:

- An eGate Participating Host
- A TCP/IP network connection
- Connectivity to Compaq NonStop™ Himalaya server
- A machine running Windows, to allow you to use the e*Gate Schema Designer and ETD Editor

Installed on the Participating Host

- Java JDK version 1.3.1

Note: *The ETD Builder wizard depends on the correct Java version being on the **classpath** as well as on the **system** path. If there is another Java version (specifically, in the **libin** directory), besides the one recommended for e*Gate, on the system path before the right one, the ETD Builder can fail. In such cases, you get an error message to that effect. To correct this problem, you must ensure the correct version is on the system path, then shut down and restart the e*Gate Schema Designer.*

The e*Way must be configured and administered using the Schema Designer.

Note: *Additional disk space can be required to process and queue the data that this e*Way processes. The amount necessary can vary based on the type and size of the data being processed and any external applications doing the processing.*

1.6 External System Requirements

To use the Compaq NonStop CORBA Client e*Way, you need to meet the following external system requirements:

- An ORB supporting CORBA version 2.3. specifications; the e*Way comes with a default ORB
- Compaq NonStop™ Himalaya server, version 2.3.3
- Interoperable Location Service Daemon (ILSD)

Installation

This chapter explains how to install the Compaq NonStop CORBA Client e*Way.

2.1 Installation on Windows Systems

2.1.1 Pre-installation

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

2.1.2 e*Way Installation Procedure

To install the Compaq NonStop CORBA Client e*Way on Windows systems

- 1 Log in as an Administrator on the workstation where you want to install the e*Way.
- 2 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 3 If the CD-ROM drive's **Auto-run** feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the **setup.exe** file on the CD-ROM drive.
- 4 After the **InstallShield** setup application launches, follow the on-screen instructions to install the 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.

Caution: *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, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

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 UNIX

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under your name with which you wish to own the e*Way files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 e*Way Installation Procedure

To install the Compaq NonStop CORBA Client e*Way on a UNIX system

- 1 Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
- 2 If necessary, mount the CD-ROM drive.
- 3 At the shell prompt, type:
cd /cdrom/setup
- 4 Start the installation script by typing:
setup.sh
- 5 A menu of options appears. Select the **e*Gate Add-on Applications** option. Then, follow any additional on-screen directions.

Be sure to install the e*Way files in the suggested `\client` installation directory. The installation utility detects and suggests the appropriate installation directory.

Caution: 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, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

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

2.3 Files/Directories Created by the Installation

The Compaq NonStop CORBA Client e*Way installation installs the files shown in Table 1. Files are installed within the **eGate** directory on the Participating Host and committed to the “default” schema on the Registry Host.

Table 1 Files Created by Installation

e*Gate Directories	Files
client\classes\	stccorbaclient.jar stccorbaclientbuilder.jar stcewcommonbuilder.jar jacorb.properties
client\etd\	nscorbaclient.ctl nscorbaclientwizard.ctl
server\registry\repository\default\configs\nscorbaclient\	nscorbaclient.def
client\bin	idl.bat
client\ThirdParty\jacorb\	idl.jar jacorb.jar JacORB1_4_beta4-full.tar.gz LICENSE

2.4 External System Requirements

The e*Way Intelligent Adapter for Compaq NonStop CORBA Client e*Way requires the following external systems:

- Compaq NonStop™ Himalaya server, version 2.3.3
- Interoperable Location Service Daemon (ILSD)

2.4.1 Compaq NonStop™ Himalaya Server

This section provides additional information on the role of the Compaq NonStop™ Himalaya server and how it provides the CORBA object interfaces required by the e*Way.

Note: *This e*Way installation requires that the Compaq NonStop™ Himalaya server and NonStop CORBA are already properly installed and configured.*

The CORBA server can publish object references as an Interoperable Object Reference (IOR) in a file, or via a Name Service. The identity and location of the object is encapsulated in the object reference. The client uses a Uniform Resource Identifier (URI), which specifies the object reference, or the mechanism and location to obtain the object reference.

To obtain the object reference from a Name Service you must have the actual TCP address and port for the Interoperable Location Service Daemon (ILSD) associated with the server. This information is then used when using the corbaname: URI bootstrapping mechanism.

For more information on corbaname, see [“Configuration Parameters” on page 24](#).

2.4.2 Interoperable Location Service Daemon

The ILSD is an administrative server naming service that listens at a fixed port and keeps track of the name-spaces in use across the network. It processes requests to locate persistent object references and redirects clients to the appropriate application server.

The ILSD is a component of the Interoperable Name Service. The ILSD services requests for name resolution of names in the URI location format. When the ILSD receives such a request, it returns a forwarding reply containing the object reference of the named object.

Obtaining the TCP address of the ILSD

If using the Name Service, you must have the TCP address for the ILSD to access object references in the Name Service from the e*Way. To obtain the actual TCP address of the ILSD associated with the server use the `cfgmgt` tool.

- Type `cfgmgt`
- Type entity `ilsd1@actual_tcp_Address`

The following is an example TCP address for the ILSD:

```
port_number 8803
host_name    172.26.27.127
```

For more information on using corbaname with the e*Way connection configuration, see [“corbaname name service” on page 29](#).

Note: For more information, see the *NonStop™ CORBA Administration Guide from Compaq®*.

2.5 Optional External System Information-Orbix 2000™

The Orbix 2000™ ORB is not provided with this e*Way. If you are using Orbix 2000™ as the client ORB on the Participating Host, this e*Way requires that the ORB already be properly installed and configured.

Sample configuration parameters for the Orbix 2000™ ORB are provided in [“Configuration Parameters for Orbix 2000™” on page 37](#).

Orbix 2000™ and its documentation for installation, configuration and operation may be obtained directly from IONA Technologies® at www.iona.com.

Multi-Mode e*Way Configuration

This chapter describes how to configure the e*Gate Integrator's Multi-Mode e*Way Intelligent Adapter.

3.1 Multi-Mode e*Way Properties

Set the Multi-Mode e*Way properties using the e*Gate Schema Designer.

To set necessary properties for a Multi-Mode e*Way

- 1 Select the Navigator pane's Components tab in the Main window of the Schema Designer.
- 2 Open the host and Control Broker where you want to create the e*Way.
- 3 On the Palette, click on the icon to create a new e*Way.
- 4 Enter the name of the new e*Way, then click **OK**.
- 5 Select the new component, then click the Properties icon to edit its properties.
The **e*Way Properties** dialog box opens
- 6 Click **Find** beneath the **Executable File** field, and select an executable file (**stceway.exe** is located in the **bin** directory).
- 7 Under the **Configuration File** field, click **New**.
The e*Way Configuration Editor window opens.
- 8 When the **Settings** page opens, set the configuration parameters for this e*Way's configuration file (see "**JVM Settings**" on page 18 and "**General Settings**" on page 21 for details).
- 9 After selecting the desired parameters, click **Save** on the **File** menu to save the configuration (**.cfg**) file.
- 10 Close the **.cfg** file and e*Way Configuration Editor.
- 11 Set the properties for the e*Way in the **e*Way Properties** dialog box.
- 12 Click **OK** to close the dialog box and save the properties.

3.2 JVM Settings

To correctly configure the Compaq NonStop CORBA Client e*Way, you must configure the Java Virtual Machine (JVM) settings. This section explains the configuration parameters in the e*Way Editor window, which control these settings.

Note: *You only need to reconfigure these settings if you are using a client ORB other than the default client ORB. If you are using the default ORB, the default JVM settings are sufficient.*

JNI DLL Absolute Pathname

Description

Specifies the absolute path name to where the JNI **.dll** file is installed by the *Java 2 SDK*, on the Participating Host. This parameter is *mandatory*.

Required Values

A valid path name.

Additional Information

The installed JNI **.dll** file name for Windows systems is **jvm.dll**.

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

```
%MY_JNIDLL%
```

Such variables can be used when multiple Participating Hosts are used.

Caution: *To ensure that the JNI **.dll** file loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory, which contain shared library or **.dll** files.*

CLASSPATH Prepend

Description

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

Required Values

Use the CLASSPATH prepend of the e*Way configuration file to type in the absolute path of the **.jar** file required.

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%
```

Note: Use the `CLASSPATH Prepend` setting to add any required classes if using a different client ORB, such as Orbix 2000™.

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`) will be 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 should 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 will be 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 will be 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 will be 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 will be used.

Required Values

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

Class Garbage Collection

Description

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

Required Values

YES or NO.

Garbage Collection Activity Reporting

Description

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

Required Values

YES or NO.

Asynchronous Garbage Collection

Description

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

Required Values

YES or NO.

Report JVM Info and all Class Loads

Description

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

Required Values

YES or NO.

Disable JIT

Description

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

Required Values

YES or NO.

Remote debugging port number

Description

Specifies whether to allow remote debugging of the JVM.

Required Values

YES or NO.

Suspend Option for Debugging

Description

Indicates whether to suspend Option for Debugging on JVM startup.

Required Values

YES or NO.

3.3 General Settings

This section contains the parameters for rollback wait and IQ messaging priority.

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

3.3.1 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.

3.3.2 Standard IQ FIFO

Description

Specifies whether the highest priority messages from all SeeBeyond Standard IQs are delivered in the first-in-first-out (FIFO) order.

Required Values

Select **Yes** or **No**. **Yes** indicates that the e*Way retrieves messages from all SeeBeyond Standard IQs in the first-in-first-out (FIFO) order. **No** indicates that this feature is disabled; **No** is the default.

e*Way Connection Configuration

This chapter explains how to configure e*Way Connections for the e*Way Intelligent Adapter for Compaq NonStop CORBA Client.

4.1 Configuring e*Way Connections

Set up e*Way Connections using the e*Gate Schema Designer.

To create and configure e*Way Connections

- 1 In the Schema Designer's **Navigation** pane, select the **Component** tab.
- 2 Select the **e*Way Connections** folder.
- 3 On the palette, click on the icon to create a new **e*Way Connection**.
The **New e*Way Connection Component** dialog box appears.
- 4 Enter a name for the **e*Way Connection**, then click **OK**.
An icon for your new e*Way Connection appears in the Navigation pane.
- 5 Double-click on the new **e*Way Connection** icon.
The **e*Way Connection Properties** dialog box appears.
- 6 From the **e*Way Connection Type** drop-down box, select **Compaq NonStop CORBA Client**.
- 7 Enter the appropriate **Event Type "get" interval** in the dialog box provided.
- 8 From the **e*Way Connection Configuration File**, click **New** to open the e*Way Editor.

Note: To use an existing file, click **Find**.

- 9 Use the e*Way Editor to create a new configuration file for this e*Way Connection. Do this operation by selecting the appropriate configuration parameters available.
- 10 When you are finished, close the e*Way Editor and save the new configuration file.

The rest of this chapter explains the Compaq NonStop CORBA Client e*Way Connection configuration parameters as follows:

- “Object Configuration” on page 24
- “Advanced Configuration” on page 33
- “Connector Configuration” on page 36

Note: These parameters can be overridden in the Event Type Definition (ETD). For more information, see “ETD Structure” on page 53.

4.2 Configuration Parameters

This section explains the configuration parameters for the Compaq NonStop CORBA Client e*Way Connection. Each CORBA object represented by an ETD requires its own connection configuration.

4.2.1 Object Configuration

This section allows you to configure the CORBA object parameters.

You must tell the e*Way where to obtain the CORBA object reference by providing a *stringified* Interoperable Object Reference (IOR), or an Interoperable Name Service (INS) Uniform Resource Identifier (URI).

Note: The INS is an extension to the CORBA *CosName Service*. An object reference contains at least three pieces of information: an address, the name of the Portable Object Adapter (POA) that created an object reference, and an Object ID. The URI identifies the object.

Interoperable Object References (IOR)

An IOR is an object reference understood by ORBs, which can interoperate using the OMG-defined protocols General Inter-ORB Protocol (GIOP) and Internet Inter-ORB Protocol (IIOP). A client can obtain an object reference using **orb.string_to_object(objRef)** or as a result of an invocation on another object reference.

The *stringified* IOR is the string representation of the object reference and can be obtained by calling:

orb.objects.string(objRef).

URI Formats for CORBA Object References

The **corbaloc** and **corbaname** are readable formats that enable you to provide a URI to access CORBA objects.

corbaloc

The **corbaloc** format is typically used to resolve the reference using the GIOP **LocateRequest** or **Request** message. Use this format to resolve to a particular CORBA Service without going through a Name Service.

The following **corbaloc** object reference example shows how to get an object reference for **TraderService** from the host **myBank.com** on port 2050:

corbaloc:iiop:1.2@MyBank.com:2050/TraderService

Note: The “1.2” in the sample URI refers to GIOP version 1.2 for the IOR that corresponds to that **corbaloc** URI. GIOP 1.2 is the default value for the default ORB. It is shown in this example in order to demonstrate how you can plug in a different version.

corbaname

The **corbaname** format provides a mechanism for a client to bootstrap directly and is typically used to resolve the *stringified* name from the Name Service. Use this format to resolve to a *stringified* name from a specific naming context.

The following **corbaname** object reference example shows that **myBank.com** is the host and **2050** is the port:

corbaname::myBank.com:2050/NameService#Personal/schedule

In the previous example, the portion of the reference up to the hash mark (**corbaname::myBank.com:2050**) is the URI that returns the root naming context. The example provides the URI to use to:

- Locate the Name Service
- Resolve the name **Personal/schedule** from the Name Service

Correct Parameters

Use the **Object** parameters to specify whether to use a *stringified* IOR or INS URI for locating and obtaining an object reference for the CORBA interface. Observe the following rules:

- If you use an IOR, you must enter the IOR in full (IOR parameter), or specify a file containing it as a URL (http/ftp parameters) or via a path (file parameter).
- If you use an INS URI, you must specify whether it is in the **corbaname** or **corbaloc** format and enter the information required by the format.

You must know how to correctly enter the parameter precisely according to the type of object reference format (bootstrapping mechanism) you use.

Note: Orbix 2000™ does not support corbaloc, ftp, file, and http bootstrapping mechanisms.

Table 2 lists the formats/mechanisms and also explains how to enter the parameters for each one.

Table 2 Supported Bootstrapping Mechanisms

Bootstrapping Mechanism	Description	Correct Entry
IOR	An unreadable (by a person) unique identifier for an object. Each IOR contains all the information necessary to find and call the desired object.	You must specify the correct IOR identifier. The IOR can change for each run of the server, depending on your system configuration and implementation.
corbaloc	Contacts a published object reference on a server, for example, a Trading Service.	Enter parameters as required by the e*Way Editor and explained later in this section.
corbaname	Makes a query against a Name Service. The corbaname: (an extension of corbaloc) contains the stringified name that identifies the right binding in a naming context.	Enter parameters as required by the e*Way Editor and explained later in this section.
File, HTTP, and FTP	These formats are supported by this e*Way, allowing you to save an IOR or URI in a file, then specify that file via: <ul style="list-style-type: none"> ▪ File and path name, if it is in a reachable system. ▪ HTTP URL if it is published on a Web server. ▪ FTP URL if it is published in an FTP server. 	Enter the appropriate character string, as follows: <ul style="list-style-type: none"> ▪ If using the file system, enter the complete path and file name. ▪ If http or ftp, enter the complete URL.
Manual URI	Allows you to specify the URI for the IOR in whatever format the underlying client ORB uses.	Enter the URI in the specific format the current client ORB understands.

The rest of this section explains the e*Way Connection's **Object** configuration parameters.

Locate Object Using

Description

Allows you to select the parameters for locating the object to use with the ETD generated for a CORBA interface.

Required Values

Select one of the following parameters:

- **IOR:** Specifies the full *stringified* IOR; be sure to use the IOR for the exact object you want to call. Be aware that, with some ORBs, the IOR changes each time the server is started up.
- **file, http URL, or ftp URL:** Specify the IOR saved in a file; enter under the appropriate format.
- **corbaname:** Queries a Name Service for the IOR of a registered object. The Name Service has to support INS specifications.
- **corbaloc:** Use if your object is published as a remote service.
- **Manual URI:** Specifies the URI for the IOR in any format that the underlying client ORB understands. You can use any value that is understood by the **string_to_object** implementation of the underlying ORB.

Note: *This parameter is the recommended setting when you are configuring a different client ORB than the default ORB listed under “[Advanced Configuration](#)” on [page 33](#).*

To specify the URI at run-time instead of the e*Way Connection configuration, use the **Manual URI** setting, which allows you to enter an empty value. This parameter is the default.

Note: *The additional parameters under **Object**, which you need to configure, depend upon your choice from the previous list, as noted under each of the parameter sections.*

IOR

Description

Specifies a IOR stringified object reference used to locate the object for the CORBA interface, for example:

```
IOR:0000000000000001B49444C3A64656D6F2F68656C6C6F2F476F6F644461793A312
E300000000000010000000000000060000102000000000D31302E312E3139302E3234360
00004EE0000001C5374616E64617264496D706C4E616D652F001041403C0A1604013
D0100000001000000010000001C00000000001000100000001050100010001010900000
00105010001
```

Note: *Only use this parameter if you specified IOR under “[Locate Object Using](#)” on [page 27](#).*

Required Values

A valid IOR stringified object reference.

corbaname host

Description

Specifies the host to use for **corbaname**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaname::corbaname_host:corbaname_port/  
corbaname_naming_service#corbaname_stringified_name
```

Note: See the appropriate vendor documentation of the server ORB regarding the port and object key for the INS, as well as for the procedures to run/enable the INS.

To specify a more complex **corbaname** URI use the **Manual URI** parameter. The host selected has to be for the INS.

Note: Only use this parameter if you specified **corbaname** under **“Locate Object Using” on page 27**.

Required Values

A valid host name.

corbaname port

Description

Specifies the port to use for **corbaname**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaname::corbaname_host:corbaname_port/  
corbaname_naming_service#corbaname_stringified_name
```

Note: See the appropriate vendor documentation of the server ORB regarding the port and object key for the INS, as well as for the procedures to run/enable the INS.

To specify a more complex **corbaname** URI use the **Manual URI** parameter. The port key selected has to be for the INS.

Note: Only use this parameter if you specified **corbaname** under **“Locate Object Using” on page 27**.

Required Values

A valid host port number.

corbaname name service

Description

Specifies the name of the object key for the Name Service to use for **corbaname**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaname::corbaname_host:corbaname_port/  
corbaname_naming_service#corbaname_stringified_name
```

Note: See the appropriate vendor documentation of the server ORB regarding the port and object key for the INS, as well as for the procedures to run/enable the INS.

To specify a more complex **corbaname** URI use the **Manual URI** parameter. The object key selected has to be for the INS.

Note: Only use this parameter if you specified **corbaname** under **“Locate Object Using”** on page 27.

Required Values

The name of a valid object key for a CORBA Name Service, for example, **NameService**.

corbaname stringified name

Description

Specifies the stringified name of the object to use for **corbaname**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaname::corbaname_host:corbaname_port/  
corbaname_naming_service#corbaname_stringified_name
```

The stringified name specified has to be the name the object was bound to in the INS, for example:

/myApp/MyObject1

Note: See the appropriate vendor documentation of the server ORB regarding the port and object key for the INS, as well as for the procedures to run/enable the INS. Consult the vendor documentation of the server ORB, as well as the server documentation/implementation regarding the stringified name the server object was bound to.

To specify a more complex **corbaname** URI use the **Manual URI** parameter.

Note: Only use this parameter if you specified **corbaname** under **“Locate Object Using”** on page 27.

Required Values

A valid stringified name of the object to use for **corbaname**.

corbaloc host

Description

Specifies the host to use for **corbaloc**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaloc::corbaloc_host:corbaloc_port/corbaloc_object_key
```

To specify a more complex **corbaloc** URI use the **Manual URI** parameter.

Note: Only use this parameter if you specified **corbaloc** under “**Locate Object Using**” on page 27.

Required Values

A valid host name.

corbaloc port

Description

Specifies the port to use for **corbaloc**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaloc::corbaloc_host:corbaloc_port/corbaloc_object_key
```

To specify a more complex **corbaloc** URI use the **Manual URI** parameter.

Note: Only use this parameter if you specified **corbaloc** under “**Locate Object Using**” on page 27.

Required Values

A valid host port number.

corbaloc object key

Description

Specifies the object key to use for **corbaloc**. This parameter is used in combination with other fields in this configuration to construct:

```
corbaloc::corbaloc_host:corbaloc_port/corbaloc_object_key
```

To specify a more complex **corbaloc** URI use the **Manual URI** parameter.

Note: Only use this parameter if you specified **corbaloc** under “**Locate Object Using**” on page 27.

Required Values

The name of a valid **corbaloc** object key.

file

Description

Specifies the fully qualified file containing the stringified IOR. The file must be reachable via the file system of an e*Gate Participating Host, using the specified file name, for example:

C:/tmp/myiorfile.txt

You must ensure that the file is reachable via the specified file name from all Participating Hosts this e*Way Connection configuration is used from. It is recommended that you use one of the other bootstrapping options, for example, HTTP or FTP instead, if possible.

Note: Only use this parameter if you specified **file** under **“Locate Object Using”** on [page 27](#).

Required Values

A valid path and file name reachable by a Participating Host.

Note: This option is not supported by all client ORBs.

http URL

Description

Specifies the HTTP URL where the e*Way can find the file containing the stringified IOR, for example:

http://myhost.com:8080/myiorfile.txt

The Web server hosting the file must be reachable via HTTP from an e*Gate Participating Host.

Note: Only use this parameter if you specified **http URL** under **“Locate Object Using”** on [page 27](#).

Required Values

A valid HTTP URL reachable by a Participating Host.

Note: This option is not supported by all client ORBs.

ftp URL

Description

Specifies the FTP URL where the e*Way can find the file containing the stringified IOR, for example:

ftp://myhost.com:8080/myiorfile.txt

The FTP server hosting the file must be reachable via FTP from an e*Gate Participating Host. If the FTP server does not allow for an anonymous log-in, you must enter a user name and password, according to the following format:

```
ftp://<username>:<password>@<host>:<port>/<file>  
ftp://user:pass@myhost.com:9090/tmp/myiorfile.txt
```

Note: Only use this parameter if you specified **ftp URL** under “Locate Object Using” on page 27.

Required Values

A valid FTP URL reachable by a Participating Host.

Note: This option is not supported by all client ORBs.

Manual URI

Description

Specifies a full, manually set INS URI used to obtain an IOR for the CORBA interface. This parameter allows you to configure URI formats not supported by the preceding options, such as specifying **corbaloc** and **corbaname** with multiple hosts. The exact format and features supported are dependent on the underlying client ORB used.

The string entered is passed without reformatting directly to the underlying ORB. For example, to specify further **corbaname** options, such as protocol versions and multiple hosts, use the format shown in the following example:

```
corbaname:iiop:1.2@host1.com,iiop:host2.com/NameService/MyObject
```

Note: For the exact format supported, as well as further details see the appropriate documentation of the client ORB vendor.

This field can be left empty if you want to specify the URI used at run time, before accessing the CORBA interface functionality.

Note: Only use this parameter if you specified **Manual URI** under “Locate Object Using” on page 27.

Required Values

A valid INS-compliant URI.

Additional Information

The URI must be reevaluated after a CORBA exception is thrown. Use the following work-around to force this action (**MyServer** is the interface):

```
try {
    // My collab code
} catch (org.omg.CORBA.SystemException ex) {
    // The following will force the interface to be looked up/
    resolved again before the next time it is used

    getanarr().setMyServer((com.stc.corbaclient.arrpkg.stubs.arrays.MyServer)null);
    throw ex;
}
```

This work-around has the effect that the next time the Collaboration is run and this interface is accessed, the object reference is retrieved/resolved from the start, according to the set URI.

The null has to be typecast in this work-around because there are two **set** methods to choose from. If you compile without the typecast, the compiler gives you an error message. For the previous example, such an error could be:

reference to setMyServer is ambiguous ...

In addition, this error string includes the two argument types you must choose from. Either type can be used. In the above work-around, the type of the underlying stub for the interface is used.

4.2.2 Advanced Configuration

This section allows you to reconfigure the e*Way's client ORB parameters. Setting these parameters is optional, and you only need to change or enter parameters in this section if you are using a client ORB other than the e*Way's default client ORB.

Note: *The e*Way communicates with other ORBs via IIOP. You only need to change or enter parameters in this section if you want to change the e*Way's default client ORB (not the server ORB). If you change the client ORB, you must copy any necessary ORB implementation .jar files to the stceway.exe file's classpath prepend.*

Additional Information

You only need to edit or enter the **Advanced** configuration parameters for the e*Way Connection if you are using a client ORB different from the Compaq NonStop CORBA Client e*Way's own default ORB. Because many types of client ORBS exist in the CORBA world, no list here can explain how to configure them all. Refer to the documentation for your ORB for complete information.

Note: *The JacORB client ORB product is used in this e*Way. JacORB and the use of JacORB are covered by the GNU Library General Public License, Version 2. For more information about JacORB, see their documentation.*

Setting ORB Properties

You can use the following ways to set ORB properties:

- In the e*Way Connection's configuration
- By manually editing the following file installed with the e*Way:

egate/client/classes/jacorb.properties

The format to use for entering all e*Way Connection configuration parameters is:

propertyname=value

Each value must be separated by a space. If the value itself contains a space, you must surround the value by quotation marks.

Default ORB Connection Properties

There are many ORB properties that can be set for the default ORB. These settings range from, for example, debug logging to retry intervals. The following list provides some examples of the properties that can be set, to give you a general idea:

- The number of retries if the connection cannot be directly established:
jacorb.retries=integer (default is 5)
- How many milliseconds the ORB waits between retries:
jacorb.retry_interval=integer (default is 500)
- The client-side time-out, set to non-zero to stop blocking, after the specified number of milliseconds:
jacorb.connection.client_timeout=integer (no default)

For more information, see the **jacorb.properties** file installed in **egate/client/classes**.

Debugging Information

Here is some basic debugging information to use for the default ORB:

- Do a hex dump of outgoing messages?
jacorb.debug.dump_outgoing_messages=off
- Do a hex dump of incoming messages?
jacorb.debug.dump_incoming_messages=off
- The output destination for the diagnostic log:
jacorb.logfile=file name

The rest of this section explains the e*Way Connection's **Advanced** configuration parameters.

ORBClass

Description

Specifies the ORB class of the client ORB implementation to use. When using a custom client ORB instead of the default ORB, be sure the ORB implementation classes are prefixed to the **classpath** of the e*Way.

This parameter is passed as the **org.omg.CORBA.ORBClass** property in the **props** argument in the call to:

```
orb.omg.CORBA.ORB.init(java.lang.String[] args, java.util.Properties props)
```

Required Values

A valid ORB class name.

ORB Singleton Class

Description

Specifies the ORB singleton class of the client ORB implementation to use. When using a custom client ORB instead of the default ORB, be sure the ORB implementation classes are prefixed to the **classpath** of the e*Way.

This parameter is passed as the **org.omg.CORBA.ORBSingletonClass** property in the **props** argument in the call to:

```
orb.omg.CORBA.ORB.init(java.lang.String[] args, java.util.Properties props)
```

Required Values

A valid Java class name.

Additional ORB Properties

Description

Allows you to specify additional key/value pair properties, separated by a space, using the following format:

```
key1=value1 key2=value2 ...
```

These properties are used for the **props** argument (in addition to the **ORBClass** and **ORBSingletonClass** parameters specified for the e*Way configuration) in the call to:

```
orb.omg.CORBA.ORB.init(java.lang.String[] args, java.util.Properties props)
```

Required Values

One or more Java key/value pair properties, for example:

```
com.stc.aproperty.title="Example 1" com.stc.aproperty.amount=2  
com.stc.aproperty.id=MyID
```

Note: If the value itself contains a space, you must surround the value by quotation marks.

ORB Arguments

Description

Use the following format:

value1 value2 ...

These values are used for the **args** argument in the call to **orb.omg.CORBA.ORB.init(java.lang.String[] args, java.util.Properties props)**.

Required Values

One or more Java values, for example:

ORBdomain_name test anotherArgument

Note: If the value itself contains a space, you must surround the value by quotation marks.

4.2.3 Connector Configuration

This section allows you to configure the Collaboration engine to identify the e*Way Connection.

Type

Description

Specifies the type of e*Way Connection.

Required Values

Corba Client ETD. The value defaults to **Corba Client ETD**.

Class

Description

Specifies the class name of the CORBA Client ETD connector object.

Required Values

A valid package name.

The default is **com.stc.eways.corbaclient.CorbaClientETDConnector**.

Property.Tag

Description

Identifies the data source. This parameter is required by the current **EBobConnectorFactory**.

Required Values

A valid data source package name.

4.3 Configuration Parameters for Orbix 2000™

This section provides examples of the configuration parameters for the Compaq NonStop CORBA Client e*Way Connection if you are using Orbix 2000™ as a client ORB. The path names, directories, and domains will vary. These are only meant as examples to illustrate how the strings correspond to the configuration parameters.

ORBClass

```
com.iona.corba.art.artimpl.ORBImpl
```

ORBSingletonClass

```
com.iona.corba.art.artimpl.ORBSingleton
```

ORBArguments

```
-ORBproduct_dir "C:\Program Files\IONA" -ORBconfig_domains_dir  
"C:\Program Files\IONA\etc\domains" -ORBdomain_name "dev" -  
ORBlicense_file "C:\Program Files\IONA\etc\licenses.txt"
```

Note: You must include the *orbix2000.jar* file in the classpath prepend for the stceway configuration, for example:

```
"C:\Program Files\IONA\orbix_art\2.0\lib\orbix2000.jar"
```

For more information, see [“CLASSPATH Prepend” on page 18](#).

Implementation

This chapter explains a sample schema to help you understand how to implement the e*Way Intelligent Adapter for Compaq NonStop CORBA Client in a production environment, including procedures for using the CORBA Client ETD wizard to create an ETD from an **.idl** file.

5.1 Implementation Overview

This section explains how to implement the Compaq NonStop CORBA Client e*Way using an e*Gate Integrator schema sample included on your installation CD-ROM. Find this sample on the CD-ROM at the following path location:

`\samples\ewnsorbaclient`

For more information, see [“Importing a Schema” on page 48](#).

This sample allows you to observe and/or create end-to-end data-exchange scenarios involving e*Gate, the e*Way, and CORBA interfaces. This chapter explains how to implement the CorbaHello schema that uses the Compaq NonStop CORBA Client e*Way.

The chapter also explains how to use the CORBA Client wizard, as well as the CORBA Event Type Definition (ETD) structure and additional information you need to know to implement this e*Way.

5.2 Using the Compaq NonStop CORBA Client ETD Wizard

The Compaq NonStop CORBA Client ETD wizard allows you to create an ETD (.xsc file) from CORBA object interfaces (Interface Definition Language **.idl** files). The rest of this section explains how to use the wizard.

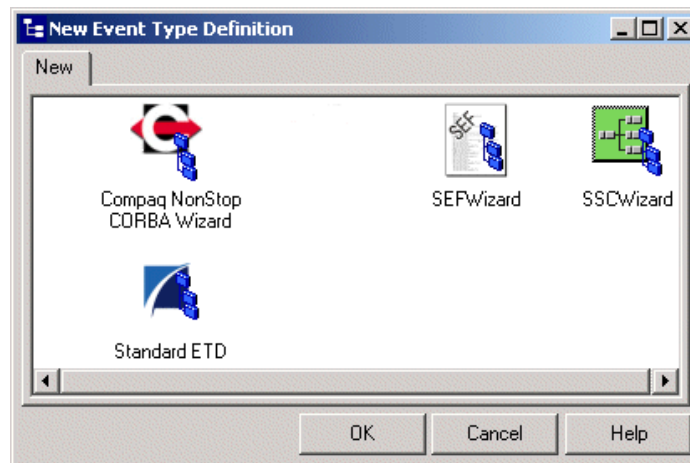
Note: Before using the Compaq NonStop CORBA Client ETD wizard, you must transfer the **.idl** files from the Compaq NonStop server to the machine running the e*Gate.

To create an ETD from a CORBA .idl file

- 1 From the e*Gate Schema Designer, display the ETD Editor. Be sure you have selected the Java editors as your default.
- 2 To access the Compaq NonStop CORBA Client ETD wizard, click **New** on the ETD Editor's **File** menu.

The New Event Type Definitions window appears, displaying all installed ETD wizards (see the example in Figure 2).

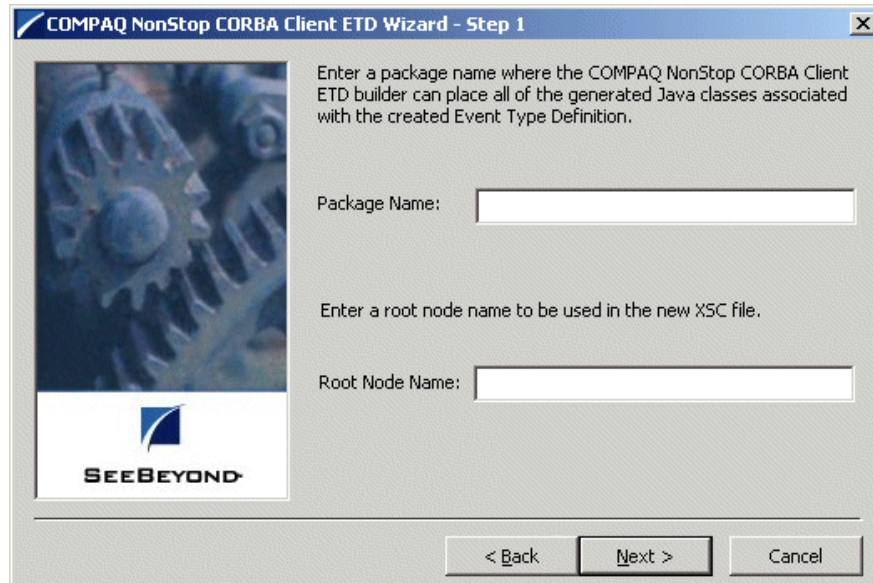
Figure 2 New Event Type Definition Window



- 3 Double-click the **CORBAWizard** icon.
- 4 Review the **CORBA Client ETD Wizard - Introduction** dialog box, then click **Next**. This dialog box gives you brief instructions on how to use the wizard.

The **Compaq NonStop CORBA Client ETD Wizard - Step 1** dialog box appears (see Figure 3).

Figure 3 Compaq NonStop CORBA Client ETD Wizard - Step 1 Dialog Box



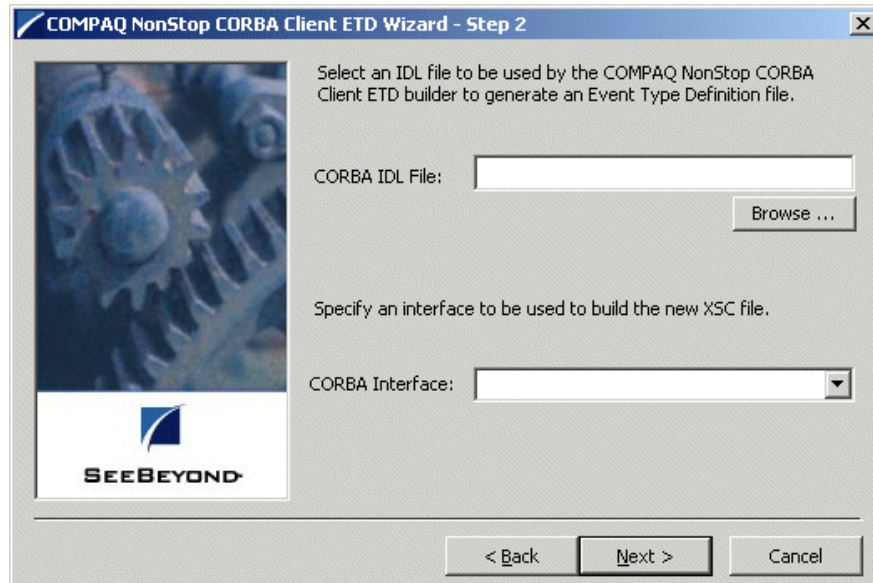
- 5 In the **Compaq NonStop CORBA Client ETD Wizard - Step 1** do the following actions:
 - ◆ Enter the desired package name for the container in which the wizard places the generated Java classes.
 - ◆ Enter the desired root-node name of the ETD.

Note: *Be sure to observe the required naming rules in these entries. See the **e*Gate Integrator User's Guide** for details.*

- 6 Click **Next** to continue.

The **Compaq NonStop CORBA Client ETD Wizard - Step 2** dialog box appears (see Figure 4).

Figure 4 Compaq NonStop CORBA Client ETD Wizard- Step 2 Dialog Box



- 7 In the **Compaq NonStop CORBA Client ETD Wizard - Step 2** do the following actions:
 - ♦ Enter the name of the **.idl** file. If you do not know the file name, click **Browse** to navigate to the desired file.
 - ♦ Select the desired interface (for the ETD) from the drop-down list.

CORBA Interfaces

A single **.idl** file can contain the description of one or more CORBA interfaces. These interfaces may or may not be contained within a module declaration in the **.idl** file.

Modules are sometimes used to convey the *scope* of interfaces. For example, there could be a module for **bank** with bank-related interfaces and there could be a **customer** module with customer-related interfaces. In this example, the bank-related interfaces can be considered *scoped* within the **bank** module, and the customer-related interfaces are *scoped* within the **customer** module.

Additionally, different modules can contain similarly named interfaces. For example, both the **bank** and **customer** modules could contain a **payment** interface. It is important to realize that each of the **payment** interfaces is totally unique. That is, the ETD for one interface cannot be used to access the other.

The wizard's drop-down list displays the interfaces with *scope* information, which are found in the **.idl** file, if there are any. Using the previous bank and customer example, the interfaces can then be presented as **bank::payment** and **customer::payment**. You must be sure to choose the correct interface.

See the following example:

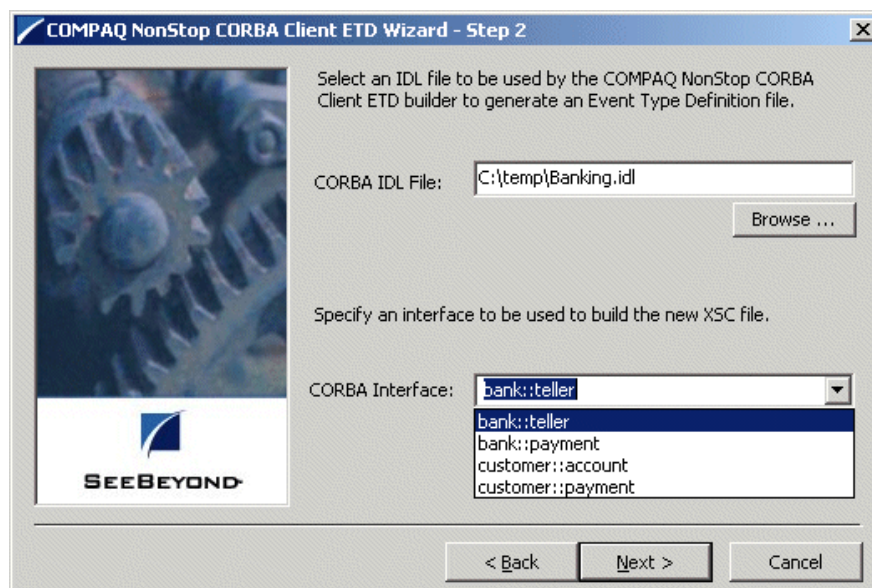
```
// File banking.idl
module bank {
    interface teller {
        ...
    };
    interface payment {
        ...
    };
}; // end of bank module

module customer {
    interface account {
        ...
    };
    interface payment {
        ...
    };
}; // end of customer module

// end of banking.idl
```

Figure 5 shows The **Compaq NonStop CORBA Client ETD Wizard - Step 2** dialog box, using the previous sample .idl file. Note the **CORBA Interface** drop-down menu.

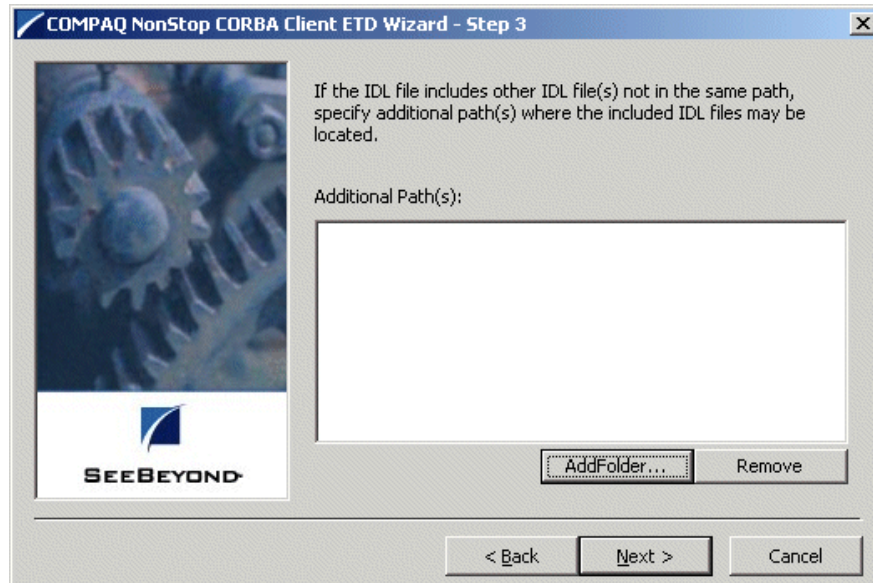
Figure 5 Compaq NonStop CORBA Client ETD Wizard- Step 2 Dialog Box: With .idl File



8 When you are finished, click **Next** to continue.

The **Compaq NonStop CORBA Client ETD Wizard - Step 3** dialog box appears (see Figure 6).

Figure 6 Compaq NonStop CORBA Client ETD Wizard- Step 3 Dialog Box



- 9 Use the **Compaq NonStop CORBA Client ETD Wizard - Step 3** to enter additional path information. This step is optional. If each interface referenced within the file you selected in step 7 is on the same path as that file (or no others are referenced), do nothing and go on to the next step.

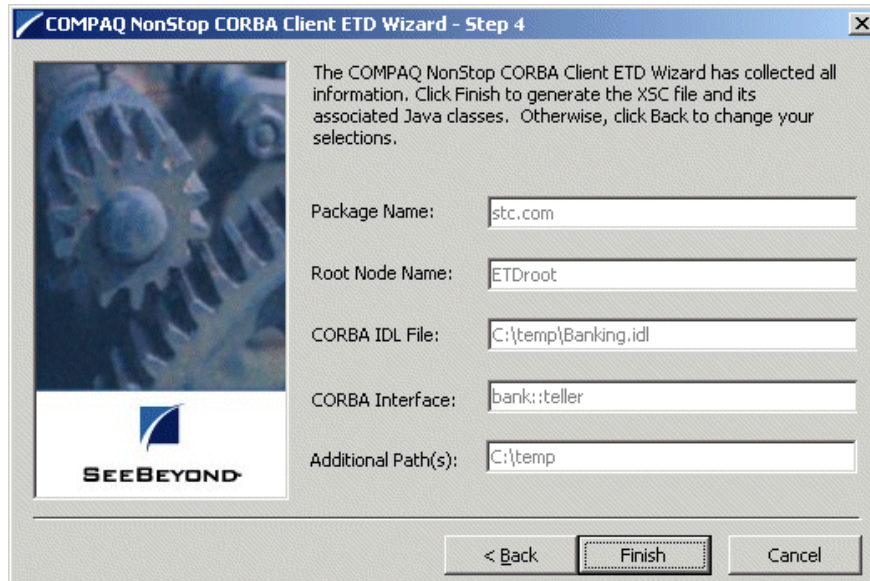
However, if your interface specifies the use of one or more other interfaces not on the path of the file you selected in step 7, you must enter the path information for each of these interfaces.

Note: *This additional step could be necessary because an **.idl** file can have an “include” directive. This directive instructs the **.idl** file-to-Java compiler to include the referenced **.idl** file during processing. If that included file is not on the same path as the selected **.idl** file, you must add that “include” path.*

- 10 When you are finished, click **Next** to continue.

The **Compaq NonStop CORBA Client ETD Wizard - Step 4** dialog box appears (see Figure 7).

Figure 7 Compaq NonStop CORBA Client ETD Wizard- Step 4 Dialog Box



- 11 Use the **Compaq NonStop CORBA Client ETD Wizard - Step 4** to review all the information you have entered and be sure it is correct. You can click **Back** to change previously entered information, if you want.
- 12 Click **Finish** when you are done with the wizard.
The structure of the ETD you have created appears in the ETD Editor's Main window.
- 13 Close the ETD Editor and save the ETD under the your desired name. This is your new CORBA Client ETD (.xsc) file.

5.3 Basic Schema Implementation

This section explains how to implement a basic sample schema for the Compaq NonStop CORBA Client e*Way. This schema demonstrates how to configure the essential features of the e*Way in a typical e*Gate environment.

5.3.1 CorbaHello Sample Schema: Overview

This section provides an overview of how to configure the sample basic schema and how the schema operates. The name of this schema is CorbaHello, and it is contained in the schema import file **CorbaHello.zip**.

Schema Operation

This sample schema does the following operations:

- Takes a file from an external CORBA system, with the text “Hello World, from Somewhere”
- Outputs the file to an external directory; the file, named **Hellooutput0.dat**, contains the same text

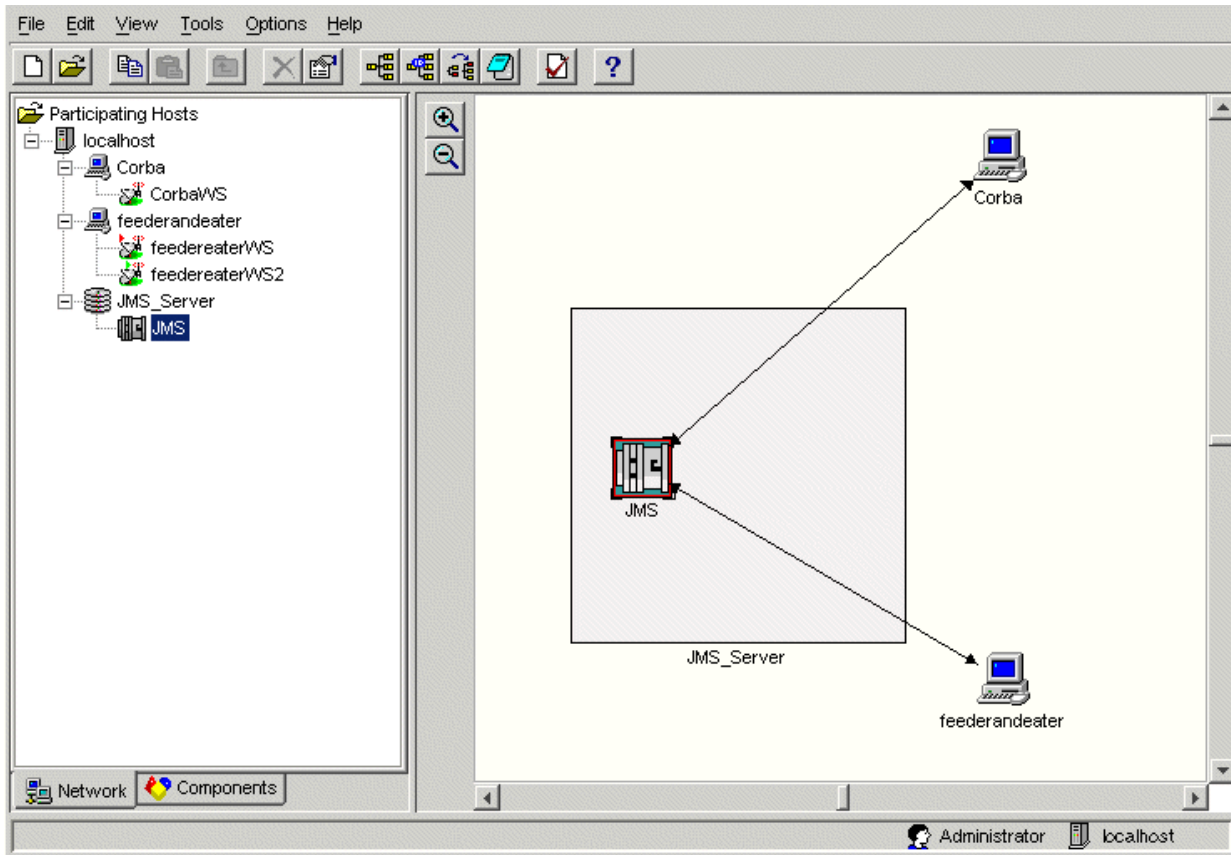
This schema uses the following **.idl** (**CorbaHello.idl**) file:

```
module demo
{
    module hello {
        interface GoodDay {
            string hello_simple();
        };
    };
};
```

Schema Setup

Figure 8 shows the schema's general architecture, using the e*Gate Schema Designer Network View.

Figure 8 CorbaHello Schema in Network View



Schema Components

This sample schema implementation consists of the following basic components:

- **feederandeater**: Inbound and outbound file e*Way that brings CORBA data into e*Gate and sends the output file to an external directory.
- **Corba**: Compaq NonStop CORBA Client e*Way (Multi-Mode) that process the CORBA data.
- **feederandeaterWS**, **feederandeaterWS2**, and **CorbaWS**: Collaborations for their respective e*Ways.
- **JMS_Server**: Java Messaging Service (JMS) Intelligent Queue (IQ) Manager that manages **JMSQ**.
- **corba**: e*Way Connection for the Compaq NonStop CORBA Client e*Way.
- **JMS**: e*Way Connection for the JMS Server.

Table 3 lists all the components for the schema, along with the logical names and configuration settings.

Table 3 CorbaHello Schema Components

Component	Logical Name	Settings
Schema	CorbaHello	
Control Broker	<i>localhost_cb</i>	
IQ Manager	JMS_Server	Start Up = Auto
Event Types	input	
	hello	
Java ETDs	input.xsc	<ul style="list-style-type: none"> ▪ Package name = inputPackage
	hello.xsc	<ul style="list-style-type: none"> ▪ Package name = com.stc.corbaclient.hellopackage
Collaboration Rules	c (for feedereaterWS)	<ul style="list-style-type: none"> ▪ Service = Pass Through ▪ Subscription = input ▪ Publication = input
	c (for feedereaterWS2)	<ul style="list-style-type: none"> ▪ Service = Pass Through ▪ Subscription = out ▪ Publication = input
	hello	<ul style="list-style-type: none"> ▪ Service = Java ▪ Input/output = int1; input.xsc (Trigger) ▪ Output = int2; hello.xsc (Trigger N/A)
Collaborations	feedereaterWS	<ul style="list-style-type: none"> ▪ Collaboration Rule = c ▪ Subscription = input Event from <EXTERNAL> ▪ Publication = input Event to JMS e*Way Connection
	feedereaterWS2	<ul style="list-style-type: none"> ▪ Collaboration Rule = c ▪ Subscription = out Event from JMS e*Way Connection ▪ Publication = input Event to <EXTERNAL>
	CorbaWS	<ul style="list-style-type: none"> ▪ Collaboration Rule = hello ▪ Subscription = input Event from JMS e*Way Connection ▪ Publication = <ul style="list-style-type: none"> ♦ hello Event to corba e*Way Connection ♦ out Event to JMS e*Way Connection
Java Collaboration Rule Classes	STCJavaPassThrough.class	<ul style="list-style-type: none"> ▪ e*Way = feederandeater ▪ Source = JavaPassThroughIn ▪ Destination = JavaPassThroughOut
	hello.class	<ul style="list-style-type: none"> ▪ e*Way = corba ▪ Source = int1 ▪ Destination = int2

Table 3 CorbaHello Schema Components (Continued)

Component	Logical Name	Settings
e*Way Connections	corba	<ul style="list-style-type: none"> ▪ 100 for Event Type "get" interval
	JMS	<ul style="list-style-type: none"> ▪ 100 for Event Type "get" interval
Inbound and outbound e*Way	feederandeater	<ul style="list-style-type: none"> ▪ Executable = stcewfile.exe ▪ Configuration file = feederandeater.cfg ▪ Start Up = Auto ▪ Collaborations = feedereaterWS and feedereaterWS2
Multi-Mode e*Way	Corba	<ul style="list-style-type: none"> ▪ Executable = stceway.exe ▪ Configuration file = Corba.cfg ▪ Start Up = Auto ▪ Collaboration = CorbaWS

5.3.2 Sample Schema: General Implementation Steps

This section explains how to import Compaq NonStop CORBA Client e*Way sample schemas and how to run the schemas.

Before Schema Implementation

To use and implement a sample schema, the e*Way and sample schema must be installed, all of the necessary files and scripts must be located in the default location, and you must also have access to a CORBA-based network.

Importing a Schema

To import a sample schema

- 1 Copy the desired **.zip** file from the **\Samples** directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
- 2 Start the e*Gate Schema Designer.
- 3 On the **Open Schema from Registry Host** dialog box, click **New**.
- 4 On the **New Schema** dialog box, click **Create from export**, and then click **Find**.
- 5 On the **Import from File** dialog box, browse to the directory that contains the sample schema.
- 6 Click the **.zip** file then click **Open**.

The sample schema is installed.

Running a Schema

To run a sample schema

- From the command line prompt, enter:

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

Substitute *hostname*, *username*, *schemaname*, and *user password* as appropriate.

The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, the final output directory contains the payload data.

5.3.3 CorbaArrays Sample Schema: Overview

This section provides an overview of how to configure the sample basic schema and how the schema operates. The name of this schema is CorbaArrays, and it is contained in the schema import file **CorbaArray.zip**.

Schema Operation

This sample schema does the following operations:

- Takes a file from an external CORBA system, which it uses to trigger the CORBA server
- Outputs the file to an external directory; the file, named **Arraysoutput0.dat**, contains the integers **42, 34, 13**

This schema uses the following **.idl** (**CorbaArrays.idl**) file:

```
module arrays  
{  
    interface MyServer  
    {  
        const short arrayBound = 3;  
  
        typedef MyServer servers[2];  
        typedef long numbers[arrayBound];  
        typedef sequence <MyServer> serfs;  
        typedef sequence <long,4> nos;  
        typedef sequence <string> strings;  
  
        struct arrayContainer {  
            short shorty[7][arrayBound];  
        };  
};
```

```

numbers write(in string s, in numbers n);
nos write2(in string s, in nos n);

// note that the following operation will be
// renamed to _notify by the IDL compiler
// because of the name clash with java.lang.Object's
notify()

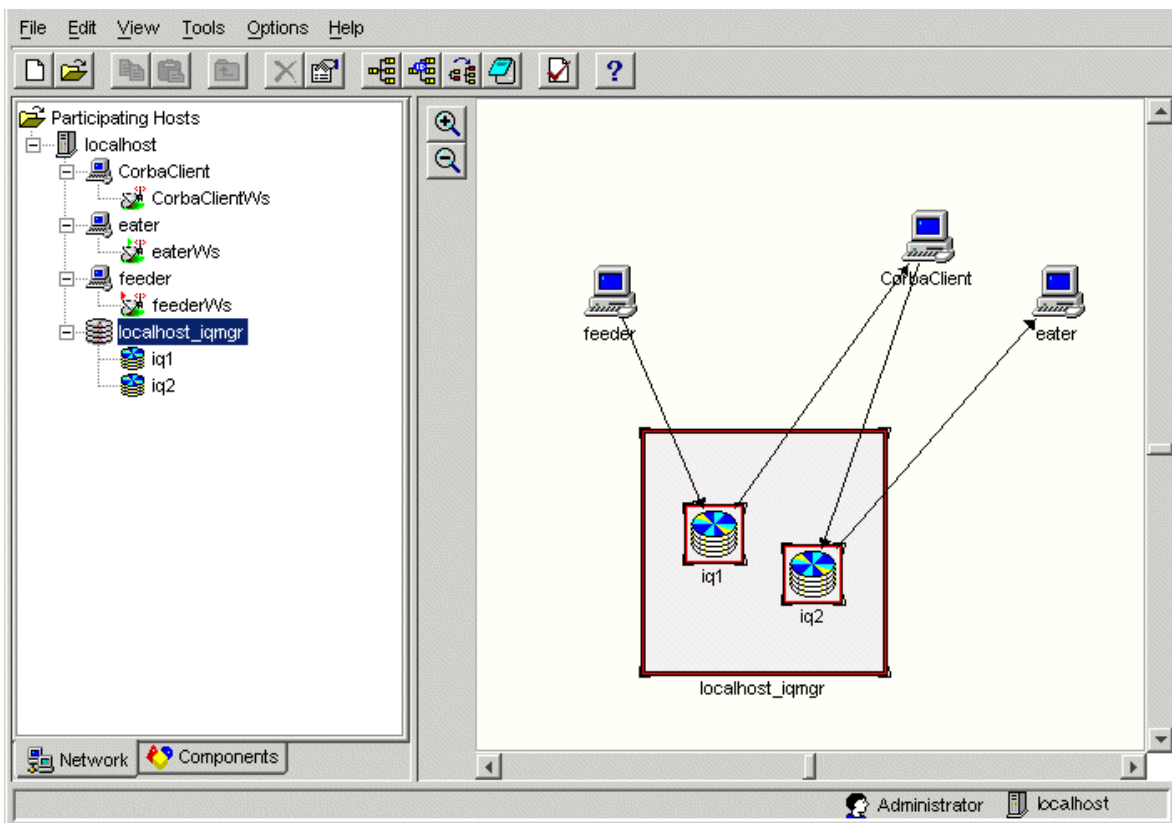
oneway void notify(in servers svcs);
oneway void notify2(in serfs svcs);
oneway void notify3(in arrayContainer ac);
};
};

```

Schema Setup

Figure 9 shows the schema's general architecture, using the e*Gate Schema Designer Network View.

Figure 9 CorbaArrays Schema in Network View



Schema Components

This sample schema implementation consists of the following basic components:

- **feeder** and **eater**: Inbound and outbound file e*Ways that bring CORBA data into e*Gate and send the output file to an external directory.
- **CorbaClient**: Compaq NonStop CORBA Client e*Way (Multi-Mode) that processes the CORBA data.
- **feederWs**, **eaterWs**, and **CorbaClientWs**: Collaborations for their respective e*Ways.
- **iq1** and **iq2**: Intelligent Queues (IQs) that the **feeder** e*Way publishes to (**iq1**) and the **eater** e*Way subscribes to (**iq2**).
- **localhost_iqmgr**: IQ Manager that manages **iq1** and **iq2**.
- **Corba**: e*Way Connection for the Compaq NonStop CORBA Client e*Way.

Table 4 lists all the components for the schema, along with the logical names and configuration settings.

Table 4 CorbaArrays Schema Components

Component	Logical Name	Settings
Schema	CorbaArrays	
Control Broker	<i>localhost_cb</i>	
IQ Manager	<i>localhost_iqmgr</i>	Start Up = Auto
Event Types	in	
	array	
Java ETDs	in.xsc	▪ Package name = inPackage
	arrays.xsc	▪ Package name = com.stc.corbaclient.arrayspackage
Collaboration Rules	CC (for feedWs)	▪ Service = Pass Through ▪ Subscription = in ▪ Publication = in
	CC (for eaterWs)	▪ Service = Pass Through ▪ Subscription = in ▪ Publication = in
	arraysCollab	▪ Service = Java ▪ Input/output = arrays1; in.xsc; (Trigger) ▪ Output = arrays2; arrays.xsc; (Trigger N/A)

Table 4 CorbaArrays Schema Components (Continued)

Component	Logical Name	Settings
Collaborations	feederWs	<ul style="list-style-type: none"> ▪ Collaboration Rule = CC ▪ Subscription = in Event from <EXTERNAL> ▪ Publication = in Event to iq1
	eaterWs	<ul style="list-style-type: none"> ▪ Collaboration Rule = CC ▪ Subscription = in Event from CorbaClientWs ▪ Publication = in Event to <EXTERNAL>
	CorbaClientWs	<ul style="list-style-type: none"> ▪ Collaboration Rule = arraysCollab ▪ Subscription = in Event from feederWs ▪ Publication = <ul style="list-style-type: none"> ♦ array Event to Corba e*Way Connection ♦ in Event to iq2
Java Collaboration Rule Classes	STCJavaPassThrough.class	<ul style="list-style-type: none"> ▪ e*Way = feeder ▪ Source = JavaPassThroughOut ▪ Destination = JavaPassThroughIn
	STCJavaPassThrough.class	<ul style="list-style-type: none"> ▪ e*Way = eater ▪ Source = JavaPassThroughIn ▪ Destination = JavaPassThroughOut
	CorbaClientWs.class	<ul style="list-style-type: none"> ▪ e*Way = CorbaClient ▪ Source = arrays1 ▪ Destination = arrays2
e*Way Connection	Corba	<ul style="list-style-type: none"> ▪ 100 for Event Type "get" interval
Inbound e*Way	feeder	<ul style="list-style-type: none"> ▪ Executable = stcewfile.exe ▪ Configuration file = feeder.cfg ▪ Start Up = Auto ▪ Collaboration = feederWs
Outbound e*Way	eater	<ul style="list-style-type: none"> ▪ Executable = stcewfile.exe ▪ Configuration file = eater.cfg ▪ Start Up = Auto ▪ Collaboration = eaterWs
Multi-Mode e*Way	CorbaClient	<ul style="list-style-type: none"> ▪ Executable = stceway.exe ▪ Configuration file = CorbaClient.cfg ▪ Start Up = Auto ▪ Collaboration = CorbaClientWs

Note: For details on how to import and run the schema, see [“Sample Schema: General Implementation Steps”](#) on page 48.

5.4 ETD Structure

This section explains the structure and layout of the Compaq NonStop CORBA Client ETD (.xsc file). Figure 10 shows an example of this ETD in the e*Gate Schema Designer ETD Editor Main window, with the two top-level sub-nodes.

Figure 10 Compaq NonStop CORBA Client ETD in ETD Editor: Top-level Nodes

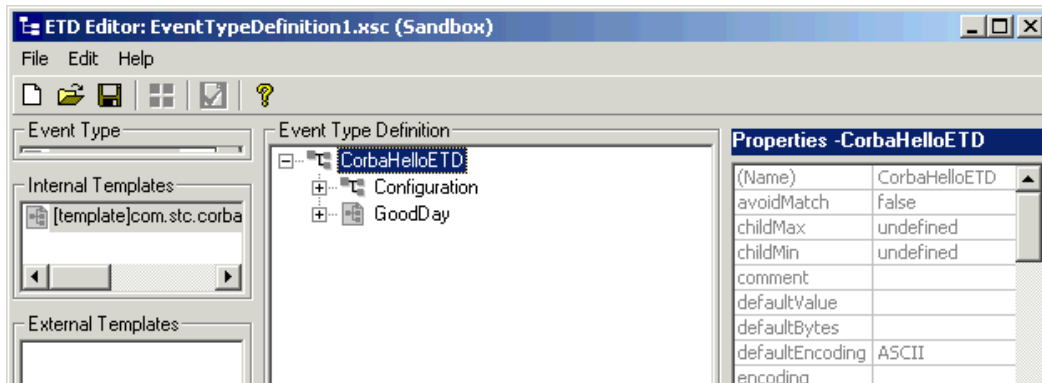
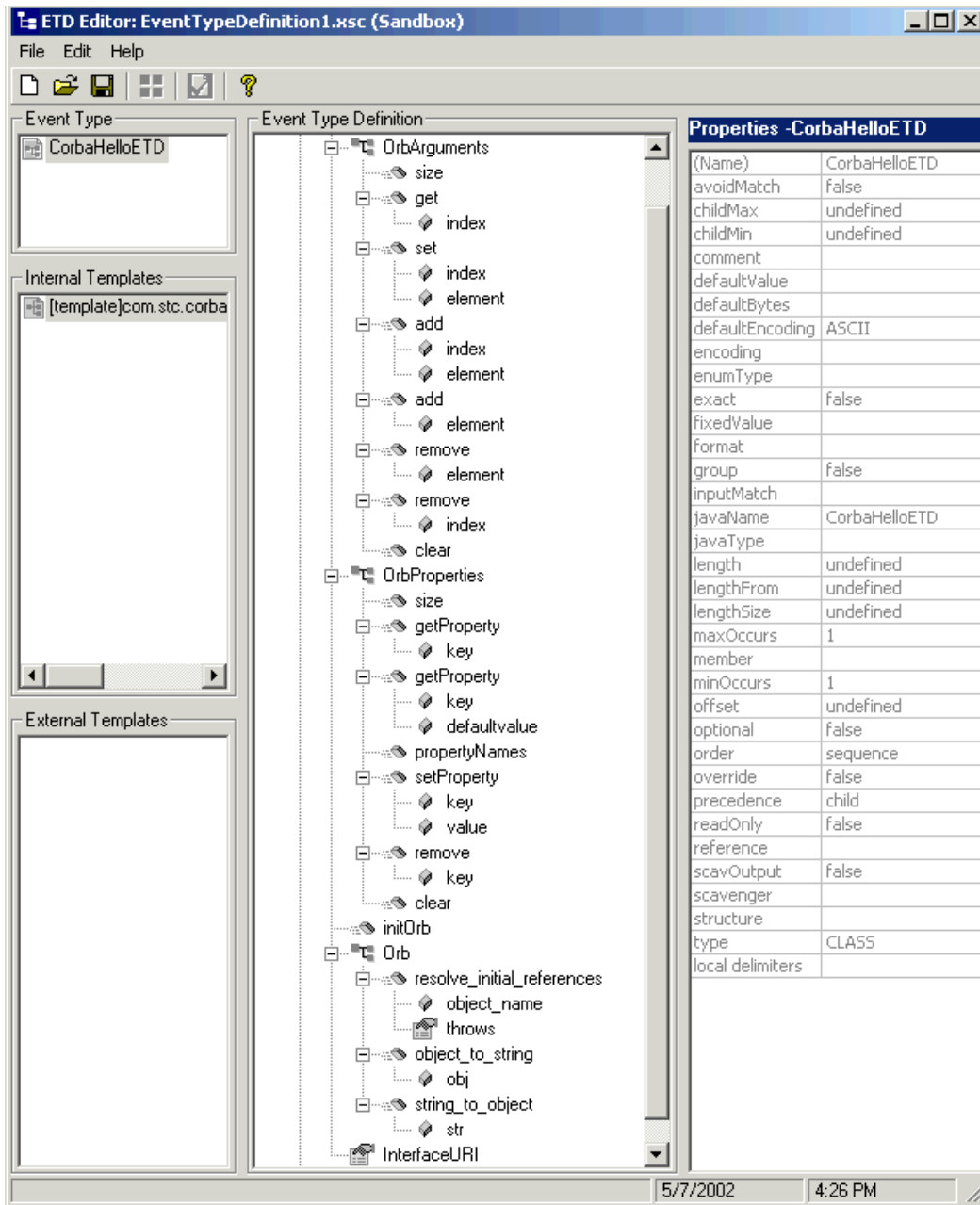


Figure 11 shows the sub-nodes available under the **Configuration** node.

Figure 11 Compaq NonStop CORBA Client ETD in ETD Editor: Configuration Node



The rest of this section explains the general structure of this ETD. This structure (see [Figure 11 on page 54](#)) is divided into the following basic categories, represented by the top-level ETD sub-nodes:

- Configuration nodes
- Object nodes

Configuration Nodes

The **Configuration** node and its sub-nodes include all configuration parameters (see [Chapter 4](#) for details) for the ORB, as well as the ORB initialization methods. These nodes remain the same for every ETD you generate. What you enter under the **OrbArguments** and **OrbProperties** parameters depends on the type of client ORB you are using.

The **Configuration** node has the following sub-nodes:

- **GuestMgr node:** Contains the interface you selected in the wizard taken from the input interface (.idl) file (see [“Using the Compaq NonStop CORBA Client ETD Wizard” on page 38](#)).
- **Method nodes:** Are different depending on the input interface. Use the desired methods to meet the input interface’s syntax requirements (see [“Building Collaboration Rules” on page 55](#) for more information).
- **Return value node:** If there is a return value for the method, a **ReturnValue** node contains the value that results after you call **execute**.

Note: For more information on ETD nodes and methods, see the *e*Gate Integrator User’s Guide*.

Object Nodes

The exact name and structure of this node in the ETD is derived from the CORBA interface. In the example in [Figure 11 on page 54](#), this node is named **GoodDay**.

The following example shows the .idl file (**CorbaHello.idl**) used for the ETD:

```
module demo
{
    module hello {
        interface GoodDay {
            string hello_simple();
        };
    };
};
```

The purpose of the “Object” node is to provide a graphical representation of the interface. Expanding the node reveals all the methods and attributes on the interface, which themselves are represented as nodes. A node representing a method is normally expandable and reveals all the parameters for the method as sub-nodes, as well as the return value (if present) as a node.

Building Collaboration Rules

All methods of a particular interface are different because interfaces have varying syntax requirements. How you call certain methods depends on these requirements. Methods and attributes are exposed in the ETD to allow you to design its structure to meet the necessary requirements.

For example, using the drag-and-drop feature of the e*Gate Schema Designer's Collaboration Rules Editor, you can populate the parameter nodes as appropriate (if any) and call the **execute** method of the node.

These available methods enable you to build the Collaboration Rules using simple drag-and-drop actions. The rest of this section provides general procedures for how to create your desired CORBA Client Business Rules.

To use methods to build a Collaboration Rule

- 1 Using the Schema Designer, open the Collaboration Rules Editor.
- 2 From the Collaboration Rules Editor Main window, open the desired Collaboration Rule file.

The window shows both ETD structures, the source and destination. Methods are available using the Business Rule toolbar.

Note: For details on how to use the Collaboration Rules Editor, see the *e*Gate Integrator User's Guide*.

- 3 Using the Main window drag and drop methods from the source to the destination ETD to assign input parameters for the method.
- 4 Using the Business Rule toolbar, call and execute the method.
- 5 Drag and drop the return or out parameters, if applicable.
- 6 When you are finished, compile and save the Business Rule then exit the Collaboration Rules Editor.

Note: If there is a return value for the method, a **ReturnValue** node contains that value. You can drag and drop it to a different node, if desired.

Dynamic Configuration

For the dynamic configuration feature, the e*Way Connection configuration maps onto the **Configuration** node in the ETD as follows:

- **ORB Arguments** = **OrbArguments** node
- **ORB Properties** = **OrbProperties** node
- **ORBClass** = An entry in **OrbProperties** with the key **org.omg.CORBA.ORBClass**
- **ORBSingletonClass** = An entry in **OrbProperties** with the key **org.omg.CORBA.ORBSingletonClass**

Interface URI

The **InterfaceURI** node is not directly referenced in the e*Way Connection configuration. In the ETD, it is whatever mechanism you specified in that configuration (see [Chapter 4](#) for details on e*Way Connection configuration).

For example, if you chose the **Manual URI** parameter/mode, the **InterfaceURI** value is the string you entered. If you chose **file**, **InterfaceURI** is the path and file name entered. If you chose **corbaname**, it is the composite of your entries under the **corbaname** configuration parameters.

Dynamic Configuration Examples

OrbArguments is a part of `java.util.ArrayList`, and **OrbProperties** is a part of `java.util.Properties`. The rest of this section gives and explains some examples of their use.

To add a further ORB argument, use the **add** method on **OrbArguments**, for example:

```
getOrbArguments().add("-ORBprofile");
```

To add or override an ORB property, use the **setProperty** method on the **OrbProperties** node, for example:

```
getOrbProperties().setProperty("org.omg.CORBA.ORBClass",  
"com.iona.corba.art.artimpl.ORBImpl");
```

You can use the **initOrb** method manually to initialize or re-initialize an ORB. Otherwise, if no initialized ORB is present upon the first use of the CORBA interface in the ETD, the ORB is initialized automatically. In such cases, all the settings in the **OrbArguments** and **OrbProperties** are used to initialize the ORB. The resulting ORB reference is **setProperty** in the **Orb** node.

The **Orb** node represents the client ORB instance. As described earlier in this section, this node is populated when the ORB is initialized automatically or manually in the ETD, or it can be set to a user-obtained instance of it. You can use this ORB reference to access the functionality exposed by the `org.omg.CORBA.ORB` class.

The value in the **InterfaceURI** node is used to find and resolve the CORBA object implementing the CORBA interface represented by the ETD. This process happens upon the first access to the interface functionality in the ETD.

Note: For details on the e*Way's Java methods, see [Chapter 6](#).

In Case of a CORBA SystemException

Once a URI has been resolved, and an object's IOR has been narrowed, the ETD continues to use that same object. The ORB handles reconnecting if it becomes necessary.

However, your server could be set up in such a way that, if it is restarted, the existing IOR becomes invalid, and a new object with a different IOR must be used. In such cases, for example, if the Server restarts on a different port and no implementation repository is used, the ETD must be able to recognize the new IOR. If it does not, a CORBA **SystemException** error is thrown.

To enable this recognition, you must add code to the appropriate Collaboration Rule to make the ETD reevaluate the URI. After this reevaluation, the ETD then connects to the new object without your having to restart the e*Way.

The following code sample shows how to accomplish this operation when a CORBA **SystemException** is thrown (**MyServer** is the interface and **MyETD** is the ETD instance name):

```
try {
    // My collab code
} catch (org.omg.CORBA.SystemException ex) {
    // The following will force the interface to be looked up/
    // resolved again before the next time it is used
    getmyETD().setMyServer((com.stc.corbaclient.mypckg.stubs.a
        rrays.MyServer)null);
    throw ex;
}
```

This work-around has the effect that, the next time the Collaboration is run and this interface is accessed, the IOR is retrieved/resolved from the start, according to the reset URI.

The null has to be typecast in this work-around because there are two **set** methods to choose from. If you compile without the typecast, the compiler gives you an error message. For the previous example, such an error could be:

reference to setMyServer is ambiguous ...

In addition, this error string includes the two argument types you must choose from, which are:

- Wrapper to support the ETD
- Type of the underlying stub for the interface

You can use either type. In the previous work-around example, the type of the underlying stub is used.

e*Way Java Methods

This chapter explains the Java classes and methods contained in the e*Way Intelligent Adapter for Compaq NonStop CORBA Client, which are used to extend the functionality of the e*Way.

6.1 e*Way Methods and Classes: Overview

For any e*Way, communication takes place both on the e*Gate system and the external system side. Communication between the e*Way and the e*Gate environment is common to all e*Ways, while the communication between the e*Way and the external system is different for each e*Way.

For the Compaq NonStop CORBA Client e*Way, the **stceway.exe** file (Multi-Mode e*Way, see [Chapter 3](#)) is used to communicate between the e*Way and e*Gate, and a Java Collaboration is utilized to keep the communication open between the e*Way and the CORBA client.

Using Java Methods

Java methods have been added to make it easier to set information in the Compaq NonStop CORBA Client e*Way Event Type Definition (ETD), as well as get information from it. The nature of this data transfer depends on the configuration parameters (see [Chapter 3](#)) you set for the e*Way in the e*Gate Schema Designer's e*Way Editor. These parameters in turn must be compatible with those of the Object Request Broker (ORB) type you are using.

Note: For more information on the CORBA Client ETD structure, its nodes, and attributes (within an e*Gate .xsc file), see [“ETD Structure” on page 53](#).

The Schema Designer's Collaboration Editor allows you to call Java methods by dragging and dropping an ETD node into the **Rules** dialog box.

Note: The node name can be different from the Java method name.

After you drag and drop, the actual conversion takes place in the .xsc file. To view the .xsc file, use the Schema Designer's ETD Editor and Collaboration Rules Editor.

For example, if the node name is **Orb**, the associated **javaName** is **Orb**. If you want to get the node value, use the Java method called **getOrb**. If you want to set the node value, use the Java method called **setOrb**.

Java Classes

The Java methods for the Compaq NonStop CORBA Client e*Way are contained in the following classes:

- [“CorbaClientConfiguration Class” on page 60](#)
- [“CorbaClientETDBase Class” on page 64](#)

6.2 CorbaClientConfiguration Class

The **CorbaClientConfiguration** class is defined as:

```
public final class CorbaClientConfiguration
```

These methods handle the properties loaded from the e*Way Connection configuration and provide the implementation for the functionality exposed in the Compaq NonStop CORBA Client e*Way **Configuration** node to override/add to that node dynamically in a Collaboration Rule.

The **CorbaClientConfiguration** class extends **java.lang.Object**.

The **CorbaClientConfiguration** methods are:

[“getOrb” on page 60](#)

[“setOrb” on page 61](#)

[“getInterfaceURI” on page 61](#)

[“setInterfaceURI” on page 61](#)

[“getOrbArguments” on page 62](#)

[“setOrbArguments” on page 62](#)

[“getOrbProperties” on page 63](#)

[“setProperties” on page 63](#)

getOrb

Description

Retrieves the reference to the ORB and initializes the ORB “on demand,” if there is no ORB reference set yet.

Syntax

```
public org.omg.CORBA.ORB getOrb()
```

Parameters

None.

Returns

Object

The ORB reference.

Throws

None.

setOrb

Description

Sets the ORB reference to a user-obtained instance of that reference.

Syntax

```
public void setOrb(org.omg.CORBA.ORB anOrbReference)
```

Parameters

Name	Type	Description
anOrbReference	Object	The reference to an ORB.

Returns

None.

Throws

None.

getInterfaceURI

Description

Retrieves the currently set URI used to locate the CORBA object.

Syntax

```
public java.lang.String getInterfaceURI()
```

Parameters

None.

Returns

java.lang.String

The currently set URI, loaded from the e*Way Connection configuration or overridden in the current Collaboration Rule.

Throws

None.

setInterfaceURI

Description

Set the URI to a user-obtained instance that URI.

Syntax

```
public void setInterfaceURI(java.lang.String aURI)
```

Parameters

Name	Type	Description
aURI	java.lang.String	The URI to set. Supported formats depend on the client ORB used at runtime; all must support IOR: , and if they support the INS, also corbaloc: and corbaname: .

Returns

None.

Throws

None.

getOrbArguments

Description

Retrieves the currently set ORB arguments used to initialize the ORB in **initOrb**. The settings are loaded from the e*Way Connection configuration or overridden in the current Collaboration Rule.

Syntax

```
public java.util.ArrayList getOrbArguments()
```

Parameters

None.

Returns

java.util.ArrayList

The currently set array list containing the ORB arguments. Updating the returned reference changes the instance in this class.

Throws

None.

setOrbArguments

Description

Sets the ORB arguments to a user-obtained instance those arguments.

Syntax

```
public void setOrbArguments(java.lang.String[] args)
```

Parameters

Name	Type	Description
args	java.lang.String	An array of arguments to pass to the ORB initialization.

Returns

None.

Throws

None.

getOrbProperties

Description

Retrieves the ORB properties.

Syntax

```
public java.util.Properties getOrbProperties()
```

Parameters

None.

Returns

java.util.Properties

The properties to be passed to the ORB initialization.

Throws

None.

setProperties

Description

Sets the ORB properties to a user-obtained instance those properties.

Syntax

```
public void setProperties(java.util.Properties props)
```

Parameters

Name	Type	Description
props	java.util.Properties	The properties to pass to the ORB initialization.

Returns

None.

Throws

None.

6.3 CorbaClientETDBase Class

The **CorbaClientETDBase** class is defined as:

```
public class CorbaClientETDBase
```

Provides the ETD implementation for the Compaq NonStop CORBA Client e*Way.

The **CorbaClientETDBase** class extends **com.stc.jcsre.SimpleETDImpl**.

The **CorbaClientETDBase** methods are:

[“getConfiguration” on page 64](#)

getConfiguration

Description

The accessor to a configuration instance in the ETD field **Configuration**. Only the retrieving method is provided, because you cannot replace the configuration instance itself.

Syntax

```
public CorbaClientConfiguration getConfiguration()
```

Parameters

None.

Returns

Object

The Configuration instance used by this ETD implementation.

Throws

None.

Index

C

- classes, Java
 - CorbaClientConfiguration 60
 - CorbaClientETDBase 64
- CLASSPATH Append From Environment Variable 19
- Classpath Override 19
- Classpath Prepend 18
- Collaboration Rules, building 55
- configuration nodes 55
- configuration parameters
 - Object 25
 - ORB 33
- Configuring 23
- CORBA types not supported 11
- CORBA types supported 10

D

- Disable JIT 21
- dynamic configuration 56

E

- e*Way Connection configuration, overview 23
- e*Way operation, general 7
- ETD structure, overview 53
- Event Type Definitions and Collaborations 8
- external system requirements 12

G

- getConfiguration 64
- getInterfaceURI 61
- getOrb 60
- getOrbArguments 62
- getOrbProperties 63

I

- implementation overview 38
- Initial Heap Size 19
- installation
 - UNIX 14

- Windows 13
- intended reader 7

J

- Java Methods 59–64
- Java methods and classes, overview 59
- Java methods, using 59

M

- Maximum Heap Size 19
- methods, Java
 - getConfiguration 64
 - getInterfaceURI 61
 - getOrb 60
 - getOrbArguments 62
 - getOrbProperties 63
 - setInterfaceURI 62
 - setOrb 61
 - setOrbArguments 63
 - setProperties 63

O

- Object configuration 25
- object nodes 55
- ORB configuration 33
- ORB implementation 8

S

- sample CorbaArrays schema
 - components 51
 - operation 49
 - overview 49
 - setup 50
- sample CorbaHello schema
 - components 46
 - operation 45
 - overview 44
 - setup 46
- sample schemas, importing and running 48
- setInterfaceURI 62
- setOrb 61
- setOrbArguments 63
- setProperties 63
- supported operating systems 11
- Suspend Option for Debugging 21
- system requirements 11
- SystemException error thrown 57

W

wizard, Compaq NonStop CORBA Client, using 38