SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Blue Martini User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

SEEBEYOND®

Contents

Chapter 1

Introduction

1.1 Overview

Blue Martini Software's Customer Interaction System (CIS) provides applications and services that enable companies to build brand equity through direct customer interaction across Internet-related customer "touch-points", such as websites, mobile wireless devices and on-line trading exchanges and traditional customer touch-points, such as stores and call centers. This approach supports increased customers revenues by coordinating customer interactions across these touch-points.

Blue Martini version 4 consists of four main applications called:

- Channels
- Marketing
- Commerce
- Service

All are based around the application server architecture.

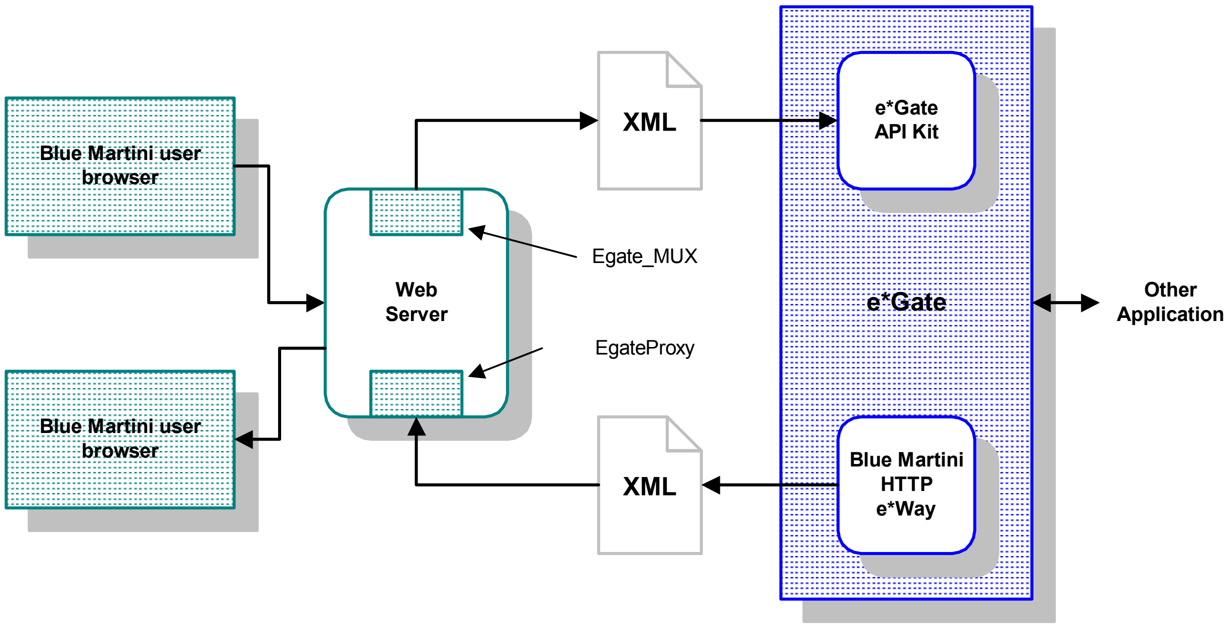
The applications allow organizations to manage product catalogs, handle customer transactions and track customer behavior. Each application consists of a number of functional modules including "catalog management", "data warehousing and reporting", "customer management" and so on. Other modules can be installed separately as needed.

Blue Martini's user model is based on the concept of Business Objects and Business Actions. Customers can reuse, create or modify Business Objects, and create Business Actions to perform work upon those objects. For example, an "Account" is a Business Object with associated Business Actions called "CreateAccount", "GetAccountDetail" and "UpdateAccount". Business actions typically take a list of business objects as input and return a list of business objects as output.

The Blue Martini e*Way provides both synchronous and asynchronous message exchange. The e*Way supports the following basic architectures:

- Synchronous request/reply from Blue Martini Server
- Asynchronous message from Blue Martini Server
- Synchronous request/reply into Blue Martini Server
- Asynchronous message into Blue Martini Server

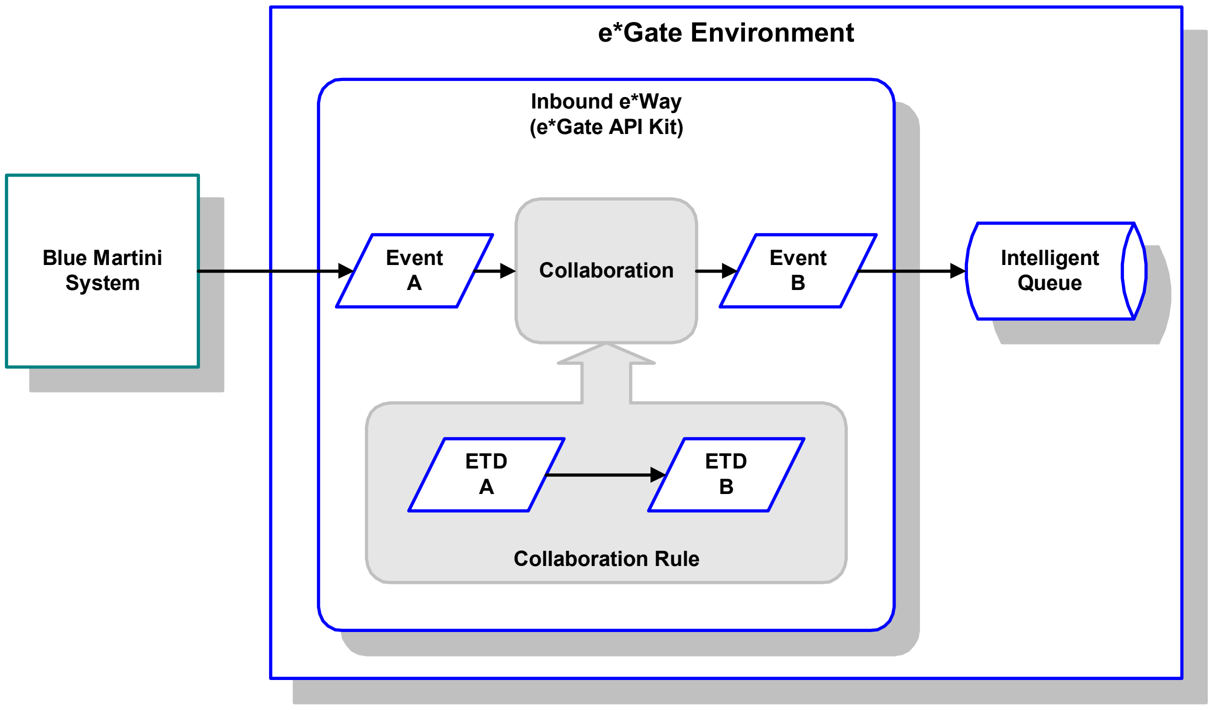
Figure 1 e*Gate Integration



1.2 e*Way Operation

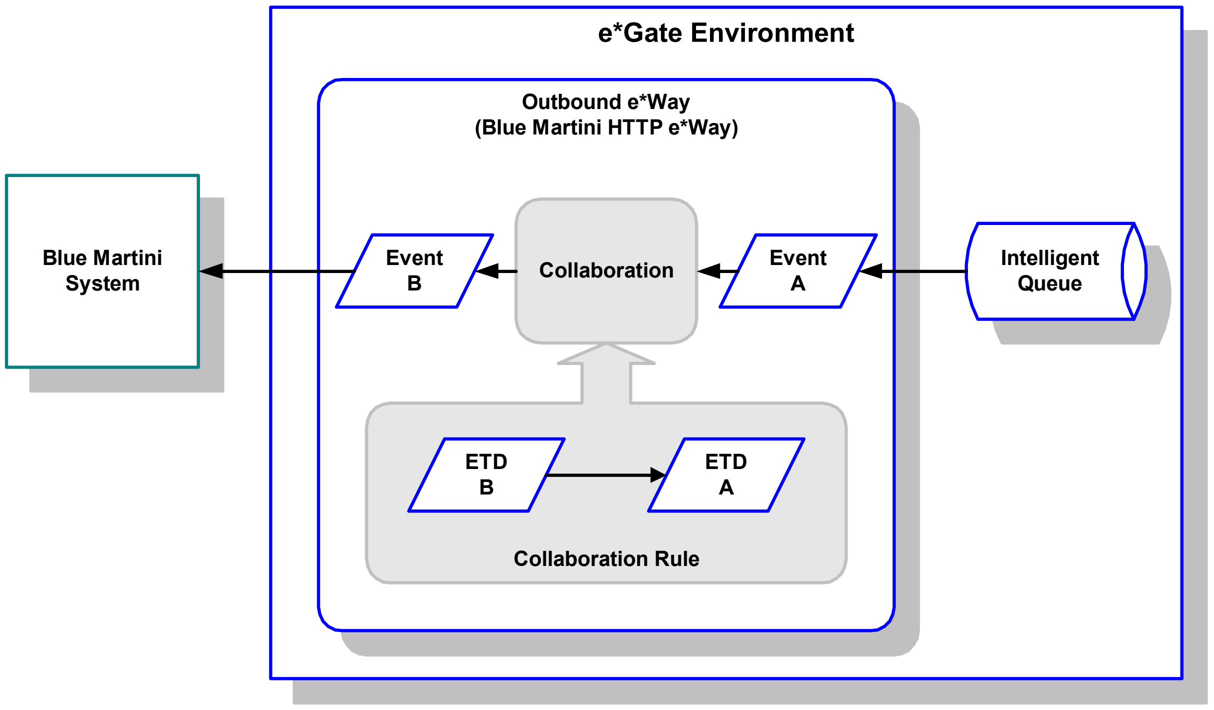
The e*Way uses Java methods to start and stop scheduled operations, exchange data with the external system, package data as e*Gate *Events*, send those Events to Collaborations, and manage the connection between the e*Way and the external system (see Figure 2).

Figure 2 Typical e*Way Data Flow (Inbound)



The e*Way uses Java methods to start and stop scheduled operations, exchange data with the external system, package data as e*Gate *Events*, send those Events to Collaborations, and manage the connection between the e*Way and the external system (see Figure 3).

Figure 3 Typical e*Way Data Flow (Outbound)



1.2.1 Event Type Definitions and Collaborations

Event Type Definitions (ETDs) embody the business rules relating to the associated external application. It is the responsibility of the user to define the format of the data to be passed between systems. We recommend using XML. Currently there is no mechanism for automatically creating DTDs or XSC files from existing Blue Martini DNA files. The user must create a ETD structure based on the content of the DNA file to be used. This structure is then converted via a build wizard into an XSC file.

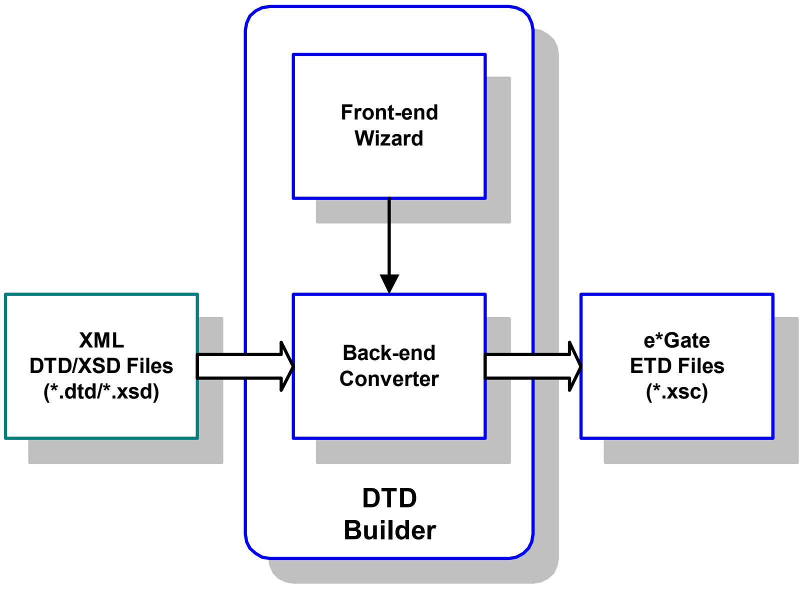
This can be accomplished in a number of ways. For example,

- The Event Type Definition Editor can be used to create the XSC file representing a DNA list from scratch.
- The ETD Editor can be used to create a simple delimited structure representing specific nodes from the DNA list.
- A text editor could be used to create the DTD structure, which could then be translated via an e*Gate wizard into an XSC file.

Note: *There are external tools that create either DTDs or XSD files.*

Once the formatted ETD exists, the diagram below demonstrates the translation to an XSC file.

Figure 4 DTD Builder Operation



Collaborations execute the business logic that enable the e*Way to do its intended work. In turn, each Collaboration executes a specified Collaboration Rule, which contains the actual instructions to execute the business logic. Each e*Way requires one or more Collaborations, and each Collaboration requires one or more IQs to which its processed Events are published (Inbound e*Way), or one to which it subscribes for incoming Events (Outbound e*Way).

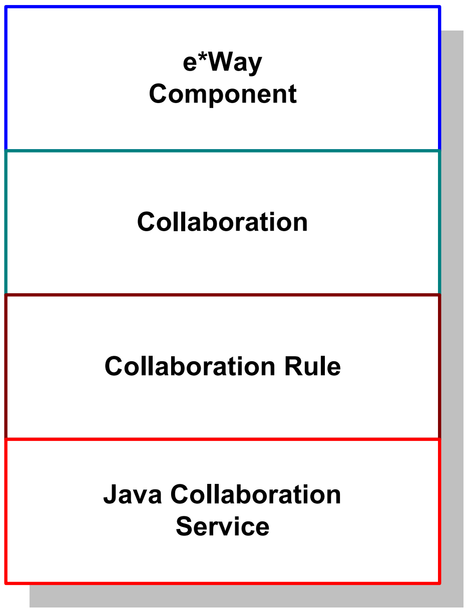
Each Collaboration uses a Collaboration Rule, which references the appropriate ETDs, and each Collaboration Rule uses a Collaboration Service.

1.2.2 The Java Collaboration Service

The Java Collaboration Service provides the foundation for the e*Way and its associated Collaborations to operate. This may be represented by a layered structure (seeFigure 5). Since there is not a direct link between the e*Way and the Collaboration Service, all intermediate components must be configured correctly.

The Java Collaboration Service makes it possible to develop external Collaboration Rules that will execute e*Gate business logic using Java code. Using the Java Collaboration Editor, you create Java classes that utilize the **initialize()**, **translate()**, and **terminate()** methods.

Figure 5 Component Relationship



For more information on the Java Collaboration Service, see the *e*Gate Integrator Collaboration Services Reference Guide*. For more information on the Java ETD Editor and the Java Collaboration Editor, see the *e*Gate Integrator User's Guide*.

1.3 System Requirements

To use the Blue Martini e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection.
- A computer running Windows, to allow you to use the e*Gate Schema Designer and ETD Editor
- 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.

Installed on the Participating Host

- Java JDK version 1.3.1

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

1.4 Supported Operating Systems

The Blue Martini e*Way is available on the following operating systems:

- Windows 2000 and Windows Server 2003
- Sun Solaris 8

1.5 External System Requirements

- Blue Martini Customer Interaction System 4.0 patch 1

Chapter 2

Installation

This chapter describes the requirements and procedures for installing the e*Way Intelligent Adapter for Blue Martini. Following installation, you must configure it for your system and incorporate it into a schema, as described in **Chapter 5**.

2.1 Installing the e*Way

2.1.1 Windows 2000

Note: ***Do not*** *edit any installation scripts or change the suggested "installation directory" setting without instructions from SeeBeyond. You must have Administrator privileges to install this e*Way.*

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Exit all Windows programs and disable any anti-virus applications before running the setup program.
3. Insert the e*Way installation CD-ROM into the CD-ROM drive.
4. If the CD-ROM drive's **Autorun** feature is enabled, the setup application should launch automatically. If not, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the installation CD-ROM.
5. The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way.
6. After the e*Way has been installed, you need to install the SeeBeyond MUX Subscription Handler. See **Blue Martini Application Server** on page 21.

2.1.2 UNIX

Note: *You are not required to have root privileges to install this e*Way. Log on under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.*

1 Log onto the workstation containing the CD-ROM drive and, if necessary, mount the drive.

2 Insert the e*Way installation CD-ROM into the CD-ROM drive.

3 At the shell prompt, type

```
cd /cdrom
```

4 Start the installation script by typing:

```
setup.sh
```

5 A menu of options will appear. Select the **Install e*Way** option. Then, follow any additional on-screen directions.

Note: *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond. Note also that* ***no spaces*** *should appear in the installation path name.*

2.1.3 Installed Files

Note: *Directory paths are shown in this section using Windows conventions. Files and directories are identical in a UNIX installation.*

The Blue Martini e*Way installation process creates the following subdirectories and installs the following files within the **\eGate\client** tree on the Participating Host. These files are then committed to the “default” schema on the Registry Host.

Table 1 Installed Subdirectories and Files

Subdirectories	Files
	stcewbluemartini.ctl
\classes\	stchttp.jar stcutil.jar
client\classes\	stchttp.jar stcutil.jar
etd\httpclient\	httpclient.xsc
etd\bluemartini	blob.jar blob.ssc blob.xsc
configs\httpclient\	httpclient.def
client\pkicerts\client	certmap.txt

Table 1 Installed Subdirectories and Files

Subdirectories	Files
client\pkicerts\trustedcas	GTECyberTrustGlobalRoot.cer MircrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer trustcacerts.jks versisign_class3.cer
\ThirdParty\gnu-getopt\classes\	gnu-getopt.jar
\ThirdParty\jsse\jsse1.0.2\classes\	jcert.jar jnet.jar jsse.jar
\ThirdParty\xerces\classes\	xerces.jar

2.2 External Configuration Requirements

The following steps can proceed in any order as long as each step is completed.

1 Blue Martini Configuration

A Required components used for communicating with e*Gate using the e*Gate API Kit.

- **stcph.jar** : a standard e*Gate component used to support the Java multiplexer e*Way (See **"Blue Martini Application Server" on page 21** for more information)
- Egate_MUX.class and Egate_MUX$OutputType.class : a class that parses Blue Martini DNA objects into XML messages, which are sent to e*Gate as designated by the host name and port specified in the multiplexer e*Way.

B e*Gate Proxy used to receive request information from the HTTP e*Way.

- EgateProxy.class
- DebugRequest.class (optional for debugging)
- EgateProxy.jsp
- bizact.dna
- bizobj.dna
- dictionary.dna
- request.dna

2 Initial Installation:

A Assuming a functional and stable Blue Martini 4.0 installation, let <BMROOT> be the Blue Martini installation path. For example,

```
D:\bms\bmapps
```

B Stop the Blue Martini Application Server currently in use.

C Create a subdirectory called "Seebeyond" in each of the following directories:

- <BMROOT>\classes

  Copy the following class and jar files (EgateProxy.class, DebugRequest.class and stchph.jar) into the new Seebeyond directory.

- <BMROOT>\config

  Copy all of the "dna" configuration files (bizact.dna, bizobj.dna, dictionary.dna, and request.dna) into the new Seebeyond directory.

- <BMROOT>\docroot\templates

  Copy the Proxy JSP/Servlet file EgateProxy.jsp into the new Seebeyond directory.

D Create a sub-directory called "stc\bluemartini" in <BMROOT>\classes\com, and copy Egate_MUX.class and Egate_MUX&OutputType.class into the newly created directory.

E Append the path "classes\Seebeyond;classes\Seebeyond\stcph.jar" to the CLASSPATH variable in the Blue Martini environment batch file **setbmenv.bat**.

F For each Blue Martini Application Server (B2Bserver, Webconnect, etc.) to be utilitzed, edit the corresponding Blue Martini custom configuration module file (b2b.appconfig.dna for B2Bserver, webconnect.appconfig.dna for Webconnect server etc.) to set the "modules" attribute to include the Seebeyond subdirectory.

3 Configuration:

A e*Gate Proxy:

None, as long as the e*Gate Java HTTP e*Way is set up correctly.

B e*Gate Multiplexer:

I): Method 1 (Preferred) - via simple addition to existing application JSP.

JSP code Fragment sample: (Make sure that there is only one "page" directive for import for each JSP.)

```
<%@ page import="com.stc.bluemartini.*" %>
…
<%
String bizobj_filled = "";
BusinessObject bo = null;
if ( dnaFormData.contains("name_of_bizobj") )
{
   bo = dnaFormData.getBusinessObject("name_of_bizobj");
   Egate_MUX m = new Egate_MUX("egate_host", 26051, 5, 5000);
   m.SendToMUX(bo, "Name_of_business_object");
}
%>
```

Pros:

Simple.

No need to restart the Blue Martini Server (but depends upon Weblogic Servlet re-compilation settings).

Cons:

The existing application JSP might require rewriting to accommodate the addition.

II) Method 2 - via minimal Java code fragment addition.

```
…
package name_of_customization_directory;
…
import com.stc.bluemartini.*;
…
public class NameOfBusinessAction extends BusinessAction
{
public DNAList execute(DNAList dnaIn)
{
BusinessObject bo;
...
//manipulate bo
...
Egate_MUX m = new Egate_MUX("egate_host", 26051, 5, 5000);

m.SendToMUX(bo, "name_of_business_object");
}
}
```

Pros:

Fairly simple.

Cons:

May not be possible if no customization Java code is to be written or only predefined off-the-shelf Blue Martini business actions are used.

Requires restarting of the Blue Martini Server.

III) Method 3 - via defining a new business action (dna as post-action when the business object is ready to be sent), and Java code to extend the application Java package for the new business action.

Add a new outbound business action in the appropriate bizact.dna file.

```
DNA
{
...
“Use_Egate_MUX” DNA
{
“class”String”name_of_customization_dir.Use_Egate_MUX”
}
...
}
```

Add the outbound business action in the appropriate request.dna file.

```
DNA
{
…
"NameOfBusinessProcess" DNA
{
…
   "process" StringArray
```

```
        [
    "NameOfbusinessAction1",
    "NameOfbusinessAction2",
     …,
     "Use_Egate_MUX" // Typ. last biz action
     ]
    }
    …
    }
```

Provide Java code for implementing the new outbound business action:

```
    package name_of_customization_dir;

    import java.io.*;
    …
    import com.bluemartini.core.*;
    import com.bluemartini.dna.*;
    …
    import com.bluemartini.html.*;
    import com.bluemartini.htmlapp.*;
    …
    import com.stc.bluemartini.*;

    public class Use_Egate_MUX extends BusinessAction
    {
    …
    public DNAList execute ( DNAList dnaIn )
    {
    HTMLResponse htmlResponse = new HTMLResponse ( );

       try
       {
    Egate_MUX m = new Egate_MUX();
    m.set_hostname("egate_host");
    m.set_port(26051);
    m.set_expiration(10); // in seconds
    m.set_timeout(10000); // in milliseconds

       if ( m != null )
       m.SendToMUX(dnaIn, "OPERATION");
       else
    …
     }
       catch (Exception ex)
       {
    System.out.println ("got exception : " + ex.toString());
    }

    return htmlResponse;

}
…
}
```

Pros:

Always possible, even if no customization Java code is utilized, requires more customization than Method 2.

Cons:

Cannot be distributed as a simple Java class, because the Java code must be packaged with application specific naming conventions. A template Java sample must be provided.

Requires restarting of the Blue Martini Server.

2.2.1 System Configuration Requirements

The following Java Collaboration components are required by e*Gate Proxy to post requests to Blue Martini.

- http_collabrule.xpr
- http_collabrule.ctl
- http_collabrule.xts
- http_collabrule.java
- http_collabrule.class
- http_collabruleBase.class

In addition to the above components, the Blue Martini e*Way add-on must be installed successfully.

Configuring the System

It is assumed that a stable e*Gate schema is configured to include a Java HTTP e*Way.

1. Set the Default URL field of the configuration for the Java enabled HTTP e*Way Connection to:

```
http://<host>:<port>/Seebeyond/EgateProxy.jsp
```

   where:

   host: The host name/IP address of the Blue Martini machine.

   port: The port number for the specific Blue Martini Application Server.

2. At this time, the user must create DTDs or XML schemas to perform the translation prior to sending to e*Gate.

2.3 Installing the MUX Handler Classes

The SeeBeyond MUX Subscription Handler consists of ten Java classes, which are contained in the **stcph.jar** file. This file is available when the e*Gate participating host is installed. The class descriptions are given in **Appendix A**.

2.3.1 Blue Martini Application Server

1. Stop the Blue Martini Application.
2. Copy the **stcph.jar** file from the e*Gate Installation CD to the **Seebeyond** subdirectory located in **<BMROOT>\classes\Seebeyond**.
3. Add the path to the **stcph.jar** file in the **CLASSPATH** for the Blue Martini.
4. Start (boot) the Application Server for the appropriate domain.
5. Start the Blue Martini Application Server.

Chapter 3

e*Way Connection Configuration

This chapter describes how to create and configure Blue Martini (HTTP) e*Way Connections, using the e*Gate Schema Designer's Component Editor.

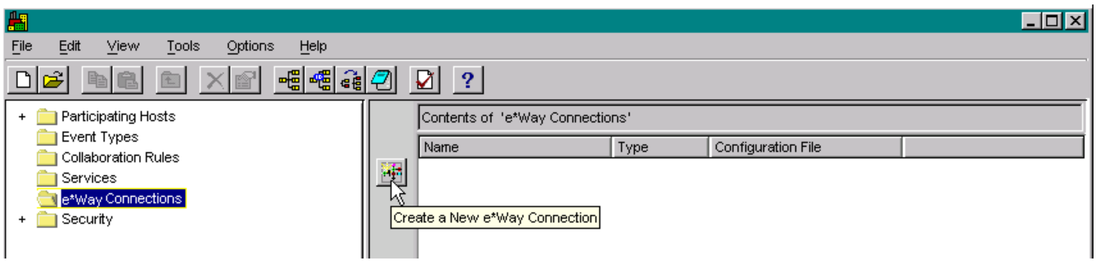
3.1 Creating e*Way Connections

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

To create and configure the e*Way Connections

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

Figure 6 Schema Designer - e*Way Connections Folder (1)



2. On the Palette, click the **Create a New e*Way Connection** button, which opens the **New e*Way Connection Component** dialog box.
3. Enter a name for the e*Way Connection and click **OK**. The new e*Way Connection will appear in the Schema Designer Contents pane.
4. Double-click the new e*Way Connection icon to open the **e*Way Connection Properties** dialog box.
5. From the **e*Way Connection Type** drop-down box, select **HTTP**.
6. Enter the **Event Type "get" interval** in the dialog box provided (optional).
7. Click **New** to invoke the e*Way Connection Configuration File Editor, where you can create a new e*Way Connection Configuration File.

3.2 Configuration Parameters

The HTTP/S e*Way Connection configuration parameters are organized into the following sections:

- connector
- HTTP
- Proxies
- HttpAuthentication
- SSL

3.2.1 Connector

This section contains a set of top level parameters:

- type
- class
- Property.Tag

Type

Description

Specifies the type of connection.

Required Values

Http. The value defaults to HTTP.

Class

Description

Specifies the class name of the HTTP Client connector object.

Required Values

A valid package name. The default is **com.stc.eways.http.HttpClientConnector**.

Property.Tag

Description

Specifies the data source identity. This parameter is required by the current EBobConnectorFactory.

Required Values

A valid data source package name.

3.2.2 HTTP

This section contains a set of top level parameters used by HTTP:

- DefaultUrl
- AllowCookies
- ContentType
- AcceptType

DefaultUrl

Description

Specifies the default URL to be used to post information to Blue Martini. If "https" protocol is specified, SSL must be configured. See the "SSL" section.

Required Values

A valid URL used for sending information to Blue Martini application Server.

Additional Information

You must include the full URL. For example,

http://<host>:<port>/Seebeyond/EgateProxy.jsp

http://www.seebeyond.com

or

http://google.yahoo.com/bin/query

If using GET functionality, you can provide the parameters, using the application/x-www-form-urlencoded notation. For example,

http://www.ee.cornell.edu/cgi-bin/cgiwrap/~wes/pq?FirstName=John&LastName=Doe

AllowCookies

Description

Specifies whether cookies sent from servers will be stored and sent on subsequent requests. If cookies are not allowed, sessions will not be supported.

Required Values

Yes or **No**.

ContentType

Description

Specifies the request content-type.

Required Values

A string. The default is set to "application/x-www-form-urlencoded". If sending other forms of data, set to the appropriate content-type. For example, "text/html".

Accept-type

Description

Specifies the parameters for the “Accept-type” request header.

Required Values

A string. For example “text/html”, “text/plain”, “text/xml” etc.

3.2.3 Proxies

The parameters in this section specify the information required for the e*Way Connection to access the external systems through a proxy server.

UseProxy

Description

Specifies whether an HTTP or HTTPS proxy will be used. If set to HTTP, then an HTTP Proxy for non-secured connection will be used. If HTTPS is selected, an HTTPS Proxy for secured connection will be used. Select NO if a Proxy is not used. See configuration parameters: HttpProxyHost, HttpProxyPort, HttpsProxyHost, HttpsProxyPort, UserName, and Password in this section.

Required Values

HTTP ,**HTTPS**, or **NO**.

HttpProxyHost

Description

Specifies the HTTP proxy host name to which to delegate requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy host for non-secured HTTP connections. To turn on proxy use, see the UseProxy configuration parameter.

Required Values

A HTTP proxy host name.

HttpProxyPort

Description

Specifies the HTTP proxy port to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy port fro non-secured HTTP connections. To turn on proxy use, see the UseProxy configuration parameter.

Required Values

A valid HTTP proxy port name.

HttpsProxyHost

Description

Specifies the HTTPS proxy host to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy port for secured HTTP connections. To turn on proxy use, see the UseProxy configuration parameter.

Required Values

A valid HTTPS proxy host name.

HttpsProxyPort

Description

Specifies the HTTPS proxy port to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy port for secured HTTP connections. To turn on proxy use, see the UseProxy configuration parameter.

Required Values

A valid HTTPS proxy port name.

User Name

Description

Specifies the user name necessary for authentication to access the proxy server. To turn on proxy use, see the UseProxy configuration parameter.

Required Values

A valid user name.

Additional Information

The username is required by URLs that require “HTTP Basic Authentication” to access the site.

Important: *Enter a value for this parameter* ***before*** *you enter a value for the* ***Password*** *parameter.*

PassWord

Description

Specifies the password corresponding to the username specified previously.

Required Values

The appropriate password.

Additional Information

The username is required by URLs that require "HTTP Basic Authentication" to access the site.

Important: *Be sure to enter a value for the **User Name** parameter before entering the **Password**.*

3.2.4 HttpAuthentication

The parameters in this section are used to perform HTTP authentication.

UseHttpAuthentication

Description

Specifies whether standard HTTP Authentication will be used. This is used when the web site requires username and password authentication. If this is selected, the UserName and Password configuration parameters must be set. See UserName and PassWord configuration parameters in this section.

Required Values

Yes or **No**.

UserName

Description

Specifies the user name for standard HTTP Authentication. See UseHttpAuthentication configuration parameter.

Required Values

A valid user name.

Important: *Enter a value for this parameter **before** you enter a value for the **Password** parameter.*

PassWord

Description

Specifies the password associated with the specified user name for standard HTTP Authentication. See UseHttpAuthentication configuration parameter.

Required Values

A valid password.

Important: *Be sure to enter a value for the **User Name** parameter before entering the **Password**.*

3.2.5 SSL

The parameters in this section control the information required to set up an SSL connection via HTTP.

UseSSL

Description

Specifies whether SSL needs to be configured in order to use the “https” protocol. If set to YES, then at least HttpsProtocolImpl and Provider must be given.

Required Values

Yes or **No**.

HttpsProtocolImpl

Description

Specifies the package that contains the HTTPS protocol implementation. This will add the “https” URLStreamHandler implementation by including the handler’s implementation package name to the list of packages which are searched by the Java URL class. The default value specified is the package which contains the SUN reference implementation of the “https” URLStreamHandler.

Required Values

A valid package name. The default is com.sun.net.ssl.internal.www.protocol. This parameter is mandatory if using HTTPS.

Provider

Description

Specifies the Cryptographic Service Provider. This will add a JSSE provider implementation to the list of provider implementations. The default value specified is the SUN reference implementation of the Cryptographic Service Provider, “SunJSSE”.

Required Values

A valid provider name. The default is com.sun.net.ssl.internal.ssl.Provider. This parameter is mandatory if using HTTPS.

X509CertificateImpl

Description

Specifies the implementation class of the X509Certificate.

Required Values

A valid package location. For example, if the implementation class is called, “MyX509CertificateImpl”, and it resides in the com.radcrypto package, you would specify com.radcrypto.MyX509CertificateImpl.

SSLSocketFactoryImpl

Description

Specifies the implementation class of the SSL Socket Factory.

Required Values

A valid package location. For example, if the implementation class is called MySSLSocketFactoryImpl and it resides in the com.radcrypto package, you would specify com.radcrypto.MySSLSocketFactoryImpl.

SSLServerSocketFactoryImpl

Description

Specifies the implementation class of the SSL Server Socket Factory.

Required Values

A valid package location. For example, if the implementation class is called MySSLServerSocketFactoryImpl and it resides in com.radcrypto package, you would specify com.radcrypto.MySSLServerSocketFactoryImpl.

KeyStore

Description

Specifies the default key store file for use by the KeyManager. If the default key store is not specified with this method, the key store managed by KeyManager is empty.

Required Values

A valid package location.

KeyStoreType

Description

Specifies the default key store type. If the default key store type is not set by this method, the default key store type, “jks” is used.

KeyStorePassword

Description

Specifies the default key store password. If the default key store password is not set by this method, the default key store password is assumed to be “ “.

TrustStore

Description

Specifies the default trust store. If the default trust store is not set here, then a default trust store search is performed. If a trust store named <java-home>/lib/security/jssecacerts is found, it is used. If not, a search for a trust store name <java-home>/lib/security/cacerts is made, and used if located. If a trust store is not found, the trust store managed by the TrustManager will be a new empty trust store.

Required Values

A valid trust store name.

TrustStore Password

Description

Specifies the default trust store password. If the default trust store password is not set by this method, the default trust store password is “ “.

KeyManager Algorithm

Description

Specifies the default key manager algorithm name to use. For example, the default key manager algorithm used in the SUN reference implementation of JSSE is “SunX509”.

Required Values

A valid key manager algorithm name.

TrustManagerAlgorithm

Description

Specifies the default trust manager algorithm name to use. For example, the default trust manager algorithm used in the SUN reference implementation of JSSE is “SunX509”.

Required Values

A valid trust manager algorithm name.

Chapter 4

e*Way Setup

This chapter summarizes the initial setup procedures for the Java-enabled Blue Martini e*Way.

4.1 Overview

After installing the Blue Martini e*Way, you must perform an initial setup for it to work with your system. A wide range of setup options allow the e*Way to conform to your system's operational characteristics and your facility's operating procedures.

The topics discussed in this chapter include the following:

4.2 Setting Up the e*Way

4.2.1 Defining e*Way Components

The first step in implementing an e*Way is to define the e*Way component using the e*Gate Schema Designer.

To create an e*Way

1. Select the e*Gate Schema Designer Navigator's **Components** tab.
2. Open the host on which you want to create the e*Way.
3. Select the Control Broker that will manage the new e*Way.
4. On the Palette, click **Create a New e*Way**.
5. Enter the name of the new e*Way, then click **OK**.

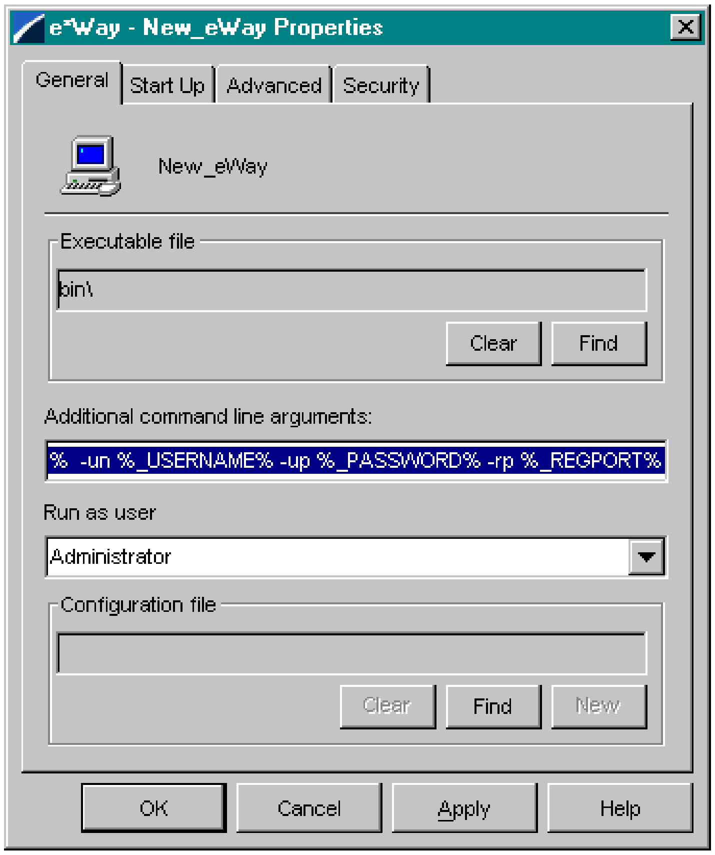
4.2.2 Modifying e*Way Properties

Note: *Selecting the executable file should be the first configuration procedure you perform once you have created the e*Way.*

To modify any e*Way properties

1. Select the Navigator's **Components** tab.
2. Open the host on which the desired e*Way runs.
3. Open the Control Broker that manages the e*Way.
4. Select the desired e*Way.
5. Right-click on the e*Way and select **Properties** to edit the e*Way's properties. The properties dialog will open to the **General** tab (shown in Figure 7).

Figure 7 e*Way Properties (General Tab)



Executable file: bin\

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

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

Note: *When you shut down an e*Way and open its property sheet in the e*Gate Schema Designer, once you click* **OK** *or* **Apply**, *the e*Way immediately restarts. This action only happens if the e*Way is in autostart mode. After you click* **OK** *or* **Apply**, *the Registry is automatically updated with any changes, if you made them using the e*Way Editor.*

4.2.3 Selecting an Executable File

Selecting the executable file is the first and most important step in configuring the e*Way. This step determines what type of e*Way will run and thus what type of external system or communications protocol it will support.

You must know which executable file to select before you perform this procedure.

To select an e*Way's executable file

1 Display the e*Way's properties (see the **procedure on page 32**).

2 On the General tab, under **Executable File**, click **Find**.

3 Use the file selection dialog box to select the executable files. All e*Way executable files have a **.exe** extension.

Note: *You must use the* **Find** *button to select the executable file. You cannot type its name directly into the Executable File box.*

4.2.4 Selecting or Creating a Configuration File

After you have selected an executable file, you must select or create a configuration file that will contain the operating parameters for the e*Way.

To select an existing configuration file

1. Display the e*Way's properties (see the **procedure on page 32**).
2. On the **General** tab, under **Configuration File**, click **Find**.
3. Use the file selection dialog box to select the desired file (***.cfg**).
4. **Exit** the e*Way Editor.

Note: *You must use the* **Find** *button to select the configuration file. You cannot type its name directly into the Configuration File box.*

To create a configuration file

1. Display the e*Way's properties (see the **procedure on page 32**).
2. On the **General** tab, under **Configuration File**, click **Find**.
3. Use the file selection dialog box to select a default configuration (template) file.

Note: *All e*Way default configuration files have a* **.def** *extension, and are intended to be used as templates.*

4. Use the e*Way Editor to change the default configuration parameters as required .
5. Edit the **Additional Command Line Arguments** box to include any arguments you require (see the following procedure).
6. **Save** the file with a **.cfg** extension and **exit** the e*Way Editor.

Note: *You must use the* **Find** *button to select the configuration file. You cannot type its name directly into the Configuration File box.*

4.2.5 Changing Command-line Parameters

Most SeeBeyond e*Ways require only the default command-line parameters shipped with the e*Gate Schema Designer. Use the procedure in this section only if the e*Way you are configuring requires special command-line options, or if you are directed to do so by SeeBeyond support personnel.

To change an e*Way's command-line options

1. Display the e*Way's properties (see the **procedure on page 32**).
2. On the **General** tab, edit the **Additional Command Line Arguments** box to include the arguments you require. Unless you have a specific need to do so, do not change

any of the existing parameters; add any new parameters to the end of the command line.

4.2.6 Changing the User Name

Like all e*Gate executable components, e*Ways run under an e*Gate user name. By default, all e*Ways run under the **Administrator** user name. You can change this if your site's security procedures so require.

To change the "run as" user name

1. Display the e*Way's properties (see the **procedure on page 32**).
2. On the **General** tab, use the Run As list to select the e*Gate user under whose name this component will run.

See the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate security system.

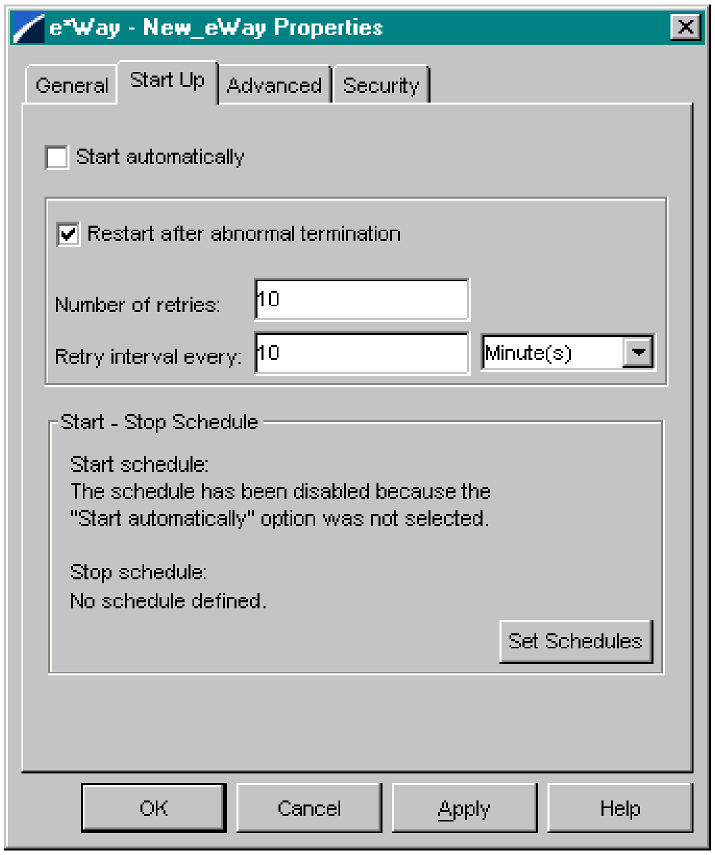
4.2.7 Setting Startup Options or Schedules

SeeBeyond e*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the e*Way automatically whenever the Control Broker starts.
- The Control Broker can start the e*Way automatically whenever it detects that the e*Way terminated execution abnormally.
- The Control Broker can start or stop the e*Way on a schedule that you specify.
- Users can start or stop the e*Way manually using an interactive monitor.

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 8). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 8 e*Way Properties (Start-Up Tab)



☐ Start automatically

☑ Restart after abnormal termination

Number of retries: 10

Retry interval every: 10 Minute(s)

Start - Stop Schedule

Start schedule:
The schedule has been disabled because the "Start automatically" option was not selected.

Stop schedule:
No schedule defined.

To determine whether the e*Way starts automatically when the Control Broker starts

1. Display the e*Way's properties (see the **procedure on page 32**).
2. Select the **Start Up** tab.
3. To activate this feature, check **Start automatically**.
4. To deactivate this feature, clear the **Start automatically** check box.
5. Click **OK**.

To determine whether the e*Way will be restarted automatically

1. Display the e*Way's properties (see the **procedure on page 32**).
2. Select the **Start Up** tab.
3. To activate this feature, check **Restart after abnormal termination** and set the desired number of retries and retry interval.
4. To deactivate this feature, clear the **Restart** check box.
5. Click **OK**.

Note: *The "auto restart" feature will not automatically restart the e*Way if the e*Way is shut down manually by an interactive monitor.*

*If the e*Way is shut down and you make any configuration changes using the e*Gate Schema Designer, the Control Broker will automatically restart the e*Way when the configuration changes are recorded in the e*Gate Registry. If you do not want the e*Way to restart when configuration changes are made, disable this feature before configuring the e*Way.*

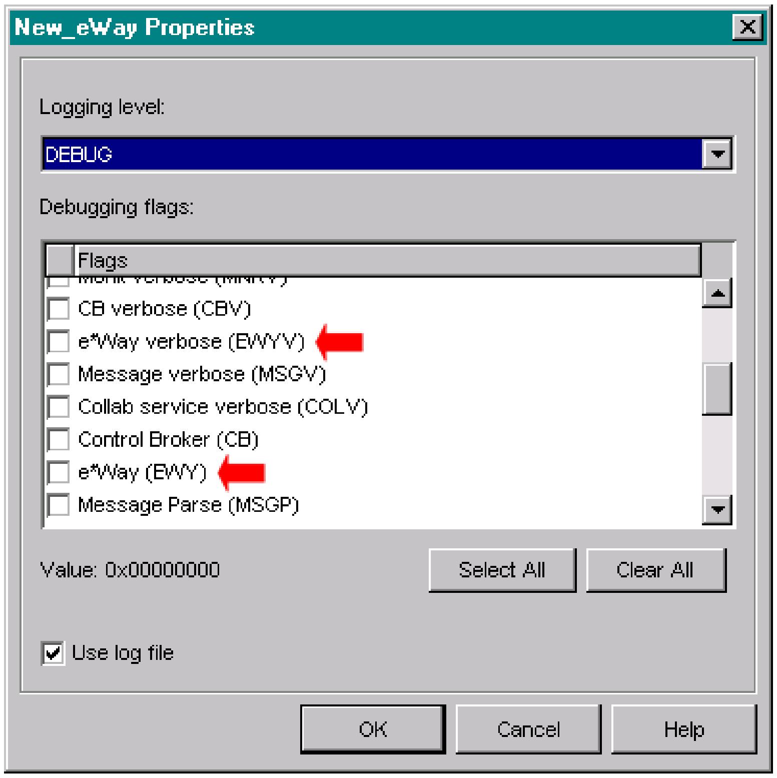
4.2.8 Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the e*Way and other e*Gate components.

To set the e*Way debug level and flag

1. Select the Navigator's **Components** tab.
2. Open the host on which the desired e*Way runs.
3. Open the Control Broker that manages the e*Way.
4. Select the desired e*Way.
5. On the Toolbar, click **Properties** to edit the e*Way's properties.
6. Select the **Advanced** tab.
7. Click **Log**. The dialog window will appear as in Figure 9.
8. Select **DEBUG** for the **Logging level**.
9. Select either **e*Way (EWY)** or **e*Way Verbose (EWYV)** for the **Debugging flag.** Note that the latter will have a significant impact on system performance.
10. Click **OK**.

Figure 9 e*Way Properties (Advanced Tab - Log Option)



The other options apply to other e*Gate components and are activated in the same manner. See the *e*Gate Integrator Alert and Log File Reference* for additional information concerning log files, logging options, logging levels, and debug flags.

4.2.9 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way will send a Monitoring Event to the Control Broker, which will be routed to the e*Gate Monitor or any number of destinations.

1. Display the e*Way's properties (see the **procedure on page 32**).
2. Select the **Advanced** tab.
3. Click **Thresholds**.
4. Select the desired threshold options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference* for more information concerning threshold monitoring, routing specific notifications to specific recipients, or for general information about e*Gate's monitoring and notification system.

4.3 Troubleshooting the e*Way

Because of the flexibility provided for customization of SeeBeyond e*Ways, it is impossible to provide a comprehensive guide to troubleshooting. However, this section provides guidelines to follow when troubleshooting of the e*Way's operation or performance. In the initial stages of developing your e*Gate Integrator system administration system, most problems with e*Ways can be traced to configuration.

4.3.1 In the Schema Designer

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rules script employed by this e*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that "feed" this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way "feeds" properly configured, and are they subscribing to the appropriate Events correctly?

4.3.2 In the e*Way Editor

- Check that all configuration options are set appropriately.
- Check that all settings you changed are set correctly.
- Check all required changes to ensure they have not been overlooked.
- Check the defaults to ensure they are acceptable for your installation.

4.3.3 On the e*Way's Participating Host

- Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e*Way's Collaborations publish.

4.3.4 In the e*Way's External Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.
- Check that the connection between the external application and the e*Way is functioning appropriately.
- Once the e*Way is up and running properly, operational problems can be due to:
  - External influences (network or other connectivity problems).
  - Problems in the operating environment (low disk space or system errors)
  - Problems or changes in the data the e*Way is processing.
  - Corrections required to Collaboration Rules scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e*Way log file. See the *e*Gate Integrator Alert and Log File Reference Guide* for an extensive explanation of log files, debugging options, and using the e*Gate monitoring system to monitor operations and performance.

Chapter 5

System Implementation

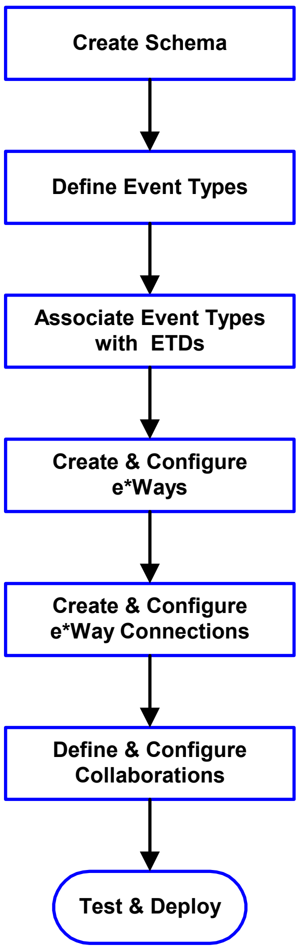
In this chapter we summarize the procedures required for implementing a working system incorporating the Java-enabled Blue Martini e*Way. Please refer to the *e*Gate Integrator User's Guide*.

5.1 Overview

This e*Way provides a specialized transport component for incorporation into an operational Schema. The schema also will contain Collaborations, linking different data or Event types, and Intelligent Queues. Typically, other e*Way types also will be used as components of the Schema.

Topics included in this chapter include:

5.1.1 Implementation Sequence


Create Schema
Define Event Types
Associate Event Types with ETDs
Create & Configure e*Ways
Create & Configure e*Way Connections
Define & Configure Collaborations
Test & Deploy

1 The first step is to create a new Schema—the subsequent steps will apply to this Schema (see **Creating a Schema** on page 44).

2 The second step is to define the Event Types you will be transporting and processing within the Schema (see **Creating Event Types** on page 44).

3 Next you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see **Generating Event Type Definitions** on page 45).

4 The fourth step is to create and configure the required e*Ways.

5 The fifth step is to configure the e*Way Connections.

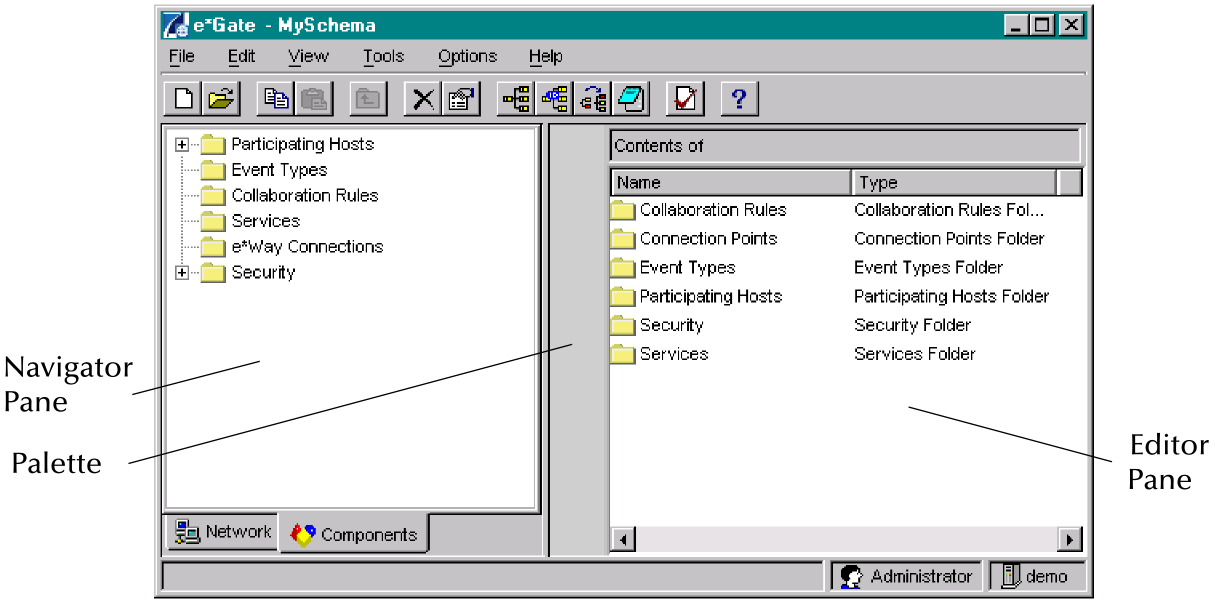
6 Next you need to define and configure the Collaborations between Event Types (see **Defining Collaborations** on page 49).

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

5.2 Using the e*Gate Schema Designer

First, here is a brief introduction to the e*Gate Schema Designer. The general features of the e*Gate Schema Designer window are shown in Figure 10.

Figure 10 e*Gate Schema Designer Window (Components View)

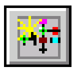


Contents of

Name	Type
Collaboration Rules	Collaboration Rules Fol...
Connection Points	Connection Points Folder
Event Types	Event Types Folder
Participating Hosts	Participating Hosts Folder
Security	Security Folder
Services	Services Folder

Use the Navigator and Editor panes to view the e*Gate components. Note that you may only view components of a single schema at one time, and that all operations apply only to the current schema. Specialized command buttons (see Figure 11) appear in the Palette area of the window, depending on which levels of the Components Tree are open. For additional information, see the *e*Gate Integrator User's Guide*.

Figure 11 Setup Command Buttons

Button	Name	Location	Function
	Create e*Way	Palette	Creates a new e*Way
	Create IQ	Palette	Creates a new Intelligent Queue
	Create BOB	Palette	Creates a new Business Object Broker
	**Create e*Insight Engine**	Palette	Creates a new e*Insight Engine (only if e*Insight BPM is installed)

5.3 Creating a Schema

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

Because all setup and configuration operations take place within an e*Gate schema, a new schema must be created, or an existing one must be started before using the system. Schemas store all their configuration parameters in the e*Gate Registry.

To create a new schema

1. Launch the e*Gate Schema Designer and log in as **Administrator** (or other user with equivalent privilege) on the appropriate Registry Host.
2. When the **Open Schema on Registry Host** dialog box appears, click **New**.
3. In the **Enter New Schema Name** text box, enter a name for the new schema (e.g., **TestSchema**), then click **Open**.

The Schema Designer will open the new schema.

4. From the **Options** menu, click on **Default Editor** and select **Java**.
5. Select the **Components** tab, found at the bottom of the Navigator pane of the e*Gate Schema Designer window (see **Figure 10 on page 43**).

The e*Gate Schema Designer then opens under your new schema name. You are now ready to begin creating the necessary components for this new schema.

Note: *From this point forward, all procedures should be performed while displaying the Components Navigator pane.*

5.4 Creating Event Types

Within e*Gate, messages and/or packages of data are defined as Events. Each Event must be categorized into a specific Event Type within the schema.

To define the Event Types

1. In the e*Gate Schema Designer's Navigator pane, select the **Event Types** folder.
2. On the Palette, click the **New Event Type** button.
3. In the **New Event Type Component** box, enter the name for the input Event Type and click **Apply**. Use this method to create all required Event Types, for example:
   - **blob (Generic Event)**
   - **HttpClient**
4. After you have created the final Event Type, click **OK**.

5.5 Generating Event Type Definitions

As the name implies, an Event Type Definition (ETD) defines the structure of the Event Types employed in your Schema. Any one ETD can be associated with more than one Event Type within the schema. In the Blue Martini e*Way, ETDs are created semi-automatically using the DTD Builder, once a DTD has been generated.

As of release 4.0, Blue Martini does not offer a DTD generation utility, nor does it include any sample DTDs.

Note: *The procedures described may not work for all DNA/BizObj definitions, and the user needs to know the data constraints for a particular DNA/BizObj definitions in order to be able to populate the message with sample data. The user should also be familiar with XML messaging and working with DTDs.*

Several steps are required to be completed for Blue Martini and e*Gate to communicate:

Currently, it is the user's responsibility to create the necessary DTDs.

Generating an ETD from the DTD on page 45

Note: *The user should be familiar with using the Blue Martini Architecture understanding and using the various components.*

5.5.1 Generating an ETD from the DTD

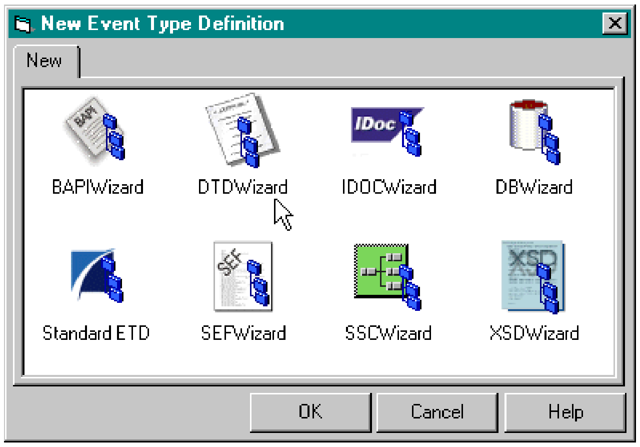
Use the SeeBeyond DTD Builder to produce an Event Type Definition for the DTD that you have generated. See the *XML Toolkit User's Guide* for detailed information on the DTD Builder.

Note: *Currently it is the responsibility of the user/administrator to create the DTD file from the DNA file.*

Using the DTD Builder

The ETD Editor contains a DTD Builder, which takes an XML DTD and converts it to an **.xsc** file. To access the DTD Builder's front-end Wizard, select the **New** option in the ETD Editor's **File** menu. The New Event Type Definitions window will appear, displaying all installed ETD Wizards (see Figure 12).

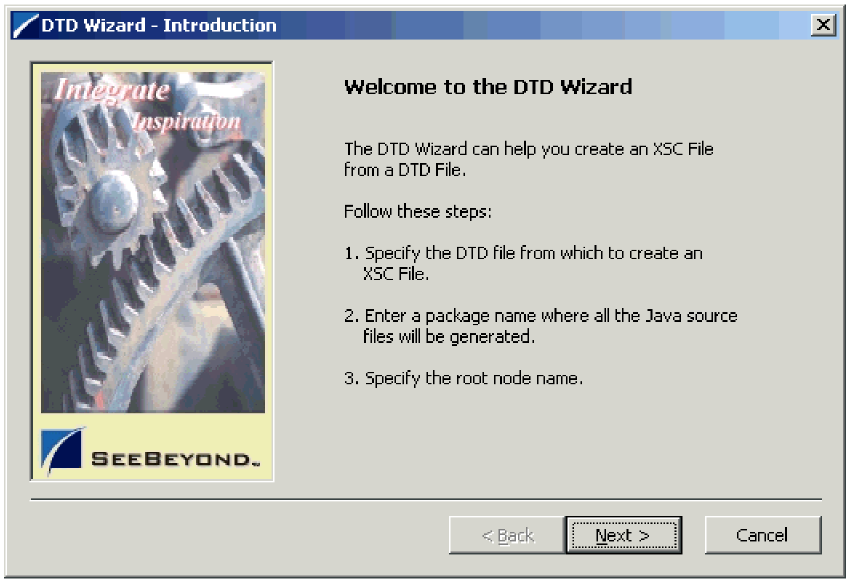
Figure 12 New Event Type Definitions Window



To run the DTD Builder

1 Invoke the DTD Wizard by clicking its icon.

Figure 13 DTD Wizard — Introduction



Welcome to the DTD Wizard

The DTD Wizard can help you create an XSC File from a DTD File.

Follow these steps:

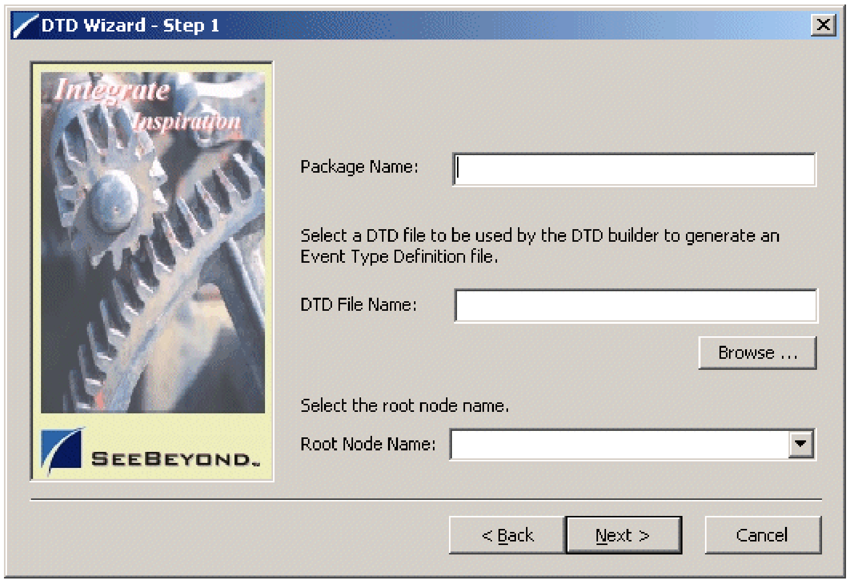
1. Specify the DTD file from which to create an XSC File.

2. Enter a package name where all the Java source files will be generated.

3. Specify the root node name.

2 Read the instructions carefully, and click **Next**. Step 1 of the DTD Wizard dialog appears (see Figure 14).

Figure 14 DTD Wizard — Step 1



Select a DTD file to be used by the DTD builder to generate an Event Type Definition file.

Select the root node name.

3 Enter the following information:

- **Java Package Name**

  Type in the name you want to give the Java package, for example, **com.tes**. This name must conform to Java package name requirements. See the appropriate Java documentation for details.

- **DTD File Name**

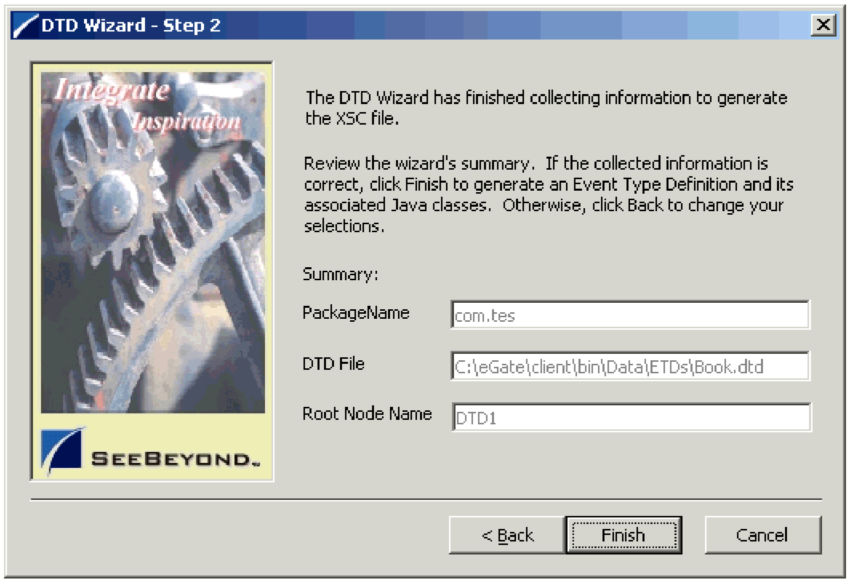
  Type in the name of the DTD file you want to convert. Click **Browse** to access an Open (file selection) dialog box, allowing you to choose the desired file.

- **Root Node Name**

  This text box is a pull-down menu. Select the desired root node name from the menu. For more information on root nodes and ETDs, see the *e*Gate Integrator User's Guide*.

4 When you are finished, click **Next**. Step 2 of the DTD Wizard dialog appears (see Figure 15).

Figure 15 DTD Wizard — Step 2



The DTD Wizard has finished collecting information to generate the XSC file.

Review the wizard's summary. If the collected information is correct, click Finish to generate an Event Type Definition and its associated Java classes. Otherwise, click Back to change your selections.

Summary:

PackageName	com.tes
DTD File	C:\eGate\client\bin\Data\ETDs\Book.dtd
Root Node Name	DTD1

5 Review the information you have entered in the Wizard. If it is correct, click **Finish** to generate a Java ETD (**.xsc** file) from the original DTD file.

The Wizard closes, and the new ETD appears in the ETD Editor Main window. See the *e*Gate Integrator User's Guide* for details on how to use this editor, including an explanation of the information it shows.

6 To save the new ETD, click the **Save** button on the Toolbar or select the **Save** command from the **File** menu. A Save dialog box appears.

7 Select the desired directory location, give the new ETD your desired name, and click **Save**. The ETD Editor saves the new Java ETD.

8 You can continue to use the ETD Editor or select the **Close** command from the **File** menu to exit the GUI.

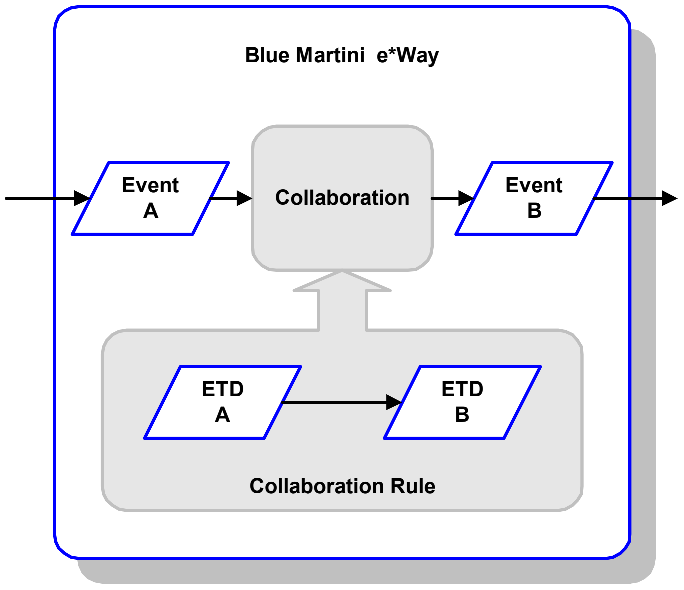
Note: *The ETD nodes created using the DTD Builder appear shaded in the ETD Editor, indicating that you cannot edit an ETD created by the Builder.*

After converting the DTD to an ETD, return to the e*Gate Schema Designer to verify the process .

5.6 Defining Collaborations

After you have created the required Event Type Definitions, you must define a Collaboration to transform the incoming Event into the desired outgoing Event. The Collaboration is driven by a Collaboration Rules script, which defines the relationship between the incoming and outgoing ETDs.

Figure 16 Collaborations



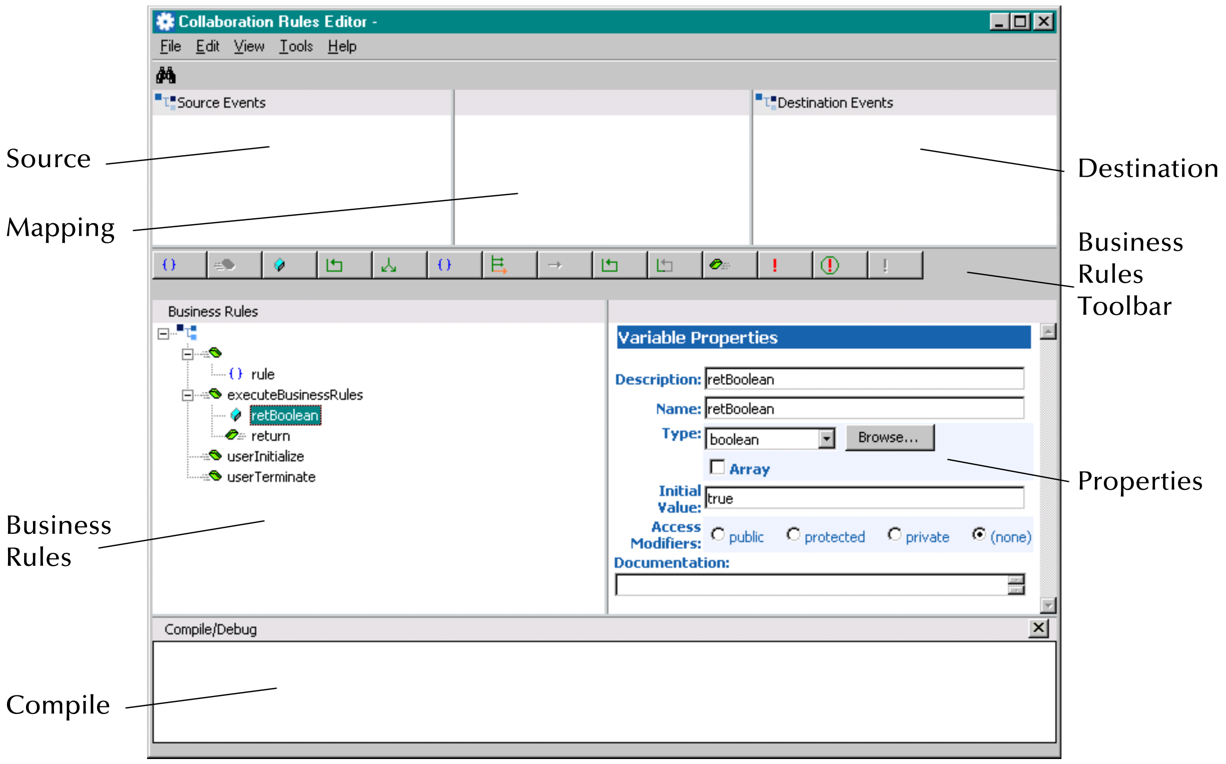
5.6.1 The Java Collaboration Rules Editor

Java Collaborations are defined using the e*Gate Java Collaboration Rules Editor (JCE). Note that unlike Monk, the Java Collaboration environment supports multiple source and destination ETDs. The file extension for Java Collaboration Rules is **.xsc**.

JCE GUI Overview

In addition to the main menu and the accompanying tool bar, the JCE window is organized into six panes. Running horizontally through center of the window is a **Business Rules Tool Bar** that contains buttons used to insert common programming constructs into the Collaboration. Figure 17 shows the names and locations of the different areas that make up the JCE GUI.

Figure 17 Java Collaboration Rules Editor Window



rule
executeBusinessRules
retBoolean
return
userInitialize
userTerminate

Variable Properties

Description: retBoolean
Name: retBoolean
Type: boolean
Array
Initial Value: true
Access Modifiers: public / protected / private / [✓ (none)]
Documentation:

The JCE Window Panes

Source

This pane contains the ETDs that correspond to the Event Types that are subscribed to by the e*Gate Collaboration Rule.

Mapping

This pane depicts which Source ETDs are mapped to which Destination ETDs.

Destination

This pane contains the ETDs that correspond to the Event Types that are published by the e*Gate Collaboration Rule.

Business Rules

This pane provides a graphical display of the business rules that form the basis for the Collaboration. Selecting the **Display Code** option from the Editor's **View** menu displays the Java source code that creates the output Events. Much of this code is created automatically by the JCE. The **executeBusinessRules** method is a placeholder for any additional programming you may need to provide.

Properties

This pane displays information about the item selected in the **Business Rules** pane. Here you can edit the selected Collaboration Rule.

Compile

After making changes, you must compile the Java Collaboration. Any errors that occur as a result of the compilation process are displayed here. Use the **Display Output** option in the Editor's **View** menu to display or hide this pane.

5.6.2 Using the JCE

Opening the Editor

You can open the Java Collaboration Rules Editor using any of the following methods:

- Select **Java** as the Schema Designer's default editor (in the **Options** menu) and
  - click the **Collaboration Editor** button, or
  - select the **Collab Editor...** option from the **Tools** menu
- Double-click on a file having an **.xsc** extension, irrespective of the default editor setting

Editing an Existing Collaboration

1. In the **File** menu, click the **Open** command.
2. Locate and select the desired Collaboration using the **Open** dialog box.
3. Edit the Collaboration as needed (see the *e*Gate Integrator User's Guide* for details).
4. In the **File** menu, click the **Compile** command.
5. Check the **Compile/Debug** pane for compilation errors, and modify your edits as necessary.
6. Repeat steps 4-5 until no errors result from the compilation.
7. In the **File** menu, click the **Save** command.
8. Exit the JCE.

Creating a New Collaboration

1. In the **File** menu, click the **Open** command.
2. Locate and select the desired Source ETDs using the **Open** dialog box.
3. Locate and select the desired Destination ETDs using the **Open** dialog box.
4. Drag and drop the ETDs to the correct locations, if necessary.
5. Develop the Collaboration (see the *e*Gate Integrator User's Guide* for details).
6. In the **File** menu, click the **Compile** command.
7. Check the **Compile/Debug** pane for compilation errors, and modify your edits as necessary.
8. Repeat steps 6-7 until no errors result from the compilation.
9. In the **File** menu, click the **Save** command.
10. Exit the JCE.

5.7 Using the e*Way

As explained in the Introduction, full integration with Blue Martini requires the use of the Java-enabled conjunction HTTP e*Way, the e*Gate API Kit, and SeeBeyond's customized Blue Martini MUX subscription handler classes. The Blue Martini HTTP e*Way is used to publish data to Blue Martini by utilizing the HTTP e*Way. The API Kit (Multiplexer e*Way) is used in conjunction with the subscription handler classes written by SeeBeyond in order to receive data from Blue Martini and publish the data to e*Gate.

5.7.1 Sample Schemas

Sample schemas are included on the e*Gate Installation CD. The sample “calculator” is located in the following directory:

```
\\samples\ewbluemartini
```

e*Gate to Blue Martini

The Egate2BM.zip (e*Gate schema files) sends HTTP post data as a blob from e*Gate to Blue Martini.

To Install and Import the e*Gate to Blue Martini Sample

1. Install the Blue Martini e*Way add-on.
2. Start the e*Gate Schema Designer GUI.
3. When the Schema Designer prompts you to log in, select the host that you specified during installation, and enter your password.
4. You are then prompted to select a schema. Click New. The New Schema dialog box opens. (Schemas can also be imported or opened from the e*Gate File menu by selecting New Schema or Open Schema.)
5. Type the name for the new schema, bluemartini_http_post.
6. To import the sample schema select Create from Export, and click Find to locate and select the Egate2BM.zip file on the CD-ROM.
7. The e*Gate Schema Designer opens to the new schema. You are now ready to make any configuration changes that may be necessary for this sample schema to run on your specific system.
8. Register a control broker for this new schema:

```
stccb.exe -ln localhost_cb -rh localhost -rs bluemartini_http_post -
un Administrator -up STC
```

   Or to use W2k service:

```
stccb.exe -ln localhost_cb -rh localhost -rs bluemartini_http_post -
un Administrator -up STC -sm
```

9. Start the newly registered Control Broker. From the control panel, double-click **Services**, right click on **e*Gate Control Broker (bluemartini_requestreply)**, click **Start**.

10 The sample data file (XML) sampledata.fin should be copied to:

```
e.g.: eGate\client\DATA\Blue_Martini
```

Blue Martini to e*Gate

The BM2Egate.zip (e*Gate schema files) receives data (XML via the multiplexer) from Blue Martini into e*Gate.

To Install and Import the Blue Martini to e*Gate Sample

1 Install the Blue Martini e*Way add-on.

2 Start the e*Gate Schema Designer GUI.

3 When the Schema Designer prompts you to log in, select the host that you specified during installation, and enter your password.

4 You are then prompted to select a schema. Click New. The New Schema dialog box opens. (Schemas can also be imported or opened from the e*Gate File menu by selecting New Schema or Open Schema.)

5 Type the name for the new schema, bluemartini_requestreply.

6 To import the sample schema select Create from Export, and click Find to locate and select the BM2Egate.zip file on the CD-ROM.

7 The e*Gate Schema Designer opens to the new schema. You are now ready to make any configuration changes that may be necessary for this sample schema to run on your specific system.

8 Register a control broker for this new schema:

```
stccb.exe -ln localhost_cb -rh localhost -rs bluemartini_requestreply
-un Administrator -up STC
```

Or to use W2k service:

```
stccb.exe -ln localhost_cb -rh localhost -rs bluemartini_requestreply
-un Administrator -up STC -sm
```

9 Start the newly registered Control Broker. From the control panel, double-click **Services**, right click on **e*Gate Control Broker (bluemartini_requestreply)**, click **Start**.

BMProxy.zip

See **"External System Requirements" on page 13** for installation information for BMProxy.zip.

Blue Martini Sample Calculator

BMsample.zip provides a simple Blue Martini customization sample to provide sample data that makes use of the Egate_MUX component (Egate Java API Kit) to send the data to e*Gate.

1 Assuming a functional and stable Blue Martini 4.0 installation, let <BMROOT> be the Blue Martini installation path (e.g. d:\bms\bmapps).

2. Stop the Blue Martini Application Server (e.g. B2Bserver, Webconnect, etc.)
3. Create a subdirectory called “calculator” in each of the following directories

```
<BMROOT>\classess
```

Copy all the Java class files (DebugRequest.class and PerformOperation.class) into the new "calculator" directory.

```
<BMROOT>\config
```

Copy all the “dna” configuration files (bizact.dna, bizobj.dna, dictionary.dna, domains_en.dna and request.dna) into the new "calculator" directory.

```
<BMROOT>\docroot\templates
```

Copy the Proxy JSP/Servlet file EgateProxy.jsp into the new "calculator" directory.

4. Append the path “classes\calculator” to the CLASSPATH variable in Blue Martini environment batch file setbmenv.bat.
5. For each Blue Martini Application Server (e.g. B2Bserver, Webconnect, etc.) to be utilized, edit the corresponding Blue Martini custom configuration module file (e.g. b2b.appconfig.dna for B2Bserver, webconnect.appconfig.dna for Webconnect server, etc.) to set the “modules” attribute to include the newly created "calculator" subdirectory.
6. Restart the Blue Martini Application Server.
7. Use HTTP browser to access the JSP page, e.g. (in an intranet environment)

```
http://name_of_BM_appserver:port/calculator/calculator.jsp
```

or

```
http://name_of_BM_appserver:port/calculator/MUX_calculator.jsp
```

(Watch for data in e*Gate log files -- with e*Way verbose trace settings turned on.

Appendix A

A.1 MUX Subscription Handler Java Classes

The MUX subscription handler consists of ten Java classes, which are contained in the **stcph.jar** file. This file is available when the e*Gate participating host is installed.

Note: *The* **MuxPublicationHandler** *class depends on the MUX Java Client classes, which also are contained in the* **stcph.jar** *file.*

Entry.class

Description

Holds the following information for an instance of a MUX subscription handler:

- Node Name
- MUX Host
- MUX Port
- MUX Seconds To Expire
- MUX Milliseconds Timeout
- Inflate flag
- Base64-Decode flag
- Log File.

MuxHandlerConstants.class

Description

Contains constant values for the MUX handler package such as the name of the configuration file to store the parameters persistently.

MuxHandlerEntry.class

Description

Maintains a collection of Entry classes. This class Loads and parses the configuration file in order to load the data into memory. It also saves changes to the values for the Entry classes to the configuration file.

AdministerMuxHandler.class

Description

Displays the MUX administration root page for administering the MUX subscription handler.

AdministerMuxHandlerAddMode.class

Description

Displays the MUX administration page for adding a MUX subscription handler.

AdministerMuxHandlerDeleteMode.class

Description

Displays the MUX administration page for deleting a MUX subscription handler.

AdministerMuxHandlerEditMode.class

Description

Displays the MUX administration Page for editing the configuration values of a MUX subscription handler.

AdministerMuxHandlerError.class

Description

Displays the error page when an error occurs while administering the MUX subscription handler.

MuxPublicationHandler.class

Description

Handles the publishing of XML messages from Blue Martini to the MUX e*Way using the MUX Java Client APIs.

MuxHandler.class

Description

Loaded by the Application Messaging Gateway and serves as an entry point to the MUX subscription handler. It will load any pre-existing MUX subscription handlers.

Appendix B

B.1 Openssl

The purpose of this appendix is to provide detailed information on the usage of the **openssl** utility. **Openssl** is a free implementation of cryptographic, hashing, and public key algorithms such as 3DES, SHA1, and RSA respectively. The **openssl** utility has many options including certificate signing that keytool does not provide. Openssl can be downloaded from:

http://www.openssl.org

Follow the build and installation instruction for **openssl**.

To learn more about SSL, and the high level aspects of cryptography, a good source of reference is a book entitled *SSL and TLS: Designing and Building Secure Systems* (by Eric Rescorla, Published by Addison Wesley Professional; ISBN: 0201615983).

A sample follows that demonstrates the use of the **openssl** utility to create a CA. This generated CA is then used to sign a CSR (whether generated from keytool or **openssl**).

B.1.1 Creating a Sample CA Certificate

For testing purposes a sample CA can be generated. To avoid spending additional funds to have a commercial CA sign our test certificates, a sample is generated, and used to sign the test certificate.

Perform the following from the command line.

1

```
openssl  req  -config c:\openssl\bin\openssl.cnf  -new  -x509  -keyout
ca-key.pem.txt -out  ca-certificate.pem.txt  -days  365

Using configuration from c:\openssl\bin\openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
.................++++++
.....................++++++
writing new private key to 'ca-key.pem.txt'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) []:US
State or Province Name (full name) []:California
Locality Name (eg, city) []:Monrovia
Organization Name (eg, company) []:SeeBeyond
Organizational Unit Name (eg, section) []:Development
Common Name (eg, your websites domain name)
[]:development.seebeyond.com
Email Address []:development@seebeyond.com
```

You will be prompted for information; you must enter a password and remember this password for signing certificates with the CA's private key. This command creates a private key and the corresponding certificate for our CA. The certificate is valid for 365 days starting from the date and time it was created.

The configuration file "C:\openssl\bin\openssl.cnf" is needed for the req command. The default config.cnf file is in the **openssl** package under "apps" subdirectory.

Note: *That to use this file in Windows, you must change the paths to use double back-slashes. See Appendix B for a complete Config.cnf that is known to work in a Windows environment.*

B.1.2 Signing Certificates With Own CA

Let's create a CSR with keytool and generate a signed Certificate for the CSR with the CA we had created. The following steps for generating a keystore and a CSR were already described in the sub-section "Creating a Keystore in JKS Format" of the "Methods for Generating a KeyStore and a TrustStore" section. No details are given here for the keytool commands; refer to the fore mentioned sections for the details.

1

```
keytool -keystore clientkeystore -genkey -alias client

Enter keystore password:  seebeyond
What is your first and last name?
[Unknown]:  development.seebeyond.com
What is the name of your organizational unit?
[Unknown]:  Development
What is the name of your organization?
[Unknown]:  SeeBeyond
What is the name of your City or Locality?
[Unknown]:  Monrovia
What is the name of your State or Province?
[Unknown]:  California
What is the two-letter country code for this unit?
[Unknown]:  US
Is <CN=Foo Bar, OU=Development, O=SeeBeyond, L=Monrovia,
ST=California, C=US>   correct?
[no]:  yes

Enter key password for <client>
(RETURN if same as keystore password):
```

2

```
keytool  -keystore clientkeystore  -certreq  -alias client  -keyalg
rsa  -file client.csr
```

3

```
openssl  x509  -req  -CA ca-certificate.pem.txt  -CAkey ca-key.pem.txt
-in client.csr -out client.cer  -days 365  -Cacreateserial
```

This is how we create a signed Certificate for the associated CSR. The option "-Cacreateserial" is needed if this is the first time the command is issued. It is used to create an initial serial number file used for tracking certificate signing. This certificate will be valid for 365 days.

4

```
keytool  -import  -keystore clientkeystore  -file client.cer  -alias
client

Enter keystore password:  seebeyond
keytool error: java.lang.Exception: Failed to establish chain from
reply
```

We get an exception because there is no certificate chain in the client certificate so we have to import the CA's certificate into the keystore first. We can then import the client.cer itself to form a certificate chain. Thus, we need two steps :

```
keytool  -import  -keystore clientkeystore  -file CA ca-
certificate.pem.txt  -alias theCARoot

Enter keystore password:  seebeyond
Owner: EmailAddress=development@seebeyond.com,
CN=development.seebeyond.com, OU=Development, O=SeeBeyond,
L=Monrovia, ST=California, C=US
Issuer: EmailAddress=development@seebeyond.com,
CN=development.seebeyond.com,
OU=Development, O=SeeBeyond, L=Monrovia, ST=California, C=US
Serial number: 0
Valid from: Tue May 08 15:09:07 PDT 2001 until: Wed May 08 15:09:07
PDT 2002
Certificate fingerprints:
MD5:  60:73:83:A0:7C:33:28:C3:D3:A4:35:A2:1E:34:87:F0
SHA1: C6:D0:C7:93:8E:A4:08:F8:38:BB:D4:11:03:C9:E6:CB:9C:D0:72:D0
Trust this certificate? [no]:  yes
Certificate was added to keystore

keytool –import –keystore clientkeystore –file  client.cer –alias
client

Enter keystore password:  seebeyond
Certificate reply was installed in keystore
```

Now that we have a private key and an associating certificate chain in the keystore "clientkeystore", we can use it as a keystore for client (e*Way) authentication. The only caveat is that our CA certificate must be imported into the trusted certificate store of the web server to which we will be connecting. More over, the web server should be configured for client authentication (httpd.conf for Apache for example).

Appendix C

C.1 Openssl.cnf

This appendix contains the contents of the **openssl.cnf** file that can be used on Windows. Make the appropriate changes to the directories.

C.1.1 Openssl.cnf for Windows

```
#
# SSLeay example configuration file.
# This is mostly being used for generation of certificate requests.
#

RANDFILE = .rnd

####################################################################
[ ca ]
default_ca= CA_default# The default ca section

####################################################################
[ CA_default ]

dir      = G:\\openssl\\\bin\\demoCA# Where everything is kept
certs    = $dir\\certs    # Where the issued certs are kept
crl_dir= $dir\\crl   # Where the issued crl are kept
database= $dir\\index.txt# database index file.
new_certs_dir= $dir\\newcerts# default place for new certs.

certificate= $dir\\cacert.pem    # The CA certificate
serial   = $dir\\serial    # The current serial number
crl      = $dir\\crl.pem    # The current CRL
private_key= $dir\\private\\cakey.pem   # The private key
RANDFILE= $dir\\private\\private.rnd # private random number file

x509_extensions= x509v3_extensions# The extentions to add to the cert
default_days= 365    # how long to certify for
default_crl_days= 30# how long before next CRL
default_md= md5      # which md to use.
preserve = no        # keep passed DN ordering

# A few difference way of specifying how similar the request should
look
# For type CA, the listed attributes must be the same, and the
optional
# and supplied fields are just that :-)
policy   = policy_match

# For the CA policy
[ policy_match ]
countryName  = match
stateOrProvinceName= match
organizationName= match
organizationalUnitName= optional
commonName   = supplied
```

```
emailAddress = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName= optional
stateOrProvinceName= optional
localityName= optional
organizationName= optional
organizationalUnitName= optional
commonName   = supplied
emailAddress = optional

####################################################################
[ req ]
default_bits= 1024
default_keyfile = privkey.pem
distinguished_name= req_distinguished_name
attributes= req_attributes

[ req_distinguished_name ]
countryName  = Country Name (2 letter code)
countryName_min= 2
countryName_max= 2

stateOrProvinceName= State or Province Name (full name)

localityName = Locality Name (eg, city)

0.organizationName= Organization Name (eg, company)

organizationalUnitName= Organizational Unit Name (eg, section)

commonName   = Common Name (eg, your website's domain name)
commonName_max= 64

emailAddress = Email Address
emailAddress_max= 40

[ req_attributes ]
challengePassword= A challenge password
challengePassword_min= 4
challengePassword_max= 20

[ x509v3_extensions ]

# under ASN.1, the 0 bit would be encoded as 80
nsCertType   = 0x40

#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
#nsCertSequence
#nsCertExt
#nsDataType
```

Index