SeeBeyond ICAN Suite

# e*Way Intelligent Adapter ADABAS Natural User's Guide

*Release 5.0.5 for Schema Run-time Environment (SRE)*

*Java Version*

# Contents

Chapter 1

# Introduction

This chapter includes a brief description of SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for ADABAS Natural, as well as system requirements for using the e*Way and an introduction to this guide.

## 1.1 ADABAS Natural e*Way User's Guide

This document gives a general overview of the ADABAS Natural e*Way and explains how to install, configure, and operate it. The guide also explains the e*Way's usability features, as well as how to implement it in a typical e*Gate Integrator environment.

## 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 moderate to advanced-level knowledge of Windows operations and administration
- To be thoroughly familiar with CICS, Natural and Batch Natural programs
- To be familiar with Windows-style GUI operations

## 1.3 General e*Way Operation

The ADABAS Natural e*Way provides an interface to Software AG's ADABAS Natural language which allows for a generic mechanism to call Natural or Natural batch programs or to allow Natural programs to pass data to e*Gate in a reliable and efficient manner.

### 1.3.1 Basic Operation

The e*Way client components reside on the Windows and Solaris (8) operating systems. The e*Way server components reside on the host under CICS. The e*Way supports two messaging modes: inbound to e*Gate, and outbound from e*Gate following the publish/subscribe and request/reply scenarios.

The inbound e*Way receives messages from Natural programs via TCP/IP in a near real-time mode. A COBOL program provides encapsulation of the functionality required to send data to e*Gate. To enable Natural programs to send data to e*Gate, the program need only call the provided COBOL program along with the data as a parameter.

The outbound e*Way calls to the Natural programs running under the control of CICS via the External Calls Interface and returns the results back to e*Gate. It is also able to call Natural programs running outside of CICS in the batch or TSO Batch environment. This is achieved via submission of JCL to the internal reader. Pre-defined XML (eXtensible markup Language) templates describe the Events which direct the e*Way as to which programs to call, while providing the values of necessary parameters. This request/reply interface allows the e*Way to return the result of a call to one of these programs via e*Gate to the calling application.

The Natural Program Call ETD Generator creates the appropriate ETD with elements for each of the specific parameters from the source code of any Natural program that must be called.

### 1.3.2 Functional Description

While the e*Way has two messaging modes, only one messaging mode can be supported by an e*Way at any given time. To obtain bi-directional functionality, multiple e*Ways must be created and configured. The inbound e*Way receives message from Natural programs via TCP/IP in near real-time mode. The outbound e*Way makes calls to Natural programs via the External Call Interface, and returns the results back to e*Gate. The outbound e*Way also makes calls to Natural batch programs through via the execution of a batch job with parameters in the Batch or TSO environments.

The default configuration of the e*Ways do not require any user intervention or additional code, other than the specification of the e*Way parameters via the e*Gate Schema Designer, provided the customer accepts the default behavior. This functionality is described further in the sections that follow.

The parameters and configuration information for the inbound COBOL CICS program are supplied in a text-based configuration file that the user edits.

### Natural Program Security

The e*Way supports the ability to call multiple programs. The ability to invoke these programs is controlled via the ADABAS user security facility. The ADABAS user is set-up for the e*Way(s), with access to the appropriate programs. This user ID and password is stored in a file located on the mainframe. The file is read during initialization by the Natural Program Processor and the Batch Natural Program Processor.

### Outbound e*Way Functionality: Sending Data to ADABAS

The outbound functionality starts with the e*Way reading in the associated configuration file with defined parameters that have been created or modified according to necessity. The configuration file contains the information that allows the e*Way to communicate with multiple host environments.

***Note:*** *Multiple instances of the e*Way may be created and configured in a subscriber pool for higher throughput and reliability.*

The e*Way transitions into the messaging state, where it checks the queue for a pre-defined Event Type. This Event is defined in one of the pre-built Event Type Definitions shipped with the e*Way (Natural Program Call ETD or Natural Batch Program ETD).

Once the e*Way receives a message, it determines the type of request; Natural program or Batch Natural. The e*Way assembles the various parameters contained in the standard Natural/Batch Natural Program Call ETD into the appropriate ECI call.

The e*Way proceeds by making the ECI call and processes the result. If the call is successful, the e*Way checks the reply data flag for the transaction. If the flag is set, the e*Way returns the result in the format defined by the reply data ETD. After this operation, or in the situation that the flag is not set, the e*Way commits the Event in the queue.

#### Natural Program Processor

When the Event is determined to be designated for a Natural program, the Event is passed to a CICS transaction, which invokes the Natural Program Processor, and load-balances these requests across a user-configurable number of Natural sessions. Upon completion of the Natural program, the processor returns the output of the program (if any exists) and the return code to the e*Way.

#### Batch Program Processor

When the Event is determined to be designated for a batch program, the Event is passed to a CICS transaction, which invokes the Batch Program Processor. This transaction processes the Event and calls the appropriate TSO or batch program. The e*Way then waits on a successful response from the submission process.

If the "Send" is unsuccessful due to communication problems, the e*Way attempts a specified number of times to call the transaction with the supplied parameters. If the e*Way is unsuccessful after the configured number of times, it shuts down and sends an alert to e*Gate. No other transactions can be successful with a communication problem in existence.

If the error was non-communication related, the e*Way attempts a configurable number of times to call the transaction with the supplied parameters. If the e*Way remains unsuccessful, the e*Way performs the action defined by the "Failed Message Delivery Parameter," causing it to either write the message with the result of the call pre-appended to the error queue and get the next message, or shut down the e*Way for manual intervention.

If the reply data flag is set, the e*Way sends the error back to the configured reply queue, continuing to the next message without shutting down.

If the e*Way receives a shut down message from e*Gate, it sends an alert and follows the standard shut down procedures.

### Natural Program Call ETD

The Natural Program Call ETD embeds and standardizes all of the information necessary for the outbound ADABAS Natural e*Way to call a Natural. The outbound e*Way only accepts ETDs of this type, or the Batch equivalent. Any message can be converted into the required XML format, before being sent to the e*Way. The ETD utilizes the XML standard as the associated structure to embed the fields.

**Note:** *It is the responsibility of the rest of the e*Gate architecture to validate the XML format of incoming messages based on the associated DTDs, or to ensure the validity, as the e*Way does not validate them.*

The XML format has the following general structure:

```
<transact_request>
<unique_event_id>trans#_123456789</unique_event_id>
<natural_application>natural_application1</natural_application>
<natural_program>natural_program1</natural_program>
<comm_area_payload>data</comm_area_payload>
</transact_request>
```

The Natural Program Call ETD generator can be used to generate an ETD that represents the structure of the Comm Area that needs to be passed to the ECI call expected by the Natural Program Processor.

**Note:** *It is imperative that the **unique_event_id** be truly unique if there is a need to match the reply data with the original request.*

### Natural Batch Program ETD

The Natural Batch Program ETD embeds all of the information necessary to execute a batch program with optional parameters.

The general structure is similar to the following structure:

```
<batch_request>
<unique_event_id>trans#_123456789</unique_event_id>
```

```
<batch_prog>
<batch_name>proc1</batch_name>
<parms>
    <parameter><parm_name>p1</parm_name><parm_value>val1</
parm_value></parameter>
    <parameter><parm_name>p2</parm_name><parm_value>val2</
parm_value></parameter>
    <parameter><parm_name>p3</parm_name><parm_value>val3</
parm_value></parameter>
.
.
    <parameter><parm_name>pN</parm_name><parm_value>valN</
parm_value></parameter>
</parms>
</batch_prog>
</batch_request>
```

The Natural Program Call ETD generator is not used for this ETD because it is not able to interpret JCL files.

### Reply Data ETD

The Reply Data ETD contains the result of the call to the Natural Program Processor (the return code of the call, and/or the program output/content). The creation and transmission of the Event is configurable during the configuration of the e*Way. The structure of the ETD contains the Natural Program Call ETD at the beginning, with the result of the call encapsulated by another tag set. The resulting ETD can be used for both successful or unsuccessful result replies.

The populated ETD contains the original request ETD followed by the return code of the ECI call, along with the contents of the resulting ECI call.

The general structure is similar to the following structure:

```
<batch_request_reply_data>
<transact_request>
<unique_event_id>trans#_123456789</unique_event_id>
<natural_application>natural_application1</natural_application>
<natural_program>natural_prgram1</natural_program>
<comm_area_payload>data</comm_area_payload>
</transact_request>
<ECI_return_code>return_code</ECI_return_code>
<return_comm_area_payload>return_data</return_comm_area_payload>
</batch_request_reply_data>
```

or

```
<batch_request_reply_data>
<batch_request>
<unique_event_id>trans#_123456789</unique_event_id>
<batch_prog>
<batch_name>proc1</batch_name>
<parms>
<parms>
    <parameter><parm_name>p1</parm_name><parm_value>val1</
parm_value></parameter>
    <parameter><parm_name>p2</parm_name><parm_value>val2</
parm_value></parameter>
    <parameter><parm_name>p3</parm_name><parm_value>val3</
parm_value></parameter>
.
.
    <parameter><parm_name>pN</parm_name><parm_value>valN</
parm_value></parameter>
```

```
</parms>
</batch_prog>
</batch_request>
<ECI_return_code>return_code</ECI_return_code>
<return_comm_area_payload>return_data</return_comm_area_payload>
</batch_request_reply_data>
```

## Failed Message Delivery Continuation

The e*Way can be configured to either skip a message that can not be delivered due to a non-communication related problem, or shut down. The skipping of the message is performed after the message has been safely stored in an error queue. The message then has the result of the call pre-appended to it before storage.

The e*Way provides a parameter representing the number of Events that can be skipped before the e*Way shuts down. This parameter allows for the following values:

- 0 = no limit to the number of skipped messages
- 1 = shut down the e*Way on the first failed message
- >1 = the actual number of skipped messages to be tolerated

## Inbound e*Way Functionality: Receiving from ADABAS

While the outbound e*Way consists of one component, the inbound e*Way consists of two: a Communications Agent called by a Natural program running in CICS, TSO or Batch, and an e*Way (a connection multiplexing server program) configured to enable the reliable and efficient exchange of data between the Communications Agent and e*Way residing on the remote system. Multiple instances of the Communications Agents can be configured to communicate with a single or multiple e*Ways. The configuration file contains the TCP/IP route information which enables a single e*Way to communicate with multiple instances of e*Gate running on different physical machines. This provides the security information to allow the e*Way to authenticate on these remote systems if not already defined as trusted.

### Multiplexer Server e*Way

The inbound functionality begins when the inbound e*Way comes up and establishes itself. To perform this task, the parameters contained within the e*Way configuration file are read. The e*Way then creates or modifies the e*Way's functionality according to the parameters specified. The e*Way transitions into the messaging state and is ready to accept connections from the Communications Agent programs. The e*Way and the Communication Agents exchange of data via standard e*Gate TCP/IP protocol.

### Communications Agent Program

The Natural programs that send information to e*Gate must be instrumented to call the Communications Agent. The call made to this program is functionally equivalent to the following:

```
"Communications_Agent data_message"
```

When invoked, the Communications Agent reads the parameters contained within the associated configuration file. The Communications Agent uses the CICS COBOL e*Gate API Kit to send the data to the appropriate e*Way. The program blocks until the sending operation is complete. The result of the “Send” is then passed back to the calling program upon exit.

## Protocols and/or APIs

TCP/IP is used as the communication protocol between the Communications Agent program and the e*Way. The outbound e*Way utilizes the transport required by IBM’s Transaction Gateway 4.0 for the platform on which it is running.

The ETD library utilizes XML standard for the structure of Events.

## Logging

In general, the e*Way relays any pertinent information as to the state, protocol position, and any conditions that are helpful to the user to understand what is taking place according to the debug level settings, and either logs the information to a file, or to notifies the Alert Agent.

Standard debug levels are set through the standard GUI.

## Errors

Any error condition are written to the log file. The inability to write to the log file or any fatal/unrecoverable errors result in the e*Way shutting down after it sends an alert tot the Alert Agent.

## Alerting

Any errors that affect the operation of the e*Way preventing the successful delivery of a message cause an alert to be sent. If the alert can not be sent, the e*Way shuts down.

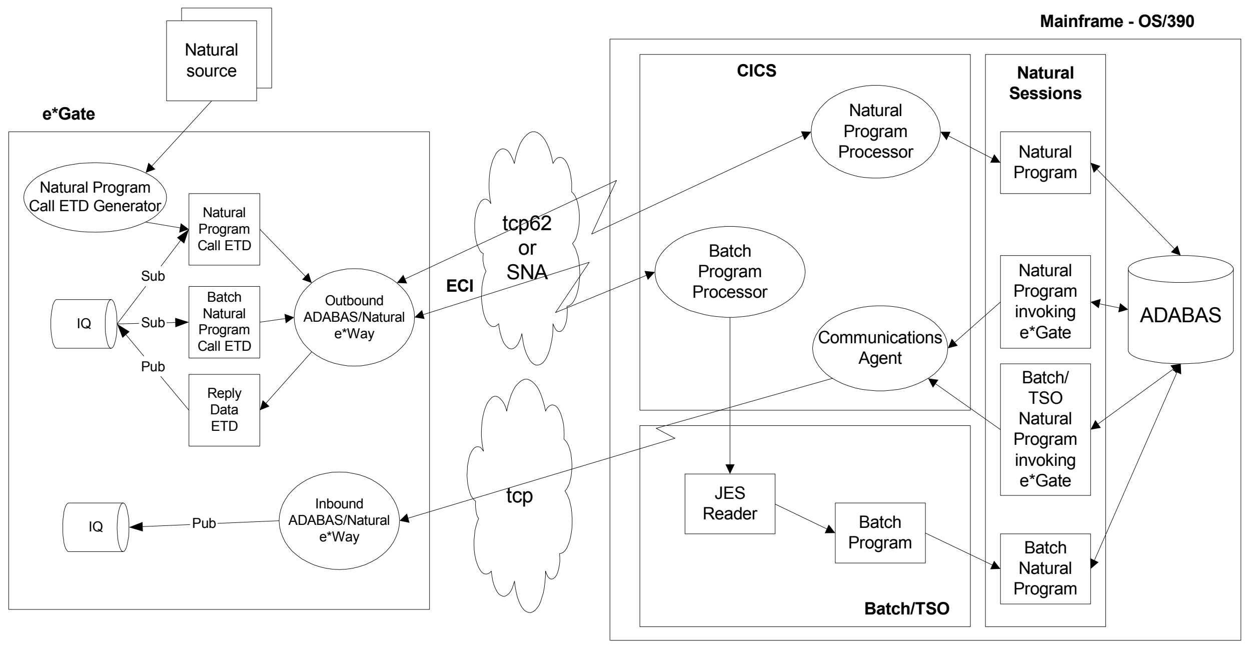
## Natural Program Call ETD Generator

The Natural Program Call ETD Generator creates the appropriate ETD containing the elements for each of the specific parameters required by the Natural program from the associated source code. This is achieved by the program reading the appropriate Natural Local Data (LDA) or Parameter Data Access (PDA) contained in a Natural SYSTRANS file.

## 1.4 Architecture: Component Interrelations

The cooperative operation of the outbound and inbound e*Ways provide a bi-directional flow of information in and out of ADABAS. Figure 1 illustrates the components involved and the interrelation these components.

**Figure 1** ADABAS Natural e*Way System Architecture



### 1.4.1 Protocols

TCP/IP is used as the communication protocol between the COBOL client program and the e*Way. The outbound e*Way utilizes the transport required by IBM's Universal Client for the platform on which it is running.

The ETD library utilizes the XML standard for the structure of its Events.

### 1.4.2 e*Way Components

The ADABAS Natural e*Way is made up of the following components:

- Multi-Mode e*Way, a core e*Gate component, executable file, **stceway.exe** (see Chapter 3 for details)
- Java methods for added functionality (see Chapter 8 for details)
- Configuration files that the e*Gate Schema Designer's e*Way Configuration Editor uses to define configuration parameters (see Chapter 4 for details)
- Additional files necessary for operation, as shown in Table 1 on page 20 (provides a complete list of installed files)

## 1.5 Supported Operating Systems

The ADABAS Natural e*Way is supported on the following operating systems:

- Windows 2000 and Windows Server 2003
- HP-UX 11.0 and 11i (PA-RISC)
- Sun Solaris 8

## 1.6 System Requirements

To use the ADABAS Natural e*Way, you need the following system requirements:

- An e*Gate Participating Host
- A TCP/IP network connection

## 1.7 External System Requirements

This section explains the ADABAS Natural e*Way's external system requirements, including personnel requirements.

### 1.7.1 CICS Transaction Gateway 4.0 Configuration

IBM CICS Transaction Gateway 4.0 is required for Java-enabled ADABAS Natural e*Ways. The following describes how to configure CICS Transaction Gateway 4.0. Transaction Gateway properties are set using the CTG Configuration Tool. The Configuration Tool is located under the CICS Transaction Gateway program menu.

For system specific settings consult the CICS Transaction Gateway Documentation or visit the IBM CICS Library Web site at the following URL:

**http://www-4.ibm.com/software/ts/cics/library/manuals/ctg40dl.html#configs.**

### 1.7.2 Suggested Personnel Requirements

The following personnel is suggested, to install and configure the system and the ADABAS Natural e*Way correctly:

- MVS/CICS system administrator and systems programmer
- RACF or equivalent security administrator
- Dedicated integration specialist to learn and operate e*Gate
- Lead developer/system administrator of applications
- Natural/ADABAS DBA

Chapter 2

# Installation

This chapter explains procedures for installing the ADABAS Natural e*Way under the following sections:

- **"Windows Systems" on page 18**
- **"UNIX Systems" on page 19**
- **"Files/Directories Created by the Installation" on page 20**

## 2.1 Windows Systems

This section explains how to install the ADABAS Natural e*Way on Windows systems.

### 2.1.1 Pre-installation

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

### 2.1.2 Installation Procedure

**To install the ADABAS Natural e*Way on Windows systems**

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

7. Clear the check boxes for all selections except **Add-ons**, and then click **Next**.
8. Follow the on-screen instructions until you come to the **Select Components** dialog box.
9. Highlight (but do not check) **e*Ways**, and then click the **Change** button. The **SelectSub-components** dialog box appears.
10. Select the **CICS e*Way**. Click **Continue** to return to the **Select Components** dialog box, then click **Next**.
11. Follow the rest of the on-screen instructions to install the ADABAS Natural e*Way.

***Caution:*** *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

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

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

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

## 2.2 UNIX Systems

This section explains how to install the ADABAS Natural e*Way on UNIX systems.

### 2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that 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 Installation Procedure

**To install the ADABAS Natural 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**

4. Start the installation script by typing

   **setup.sh**

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

**Note:** *Be sure to install the e*Way files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

6 After installation is complete, exit the installation utility and launch the Schema Designer.

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

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

## 2.3 Files/Directories Created by the Installation

The ADABAS Natural e*Way installation process installs the files listed in Table 1 within the e*Gate directory tree. Files are installed within the **eGate\client** tree on the Participating Host and committed to the "default" schema on the Registry Host.

**Table 1** Files Created by Installation

| Directories | Files |
|---|---|
| classes | stccics.jar |
| classes | stcewnatural.ctl |
| classes | stcnatural.jar |
| classes | stccics.jar |
| ThirdParty\ibmctg\classes | ctgclient.jar |
| ThirdParty\ibmctg\classes | ctgserver.jar |
| ThirdParty\gnu-getopt\classes | gnu-getopt.jar |
| etd\naturalclient | naturalclient.xsc |
| configs\naturalclient | naturalclient.def |
| etd | natural.ctl |

## 2.4 CICS Installation Verification

The simulator virtual e*Ways explained in this section allow you to perform virtual tests on the ADABAS Natural e*Way to verify that the CICS installation has been successful and that e*Way installation can be continued.

### 2.4.1 Virtual Natural e*Way Interactive Simulator

After the components have been installed on CICS, use the Virtual Natural (VN) e*Way in CICS to verify that all of the components are properly installed and working correctly.

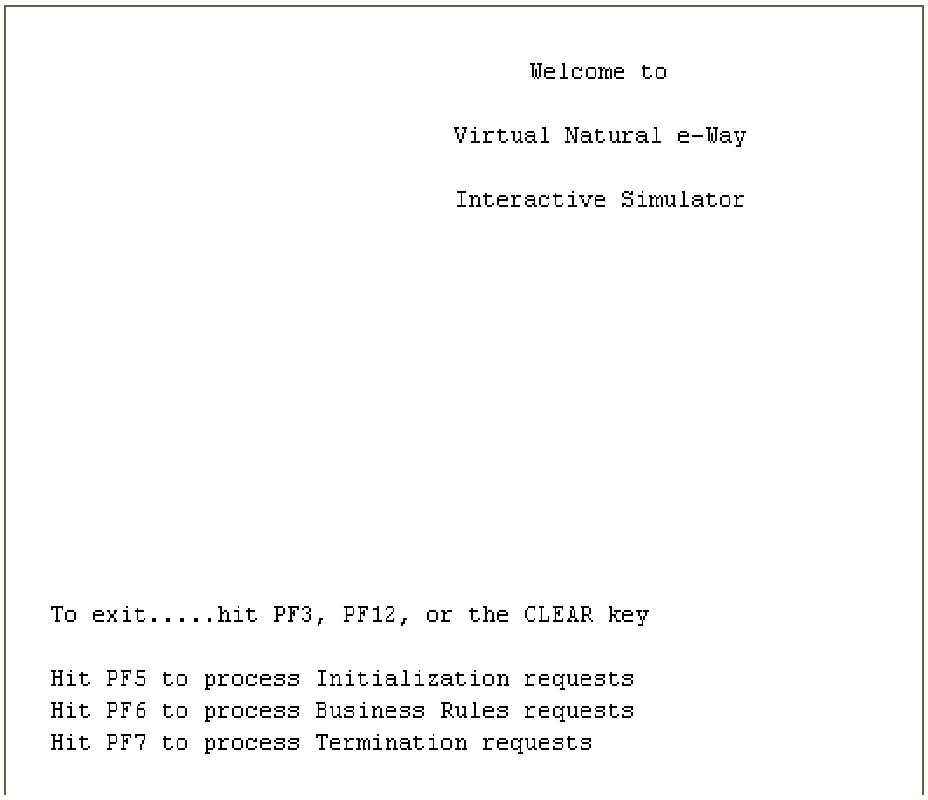
**To verify the e*Way installation and operation using the Virtual Natural e*Way**

1 Sign onto the CICS region, from a blank screen, by entering the following tranid:

```
QANE
```

2 Press ENTER to display the screen shown in Figure 2.

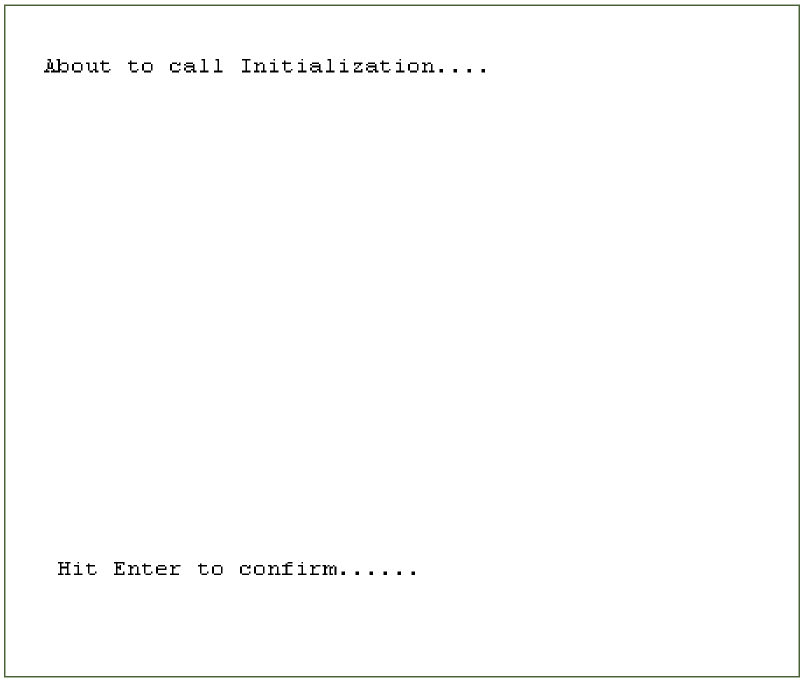
**Figure 2** VN e*Way: Welcome Screen



```
                                   Welcome to

                              Virtual Natural e-Way

                              Interactive Simulator













   To exit.....hit PF3, PF12, or the CLEAR key

   Hit PF5 to process Initialization requests
   Hit PF6 to process Business Rules requests
   Hit PF7 to process Termination requests
```

3 Press PF5 to display the screen shown in Figure 3.

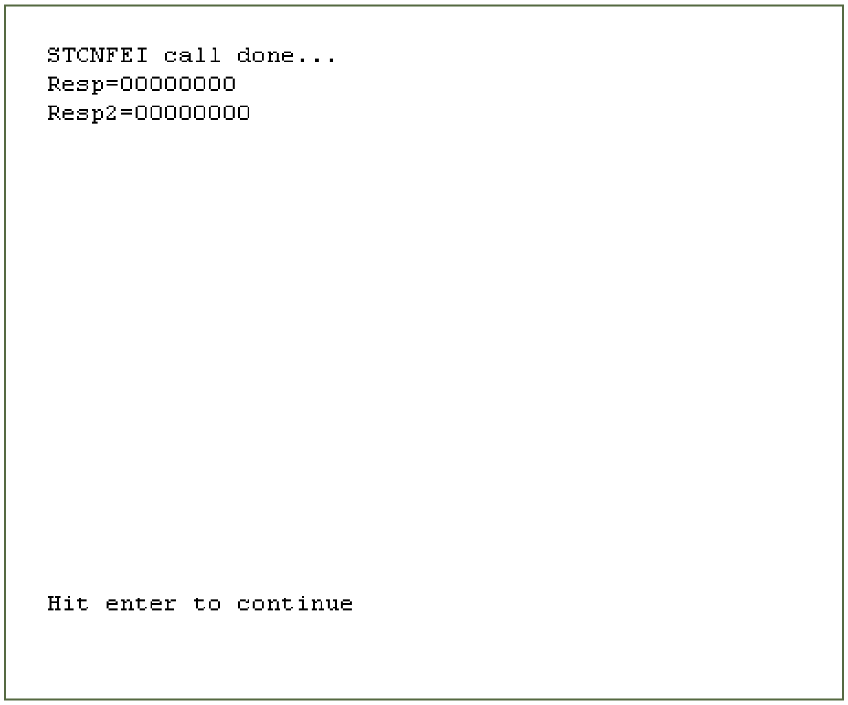
**Figure 3** VN e*Way: About To Call Screen



```
About to call Initialization....



 Hit Enter to confirm......
```

4 Press ENTER to display the screen shown in Figure 4.

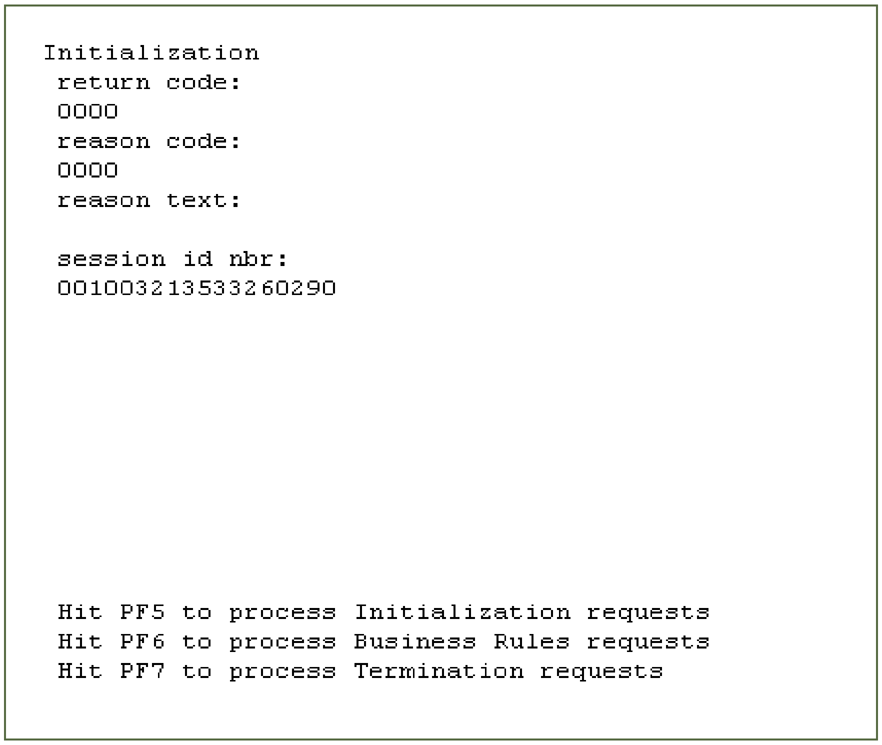
**Figure 4** VN e*Way: First Call Done Screen



```
STCNFEI call done...
Resp=00000000
Resp2=00000000



Hit enter to continue
```

5 Press ENTER to display the screen shown in Figure 5.

**Figure 5** VN e*Way: Initialization Screen



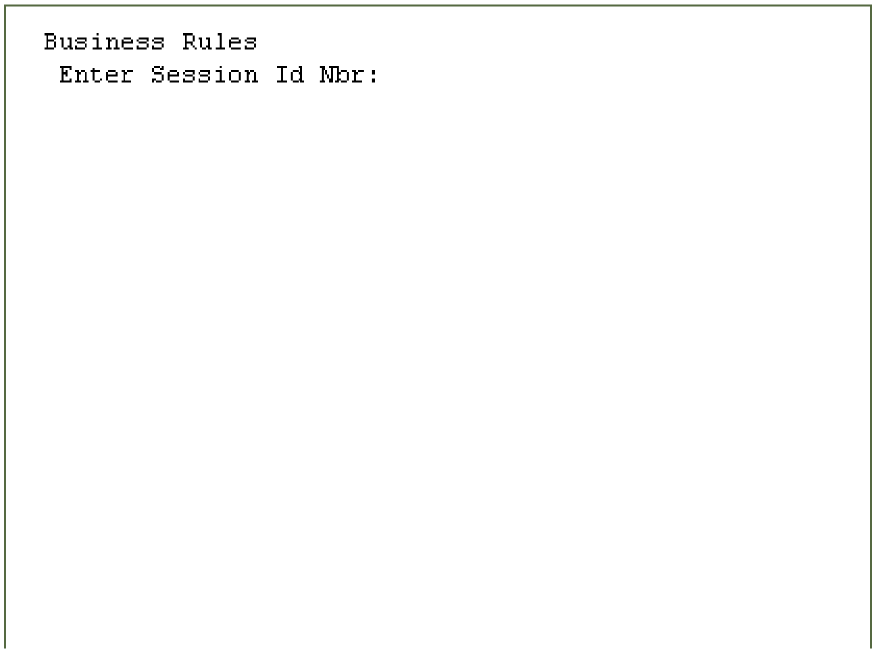
```
Initialization
 return code:
 0000
 reason code:
 0000
 reason text:

 session id nbr:
 001003213533260290









 Hit PF5 to process Initialization requests
 Hit PF6 to process Business Rules requests
 Hit PF7 to process Termination requests
```

6 While the previous screen is displayed, use the mouse to highlight the eighteen-digit number directly under **session id nbr**. This is your session ID.

7 Copy and paste this number into a Notepad (or similar text editor) screen. You must keep this screen open for later use in this test script.

8 Press PF6 to display the screen shown in Figure 6.

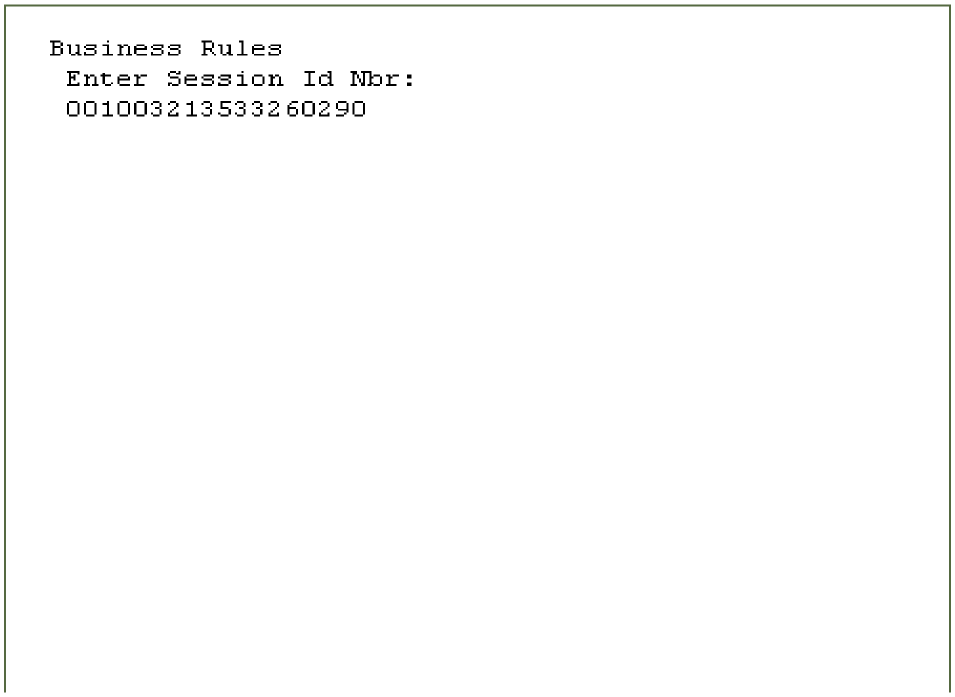
**Figure 6** VN e*Way: Session ID Screen 1



```
Business Rules
 Enter Session Id Nbr:
```

9 Paste the number from the clipboard onto the previous screen. The screen then appears as shown in Figure 7.

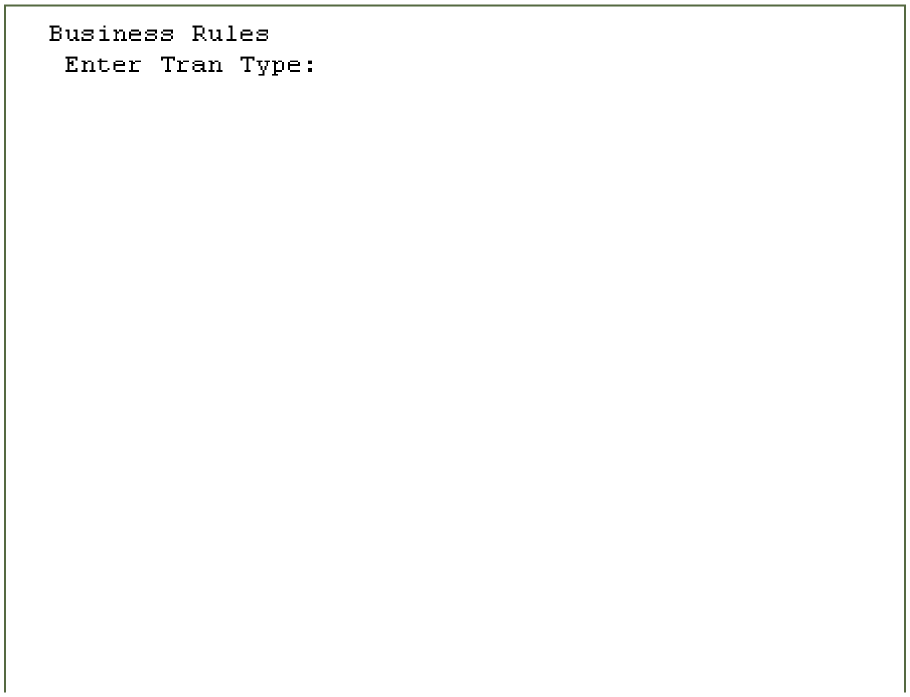
**Figure 7** VN e*Way: Session ID Screen 2



```
Business Rules
 Enter Session Id Nbr:
 0010032135332 60290
```

10 Press ENTER to display the screen shown in Figure 8.

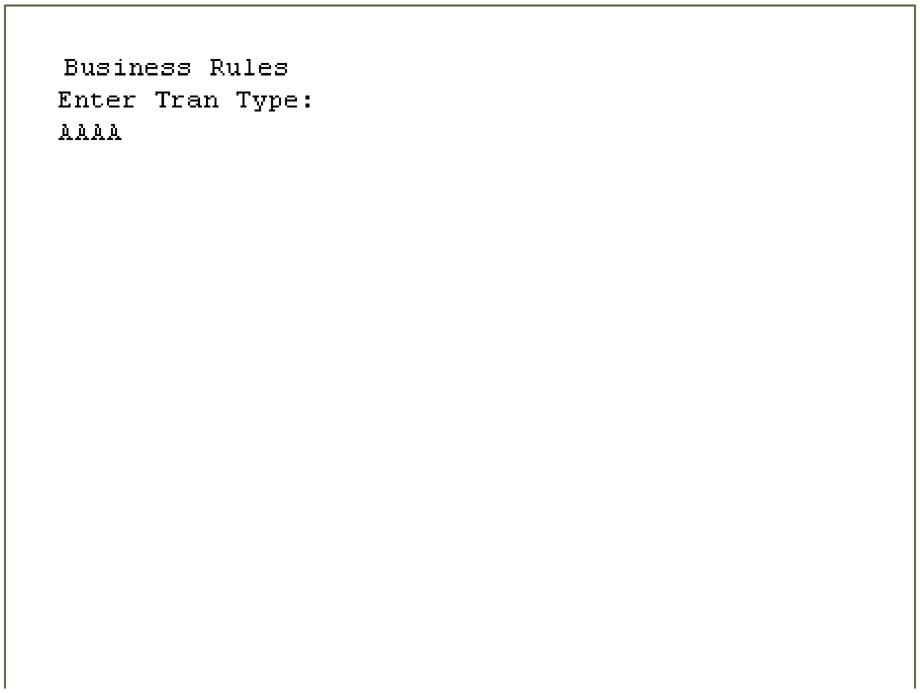
**Figure 8** VN e*Way: Transaction Type Screen 1



```
Business Rules
 Enter Tran Type:
```

11 Enter a valid transaction type that has been stored into the STCNCTL VSAM file. Enter the type as shown in Figure 9.

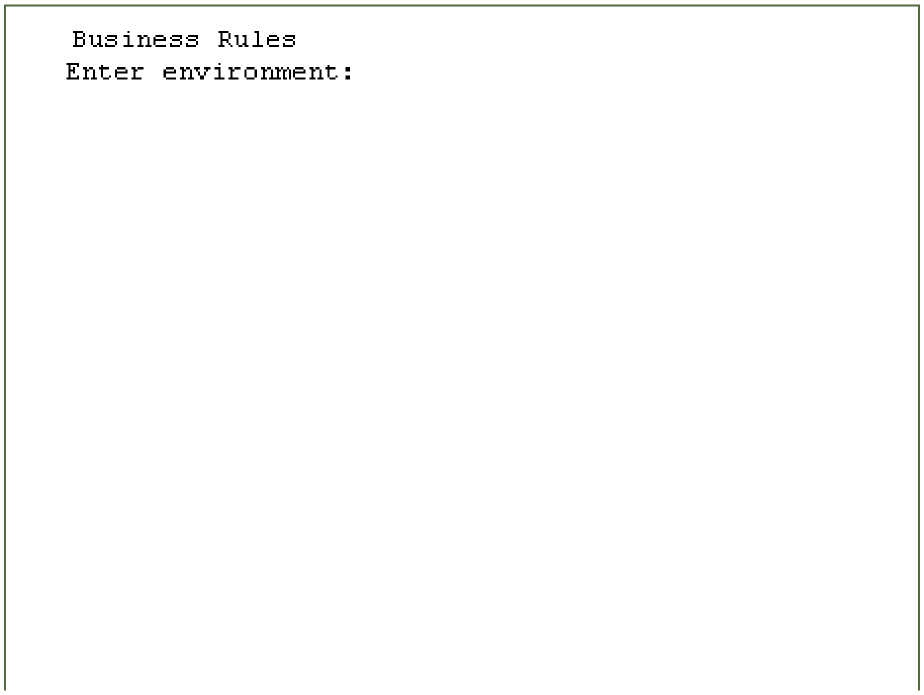
**Figure 9** VN e*Way: Transaction Type Screen 2



```
 Business Rules
Enter Tran Type:
AAAA
```

12 Press ENTER to display the screen shown in Figure 10.

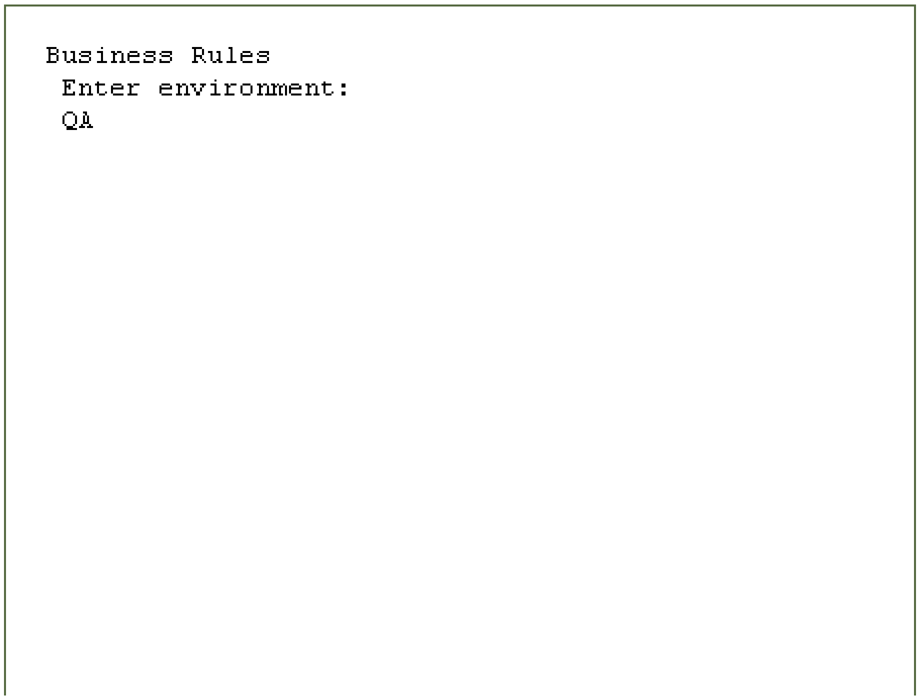
**Figure 10** VN e*Way: Environment Screen 1



```
 Business Rules
Enter environment:
```

13 Enter a valid Environment that has been stored into the STCNCTL VSAM file. Enter the environment name as shown in Figure 11.

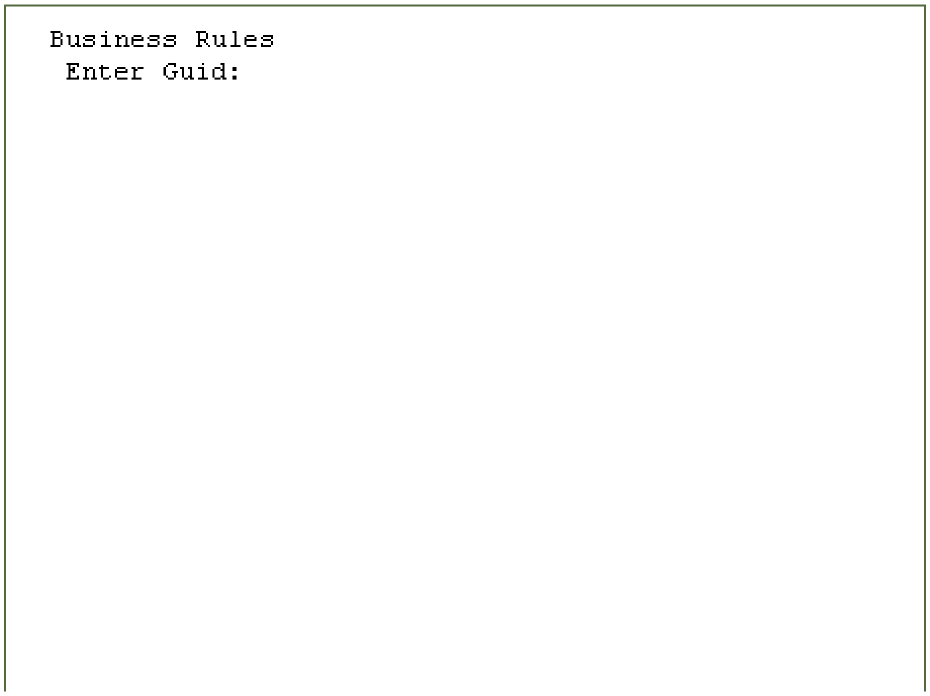
**Figure 11** VN e*Way: Environment Screen 2



```
Business Rules
 Enter environment:
 QA
```

14 Press ENTER to display the screen shown in Figure 12.

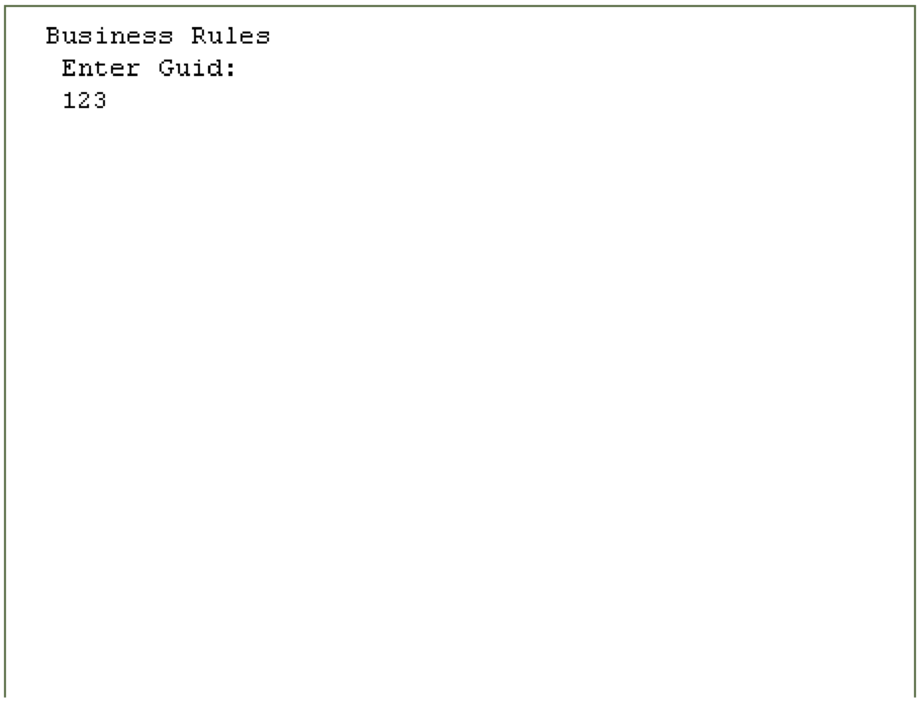
**Figure 12** VN e*Way: GUID Screen 1



```
Business Rules
 Enter Guid:
```

15 Enter a valid globally unique identifier (GUID) value (any 1-to-38-character string that you have not used for this test before) as shown in Figure 13.

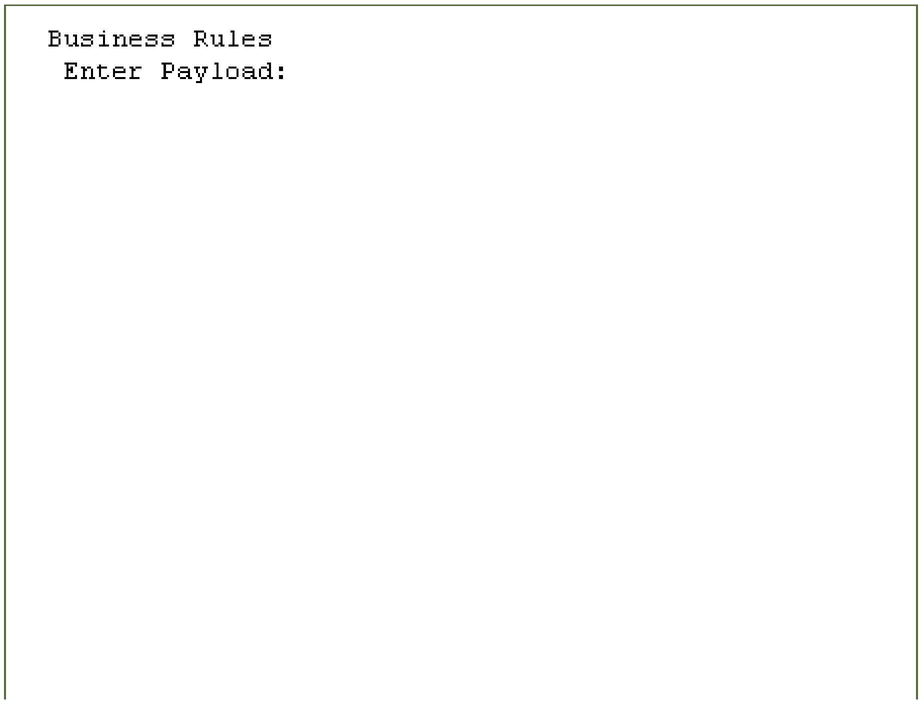
**Figure 13** VN e*Way: GUID Screen 2



```
Business Rules
 Enter Guid:
 123
```

16 Press ENTER to display the screen shown in Figure 14.

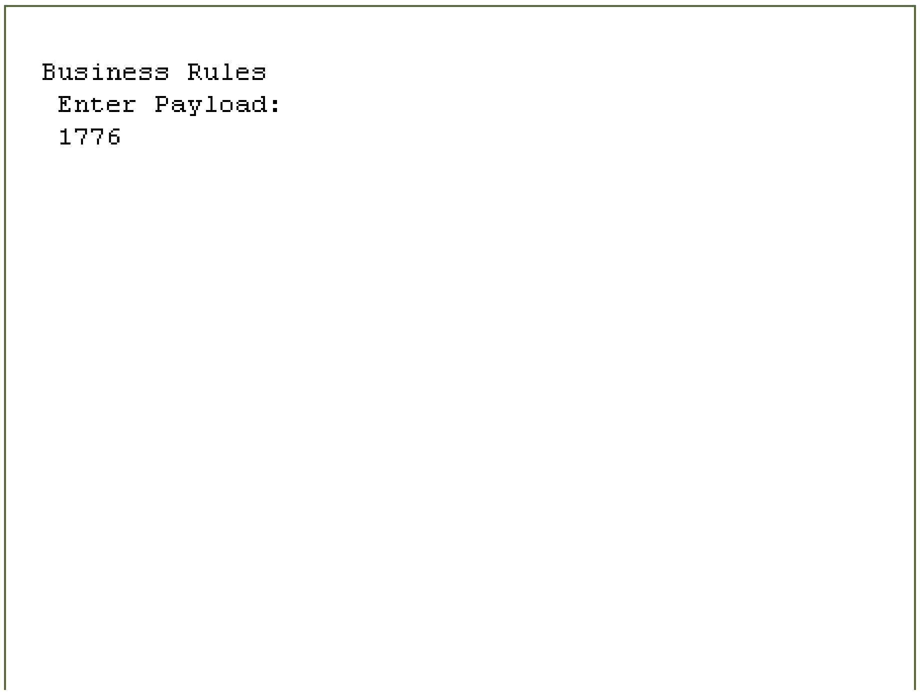
**Figure 14** VN e*Way: Payload Screen 1



```
 Business Rules
  Enter Payload:
```

17 Enter in the number 1776 as shown in Figure 15.

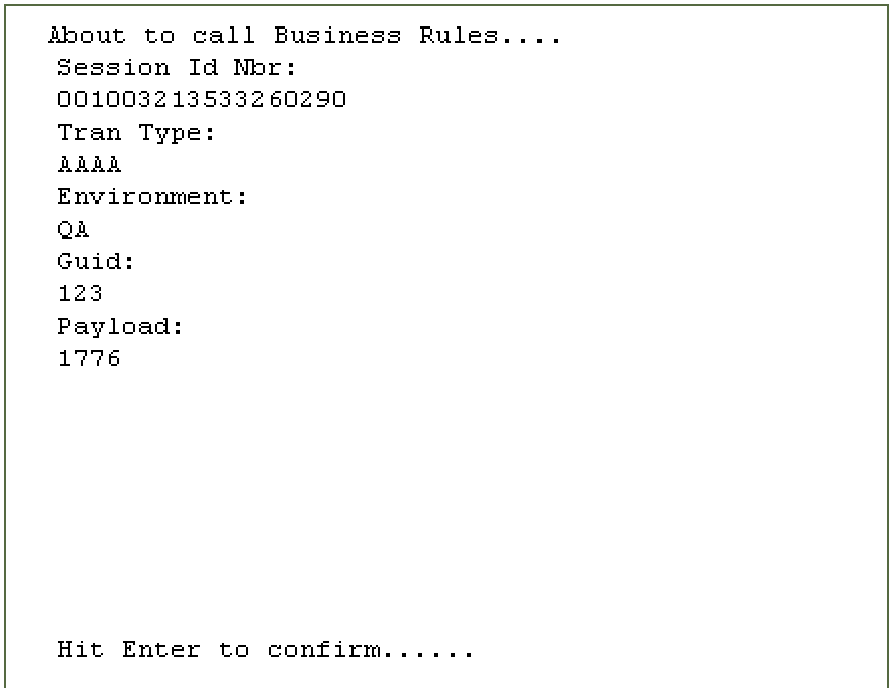
**Figure 15** VN e*Way: Payload Screen 1



```
 Business Rules
  Enter Payload:
  1776
```

18 Press ENTER to display the screen shown in Figure 16.

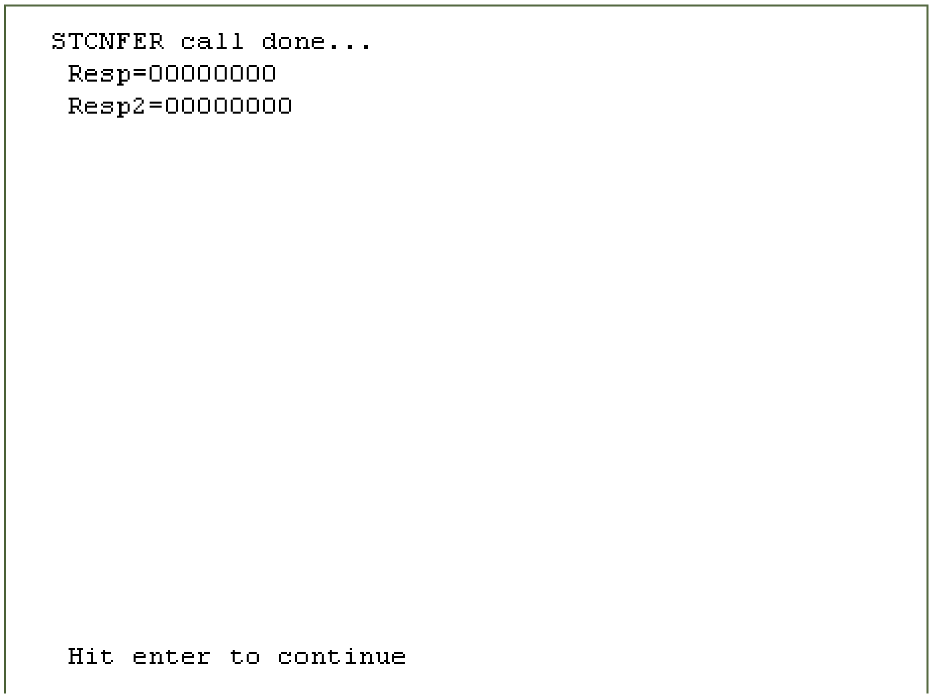
**Figure 16** VN e*Way: Call Business Rules Screen



```
About to call Business Rules....
 Session Id Nbr:
 00100321353326029O
 Tran Type:
 AAAA
 Environment:
 QA
 Guid:
 123
 Payload:
 1776








 Hit Enter to confirm......
```

19 Press ENTER to display the screen shown in Figure 17.

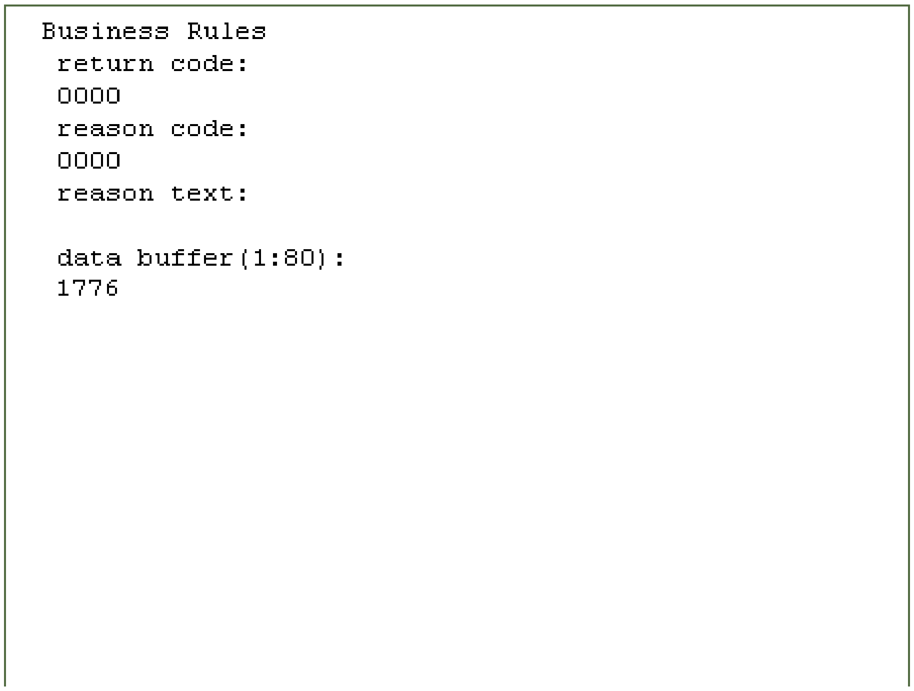
**Figure 17** VN e*Way: Last Call Done Screen



```
STCNFER call done...
 Resp=00000000
 Resp2=00000000















 Hit enter to continue
```

20 Press ENTER to display the screen shown in Figure 18.

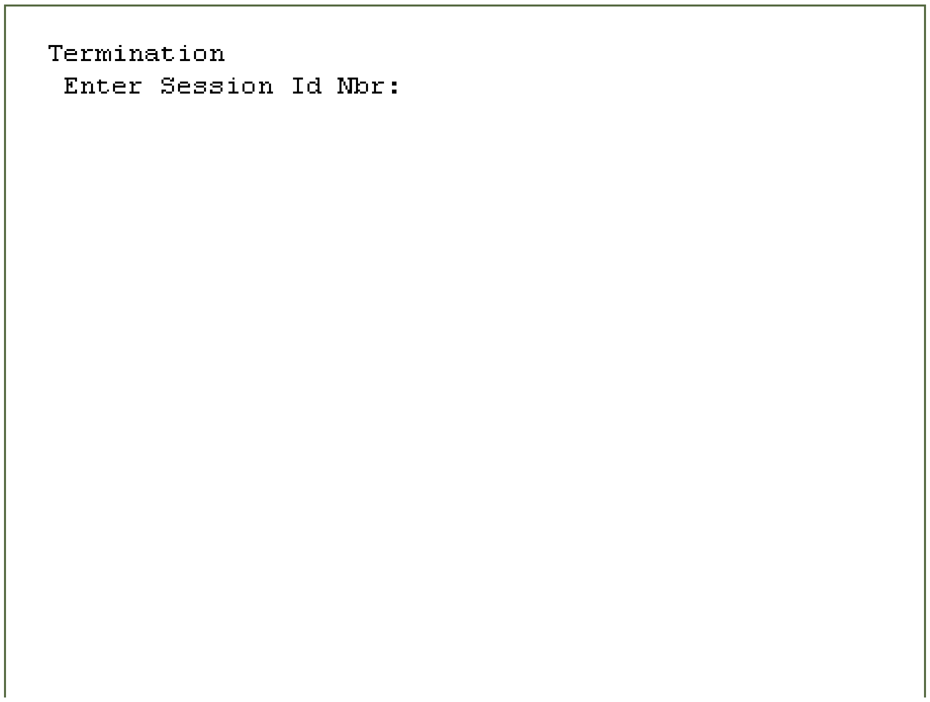
**Figure 18** VN e*Way: Return Screen



```
Business Rules
 return code:
 0000
 reason code:
 0000
 reason text:

 data buffer(1:80):
 1776
```

21 Press PF7 to display the screen shown in Figure 19.

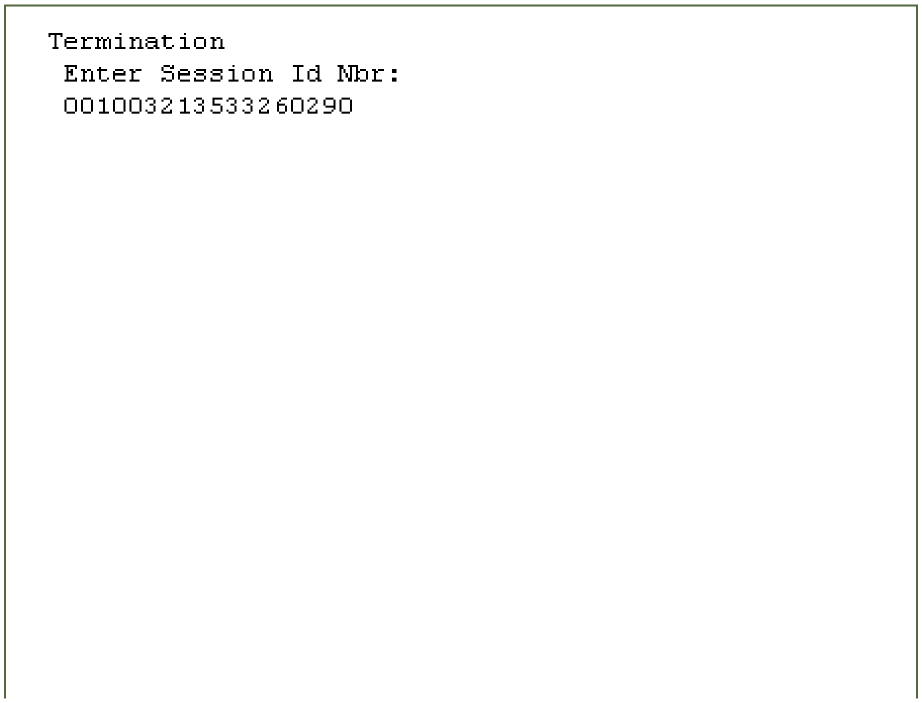
**Figure 19** VN e*Way: Termination Screen 1



```
Termination
 Enter Session Id Nbr:
```

22 Copy the session ID number from the clipboard file you opened earlier. Paste it into the screen as shown in Figure 20.

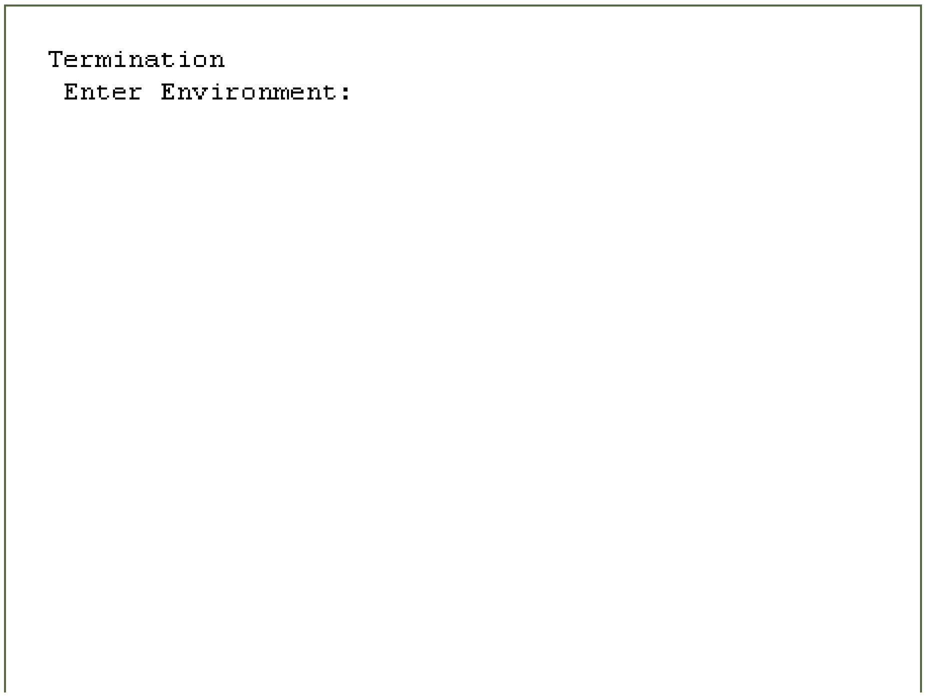
**Figure 20** VN e*Way: Termination Screen 2



```
Termination
 Enter Session Id Nbr:
 00100321353326O290
```

23 Press ENTER to display the screen shown in Figure 21.

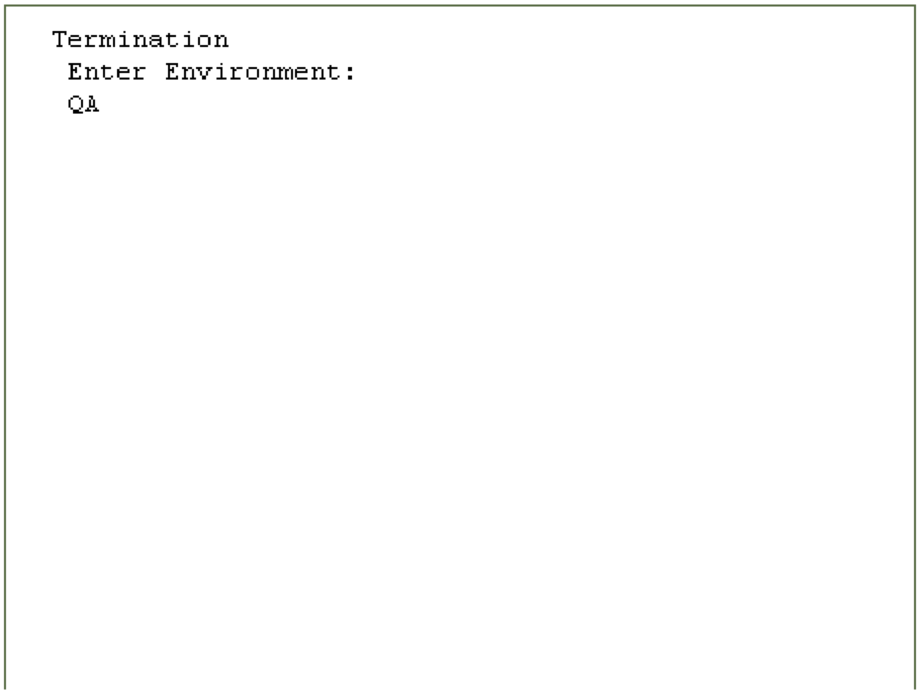
**Figure 21** VN e*Way: Termination Screen 3



```
Termination
 Enter Environment:
```

24 Enter the same environment name that you entered earlier in this script, as shown in Figure 22.

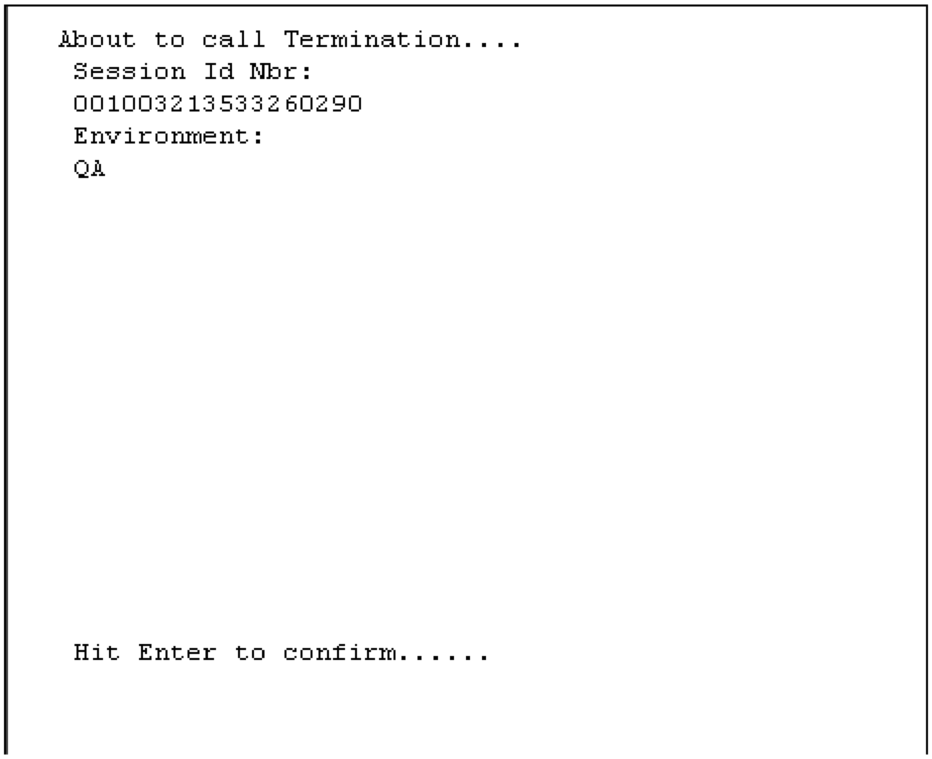
**Figure 22** VN e*Way: Termination Screen 4



```
Termination
 Enter Environment:
 QA
```

25 Press ENTER to display the screen shown in Figure 23.

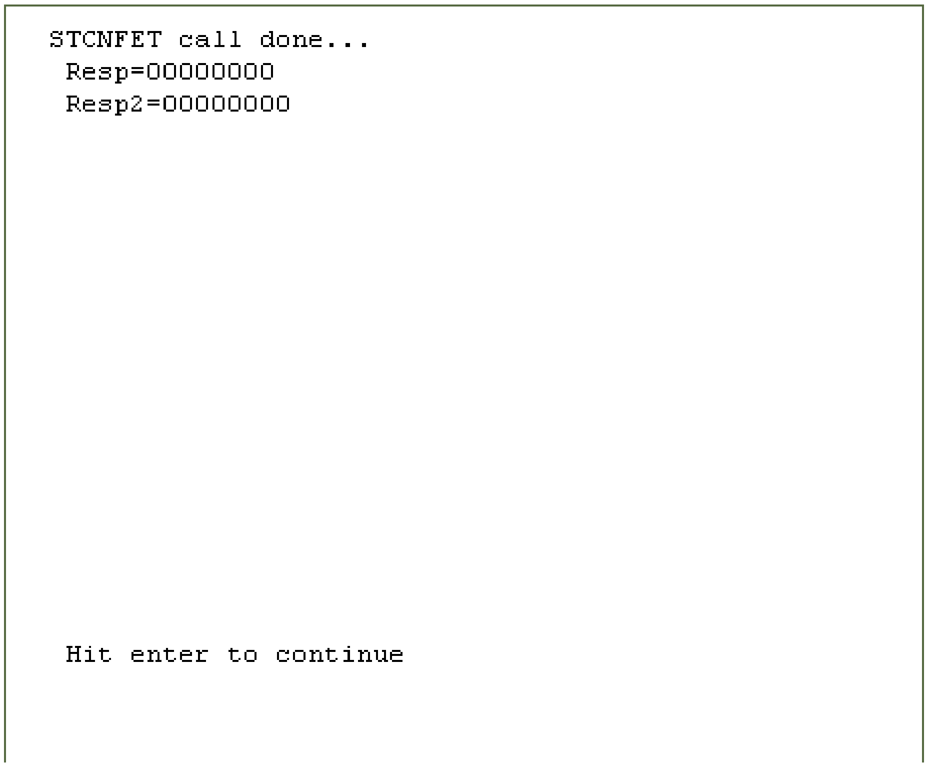
**Figure 23** VN e*Way: Termination Screen 5



```
 About to call Termination....
  Session Id Nbr:
  0010032135332 60290
  Environment:
  QA











  Hit Enter to confirm......
```

26 Press ENTER to display the screen shown in Figure 24.

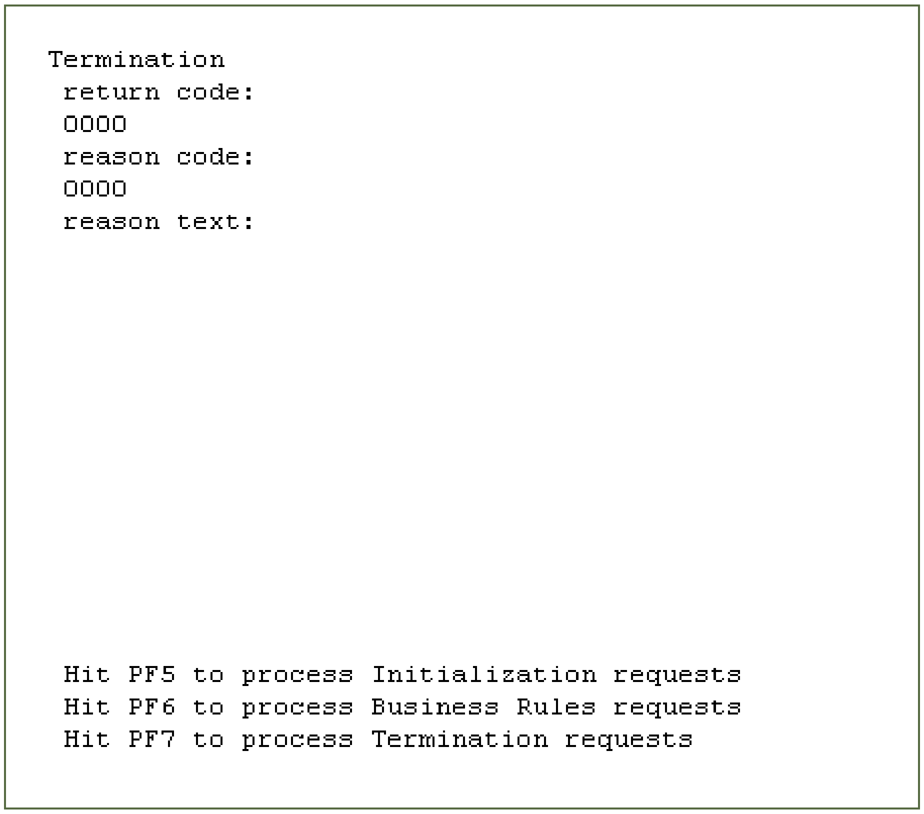
**Figure 24** VN e*Way: Termination Screen 6



```
STCNFET call done...
 Resp=00000000
 Resp2=00000000











 Hit enter to continue
```

27 Press ENTER to display the screen shown in Figure 25.

**Figure 25** VN e*Way: Termination Screen 7



```
Termination
 return code:
 0000
 reason code:
 0000
 reason text:













 Hit PF5 to process Initialization requests
 Hit PF6 to process Business Rules requests
 Hit PF7 to process Termination requests
```

28 Press CLEAR to exit the Virtual Natural e*Way.

***Note:*** *You can press PF5, PF6, or PF7 to continue testing with additional test scenarios.*

### 2.4.2 Virtual JCL Submit e*Way Interactive Simulator

After the components are installed in CICS, use the Virtual JCL Submit e*Way in CICS to verify that all components are properly installed and working correctly.

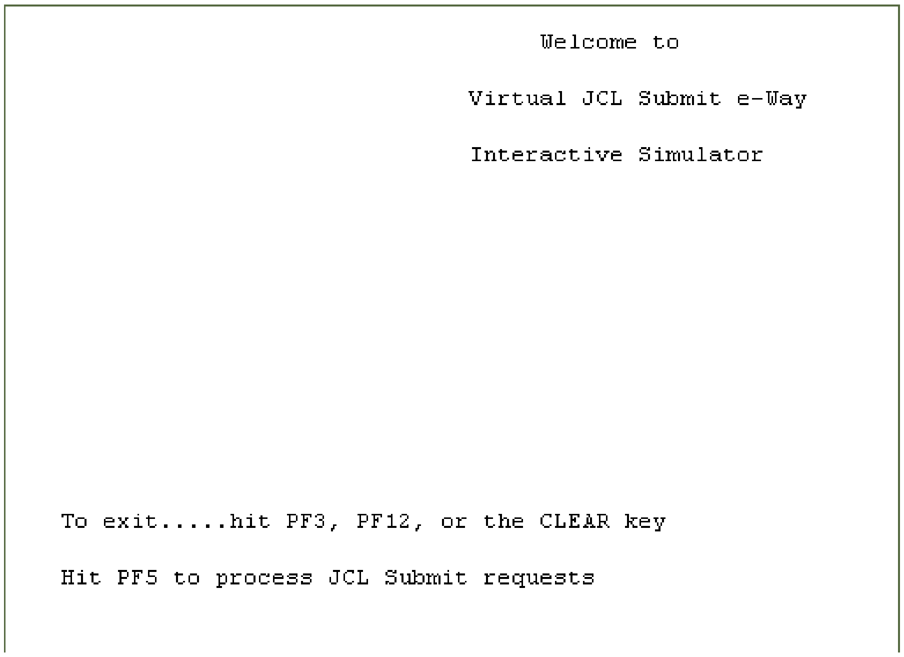
**To verify the e*Way installation and operation using the Virtual JCL Submit e*Way**

1 Sign onto the CICS region, from a blank screen, by entering the following tranid:

```
NACJ
```

2 Press ENTER to display the screen shown in Figure 26.

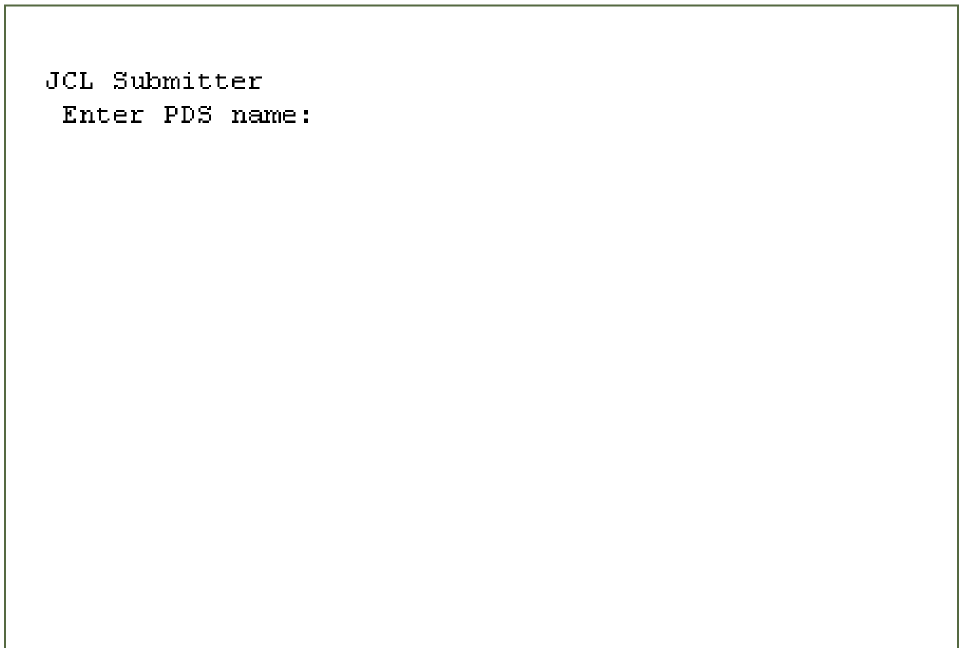
**Figure 26** JCL: Welcome Screen



```
                                        Welcome to

                                   Virtual JCL Submit e-Way

                                   Interactive Simulator




   To exit.....hit PF3, PF12, or the CLEAR key

   Hit PF5 to process JCL Submit requests
```

3 Press PF5 to display the screen shown in Figure 27.

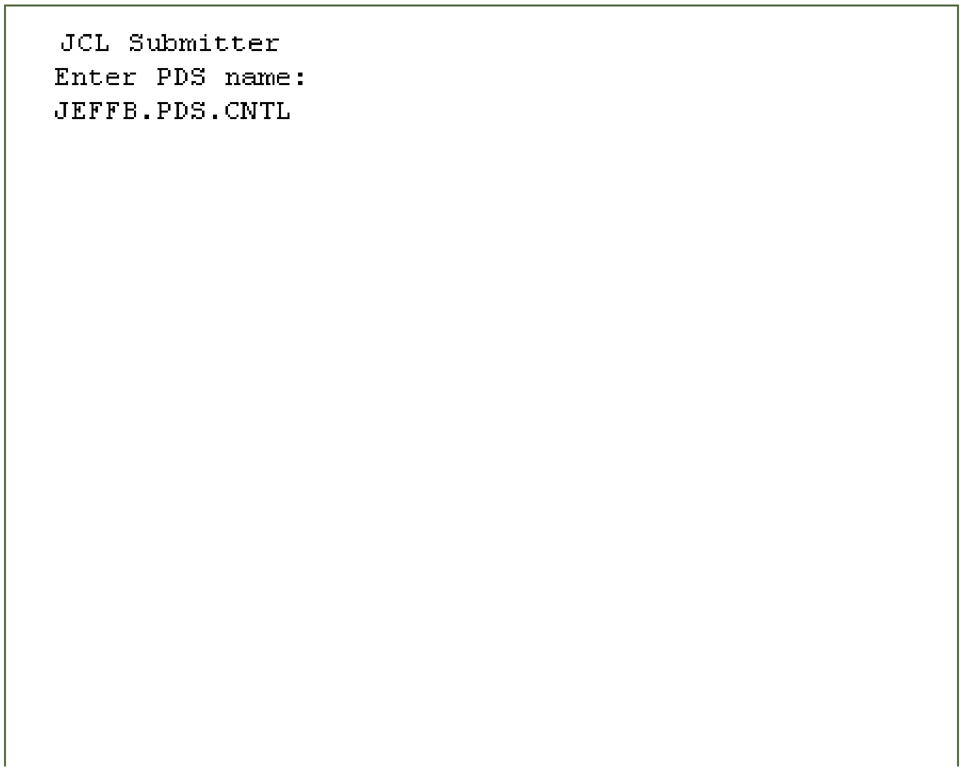
**Figure 27** JCL: PDS Name Screen 1



```
JCL Submitter
 Enter PDS name:
```

4 Enter your own test PDS name as shown in Figure 28.

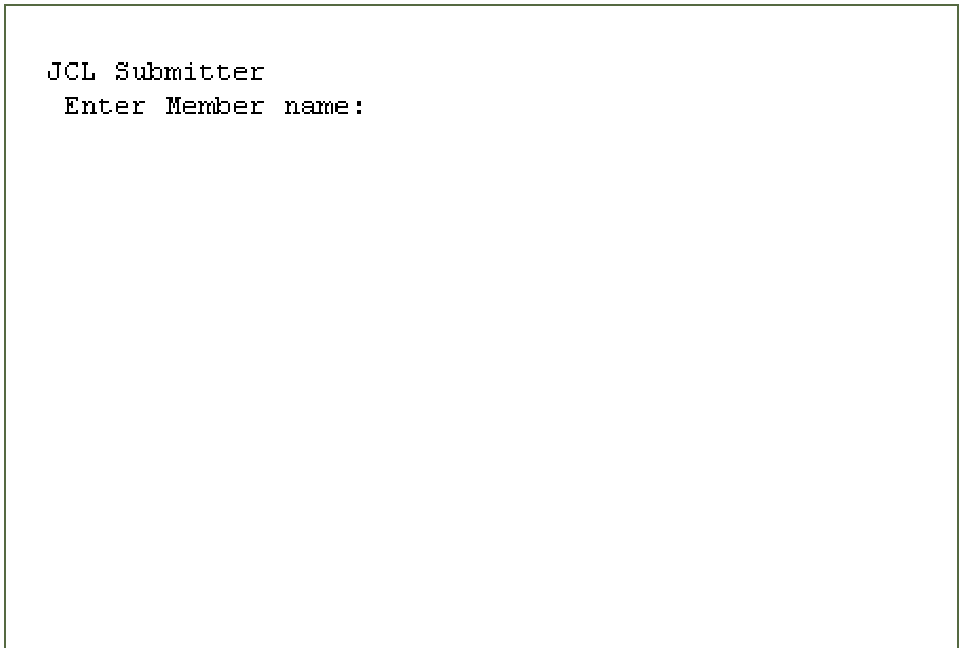
**Figure 28** JCL: PDS Name Screen 2



```
JCL Submitter
Enter PDS name:
JEFFB.PDS.CNTL
```

5 Press ENTER to display the screen shown in Figure 29.

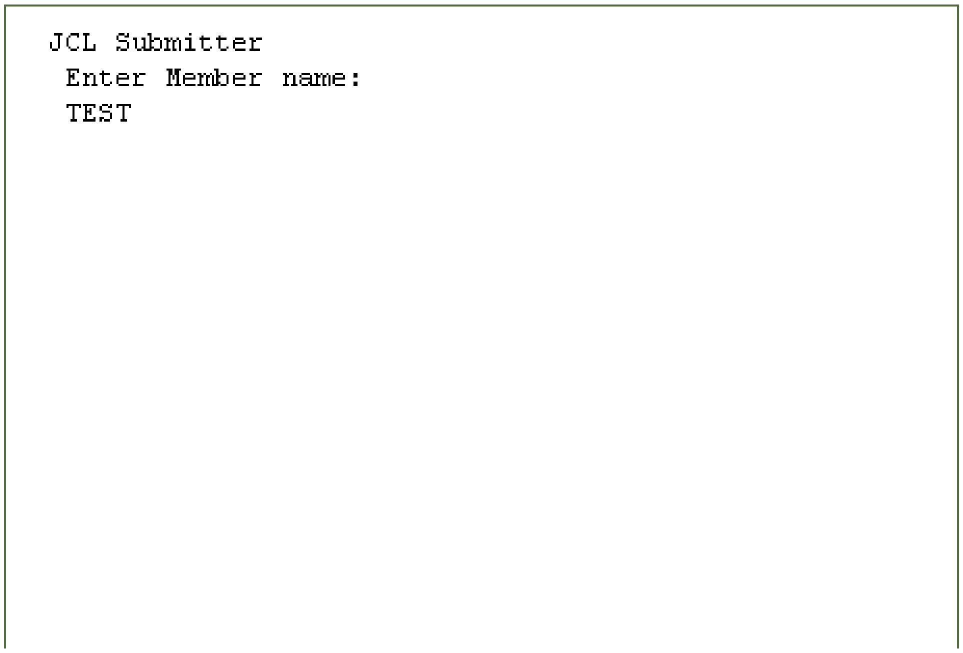
**Figure 29** JCL: Member Name Screen 1



```
JCL Submitter
 Enter Member name:
```

6 Enter your own test JCL member name as shown in Figure 30.

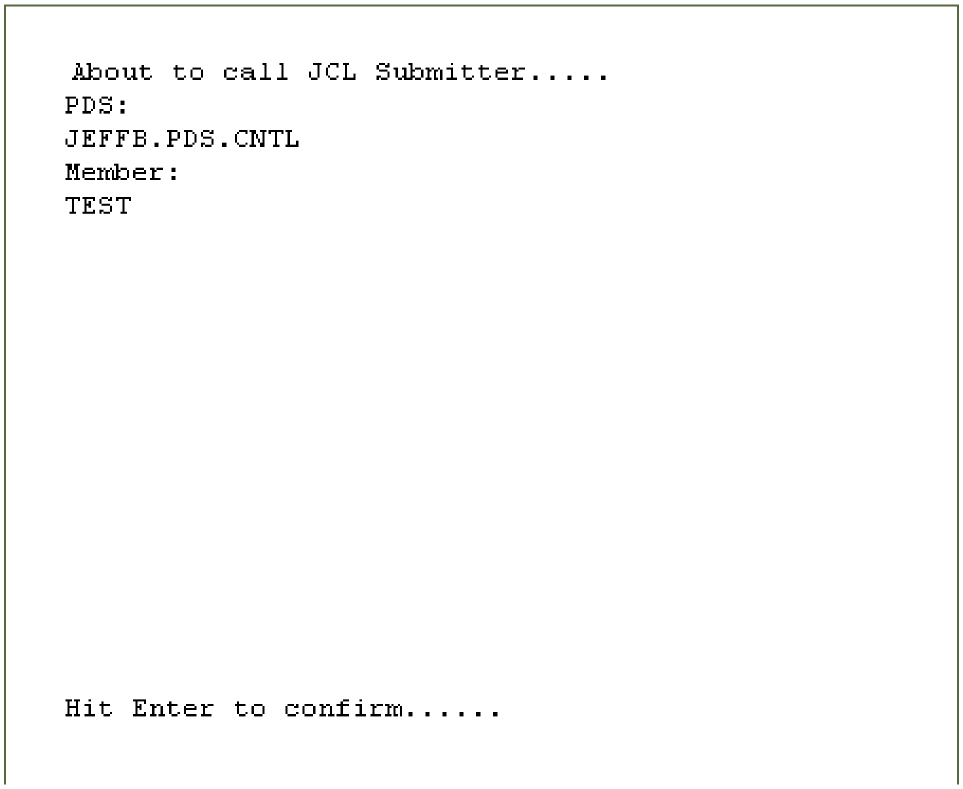
**Figure 30** JCL: Member Name Screen 2



```
JCL Submitter
 Enter Member name:
 TEST
```

7 Press ENTER to display the screen shown in Figure 31.

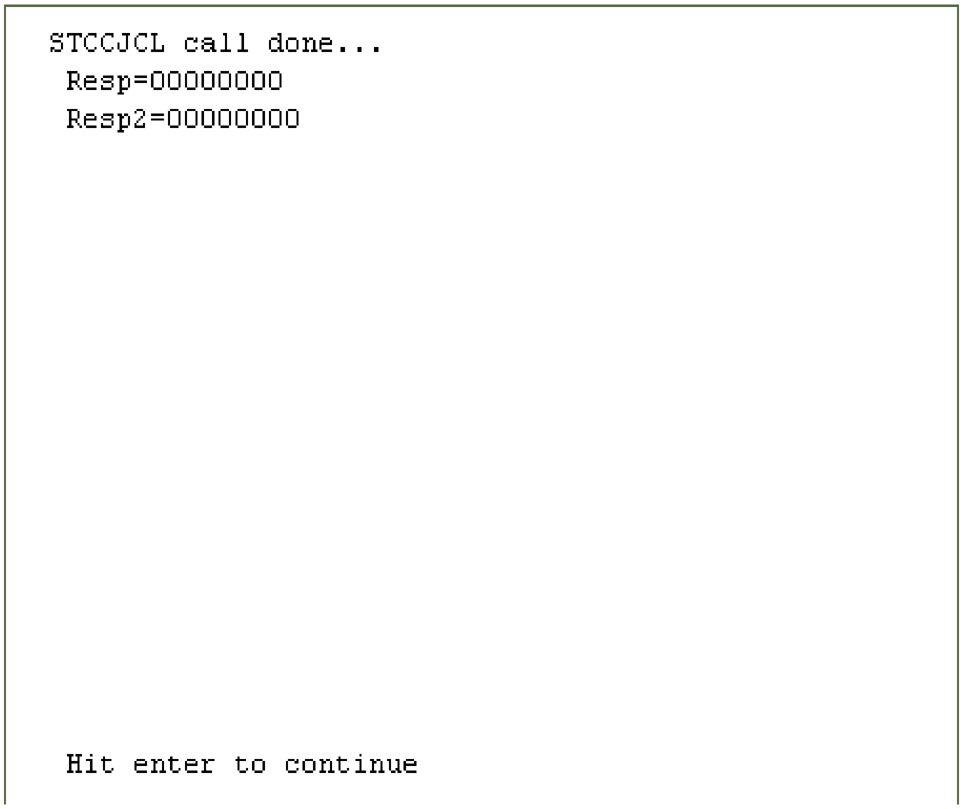
**Figure 31** JCL: About To Call Screen



```
 About to call JCL Submitter.....
PDS:
JEFFB.PDS.CNTL
Member:
TEST




Hit Enter to confirm......
```

8 Press ENTER to display the screen shown in Figure 32.

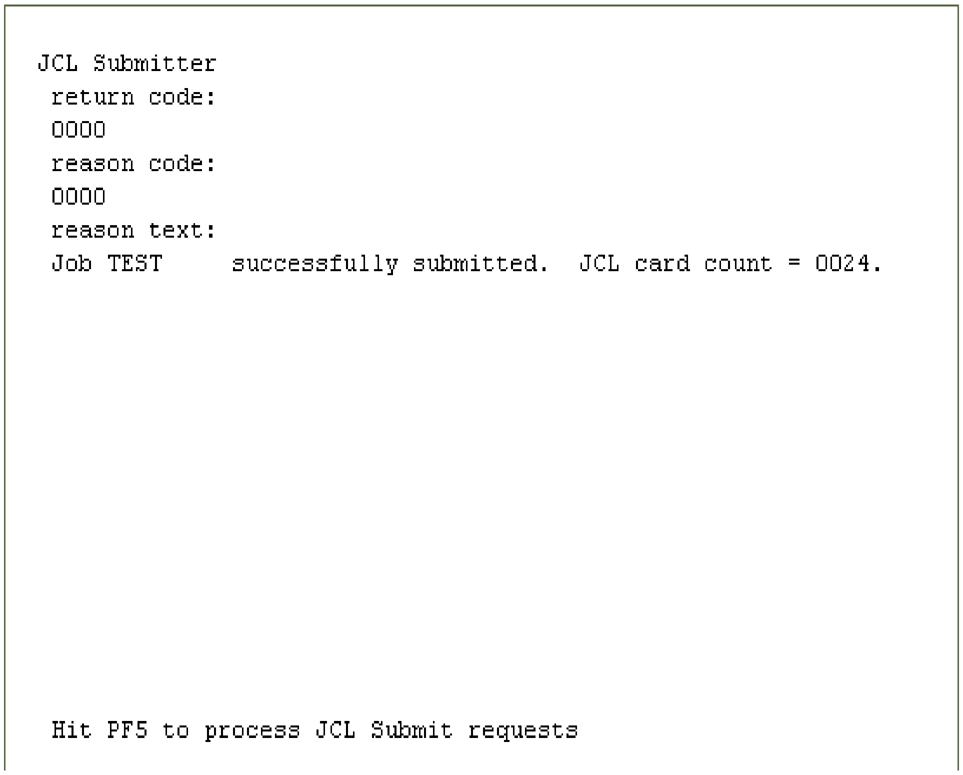
**Figure 32** JCL: Call Done Screen



```
STCCJCL call done...
 Resp=00000000
 Resp2=00000000










 Hit enter to continue
```

9 Press ENTER to display the screen shown in Figure 33.

**Figure 33** JCL: Return Screen



```
JCL Submitter
 return code:
 0000
 reason code:
 0000
 reason text:
 Job TEST     successfully submitted.  JCL card count = 0024.














 Hit PF5 to process JCL Submit requests
```

10 Press CLEAR to exit the Virtual JCL e*Way.

***Note:*** *You can press PF5, PF3, or PF12 to continue testing with additional test scenarios.*

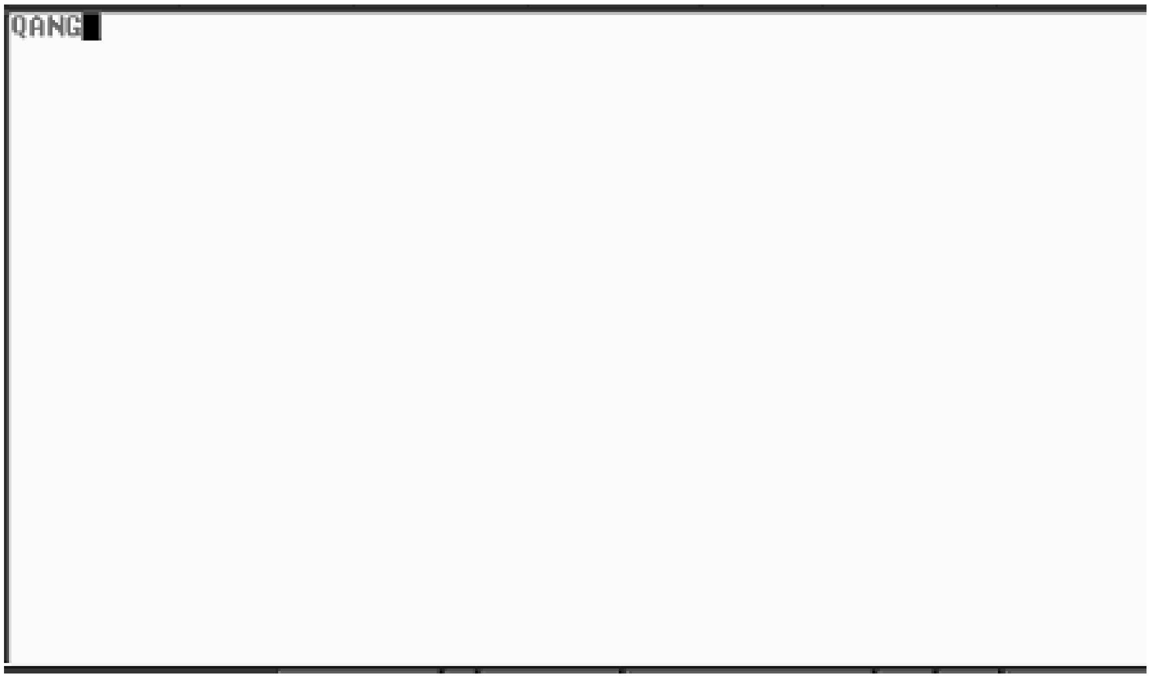
## 2.4.3 Virtual Natural e*Way Monitoring Screens OS/390 CICS

After the components are installed in CICS, use the Natural e*Way Monitoring screens in CICS to verify that all the components are properly installed and working correctly.

**To verify the e*Way installation and operation using the Virtual Natural e*Way Monitoring screens in CICS**

1. Sign on to the CICS region, from a blank screen, by entering the tranid shown in Figure 34.

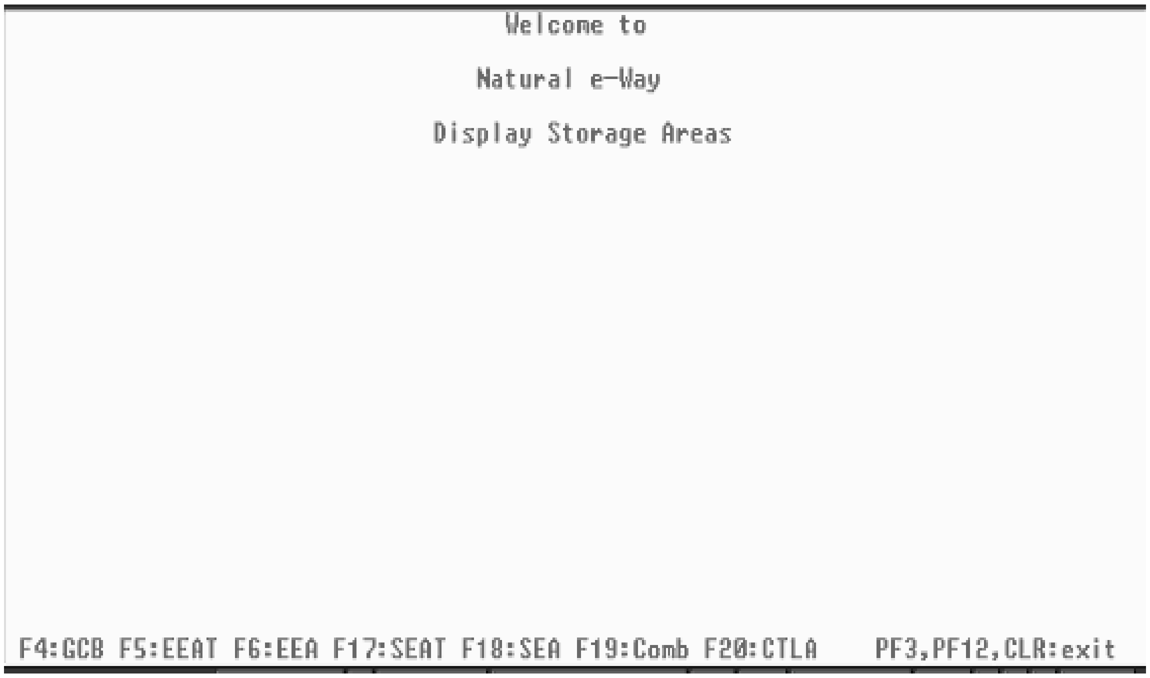
**Figure 34** JCL: Sign-on Screen



```
QANG
```

2 Press ENTER to display the screen shown in Figure 35.

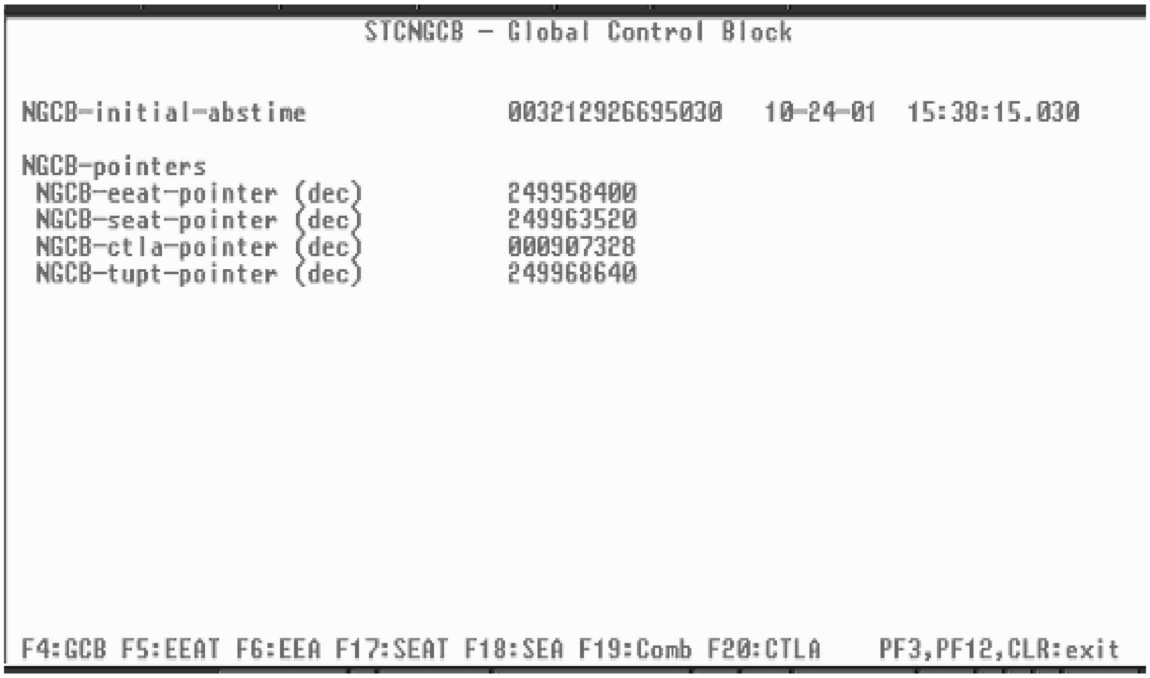
**Figure 35** CICS: Welcome Screen



```
                         Welcome to

                        Natural e-Way

                    Display Storage Areas
















F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

3 Press PF4 to display the screen shown in Figure 36.

**Figure 36** CICS: Global Control Block Screen

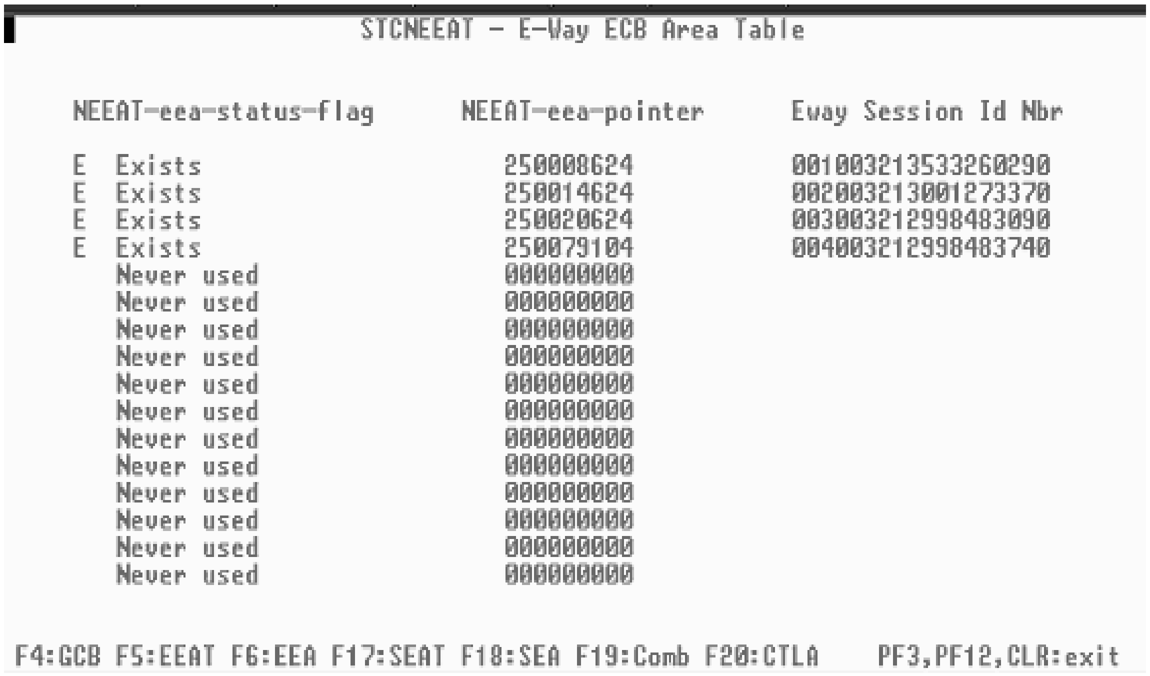


```
                    STCNGCB - Global Control Block


NGCB-initial-abstime            003212926695030   10-24-01   15:38:15.030

NGCB-pointers
 NGCB-eeat-pointer (dec)        249958400
 NGCB-seat-pointer (dec)        249963520
 NGCB-ctla-pointer (dec)        000907328
 NGCB-tupt-pointer (dec)        249968640










F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (**Figure 36 on page 42**) is the Global Control Block (GCB) screen. It displays the date and time it was created. It also displays the pointers to the internal control blocks that reside in memory in CICS that control the execution of the OS/390 components of the Natural e*Way.

4 Press PF5 to display the screen shown in Figure 37.

**Figure 37** CICS: e*Way ECB Area Table Screen 1



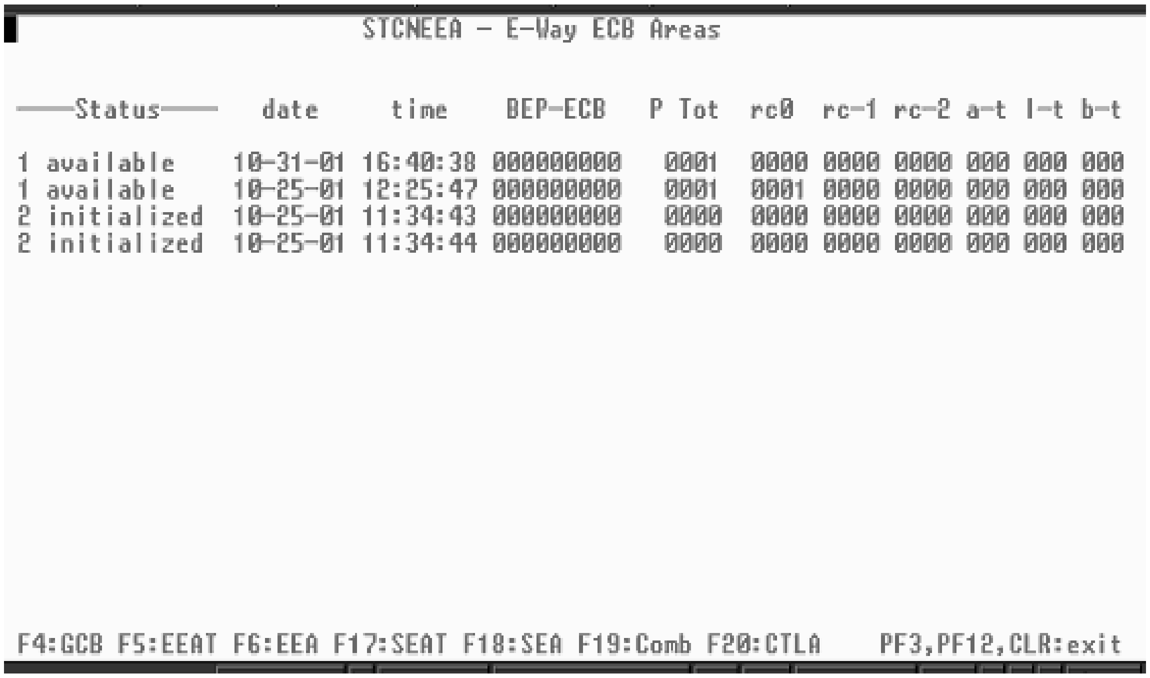
```
                          STCNEEAT - E-Way ECB Area Table


   NEEAT-eea-status-flag      NEEAT-eea-pointer      Eway Session Id Nbr

   E  Exists                     250008624           001003213533260290
   E  Exists                     250014624           002003213001273370
   E  Exists                     250020624           003003212998483090
   E  Exists                     250079104           004003212998483740
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000
      Never used                 000000000

F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (Figure 37) is the e*Way ECB Area Table (EEAT) screen. It displays the table of status flags, pointers to the e*Way ECB Areas (EEAs), and current session ID numbers assigned to each e*Way.

EEAs are created as each e*Way is initialized and connects to the CICS region. EEAs are not deleted when an e*Way terminates, therefore this screen only shows the presence of EEAs that have been created, but not the current state of each e*Way.

5 To view the EEA of each e*Way in order to know the status, press PF6. The screen shown in Figure 38 appears.

**Figure 38** CICS: e*Way ECB Area Table Screen 2



```
                         STCNEEA - E-Way ECB Areas

----Status----    date      time     BEP-ECB    P Tot   rc0  rc-1 rc-2 a-t l-t b-t

1 available    10-31-01 16:40:38 000000000   0001   0000 0000 0000 000 000 000
1 available    10-25-01 12:25:47 000000000   0001   0001 0000 0000 000 000 000
2 initialized  10-25-01 11:34:43 000000000   0000   0000 0000 0000 000 000 000
2 initialized  10-25-01 11:34:44 000000000   0000   0000 0000 0000 000 000 000

F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (Figure 38) displays information about each e*Way ECB Area (EEA) that has been created in memory in CICS. Each line displays on EEA. The status indicates whether the EEA is currently associated with an external Natural e*Way.

When the status shows **available**, the EEA is not currently being used. It is available for use by the Natural e*Way that is initialized and connects to the CICS region. The remaining fields on this screen are:

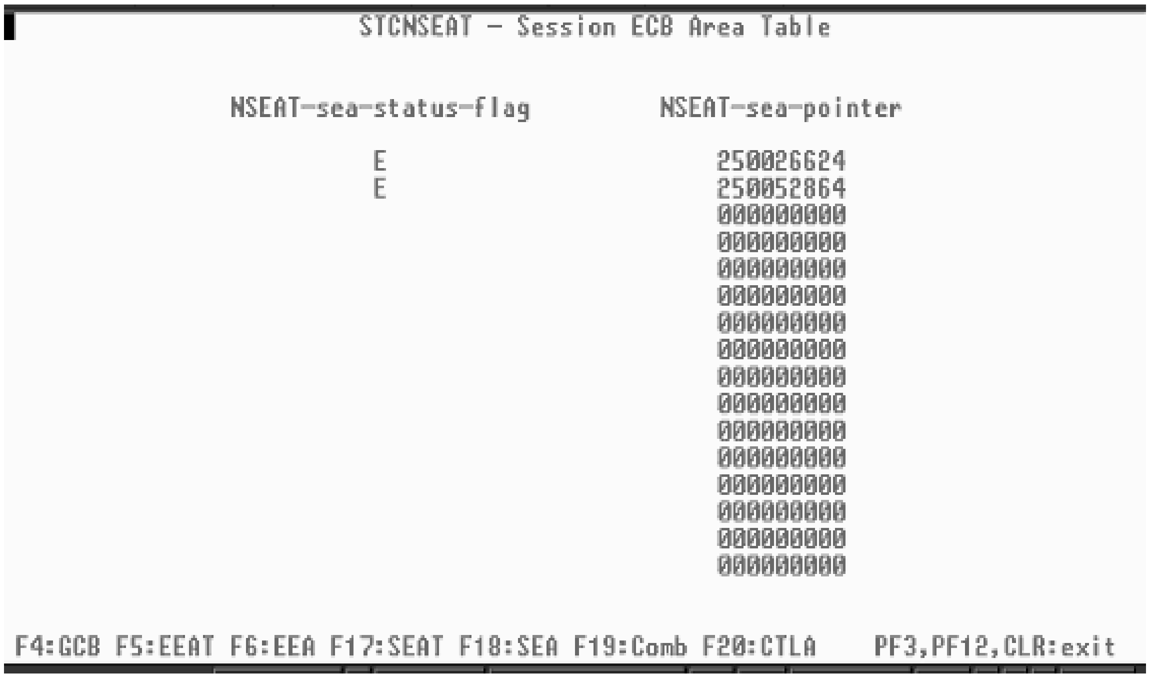
- **Date**: Date of the last update to this EEA.
- **Time**: Time of the last update to this EEA.
- **BEP-ECB**: Event Control Block (ECB) used for communicating with the Natural session Back End Program (BEP).
- **P**: Previous status of this EEA if the current status is invalid.
- **Tot**: Count of Business Rules message blocks that have been sent in from the external ADABAS Natural e*Way.
- **RC0**: Number of times the business request was processed in CICS with a return code 0 (ok) back to the ADABAS Natural e*Way.

- **RC-1**: Number of times the business request was processed in CICS with a return code –1 (error) back to the e*Way.
- **RC-2**: Number of times the business request was processed in CICS with a return code –2 (shutdown) back to the e*Way.
- **a-t**: Number of times a time-out occurred scanning for an available Natural session.
- **l-t**: Number of times a time-out occurred logging on to a new Natural session.
- **b-t**: Number of times a time-out occurred while waiting for the Natural session Back End Program (BEP) to process a Business Rule request.

There is also a set of control blocks for each back-end Natural session that is executing.

6 Press PF17 (SHIFT-F5) to display screen shown in Figure 39.

**Figure 39** CICS: Session ECB Area Table Screen 1



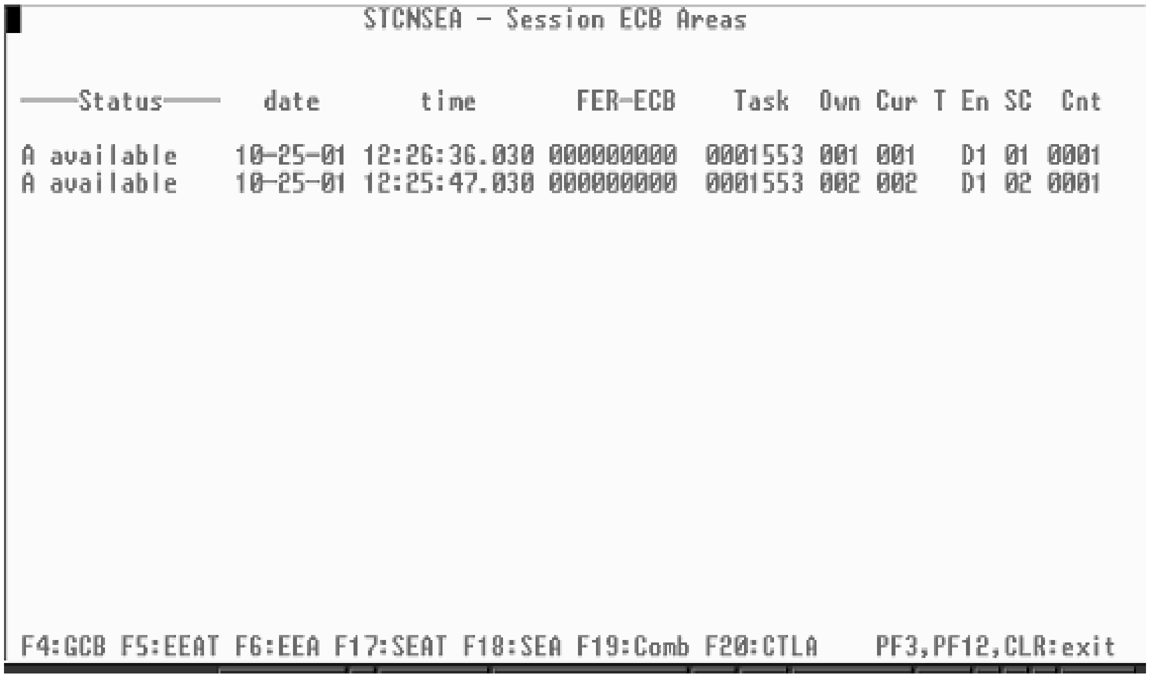
```
                 STCNSEAT - Session ECB Area Table


          NSEAT-sea-status-flag          NSEAT-sea-pointer

                    E                        250026624
                    E                        250052864
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000
                                             000000000

F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

Figure 39 shows the Session ECB Area Table (SEAT) screen. It displays the table of status flags and pointers to the Session ECB Areas (SEAs), assigned to each back-end Natural session. SEAs are created when each back-end Natural session is started in the CICS region.

SEAs are not deleted when a back-end Natural session is terminated. Therefore, this screen only shows the presence of SEAs that have been created, but not the current state of each back-end Natural session.

7 To view the SEA of each back-end Natural session, to ascertain its status, press PF18 (SHIFT-F6). The screen shown in Figure 40 appears.

**Figure 40** CICS: Session ECB Area Table Screen 2



```
                         STCNSEA - Session ECB Areas

 ----Status----     date       time        FER-ECB    Task   Own Cur T En SC  Cnt

 A available    10-25-01 12:26:36.030 000000000  0001553 001 001   D1 01 0001
 A available    10-25-01 12:25:47.030 000000000  0001553 002 002   D1 02 0001

 F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (Figure 40) displays information about each Natural session ECB Area (SEA) that has been created in memory in CICS. Each line displays one SEA. The status indicates whether the SEA is currently associated with a back-end Natural session.

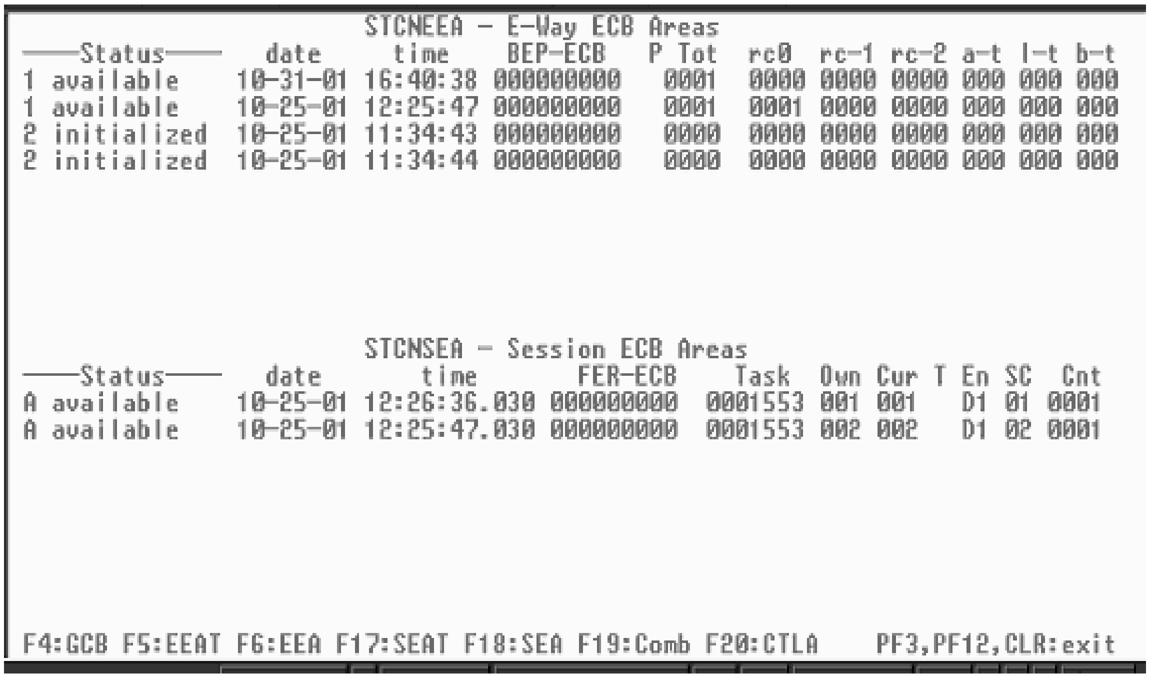
When the status shows **available**, the SEA is not currently being used. It is available for use the next time a new Natural session needs to be started in CICS. The remaining fields on this screen are:

- **Date**: Date of the last update to this SEA.
- **Time**: Time of the last update to this SEA.
- **FER-ECB**: Event Control Block (ECB) used for communicating with the Natural e*Way Front End Business Rules Program (FER).
- **Task**: The internal CICS task number of this Natural session. This is useful in associating a given Natural session to its task information that is displayed in the CEMT INQ TASK command screen
- **Own**: The session ID number of the external ADABAS Natural e*Way that started this back-end Natural session in CICS.
- **Cur**: The session ID number of the most recent ADABAS Natural e*Way that used this back-end Natural session to process a Business Rule request.

- **T**: Time-out flag set when the Front End Business Rules (FER) program timed out waiting for the Natural session Back End Program (BEP) to process a Business Rules request.
- **En**: The environment for which this Natural session is executing Business Rules requests.
- **SC**: The session configuration ID number of this Natural session. The session Configuration is stored in the STCNCTL VSAM file.
- **Cnt**: The number of Business Rules requests that have been processed by this back-end Natural session.

8 To see the external ADABAS Natural e*Ways and the back-end Natural sessions at the same time, press PF19 (SHIFT-F7). The screen shown Figure 41 in appears.

**Figure 41** CICS: e*Way and Session ECB Areas Screen

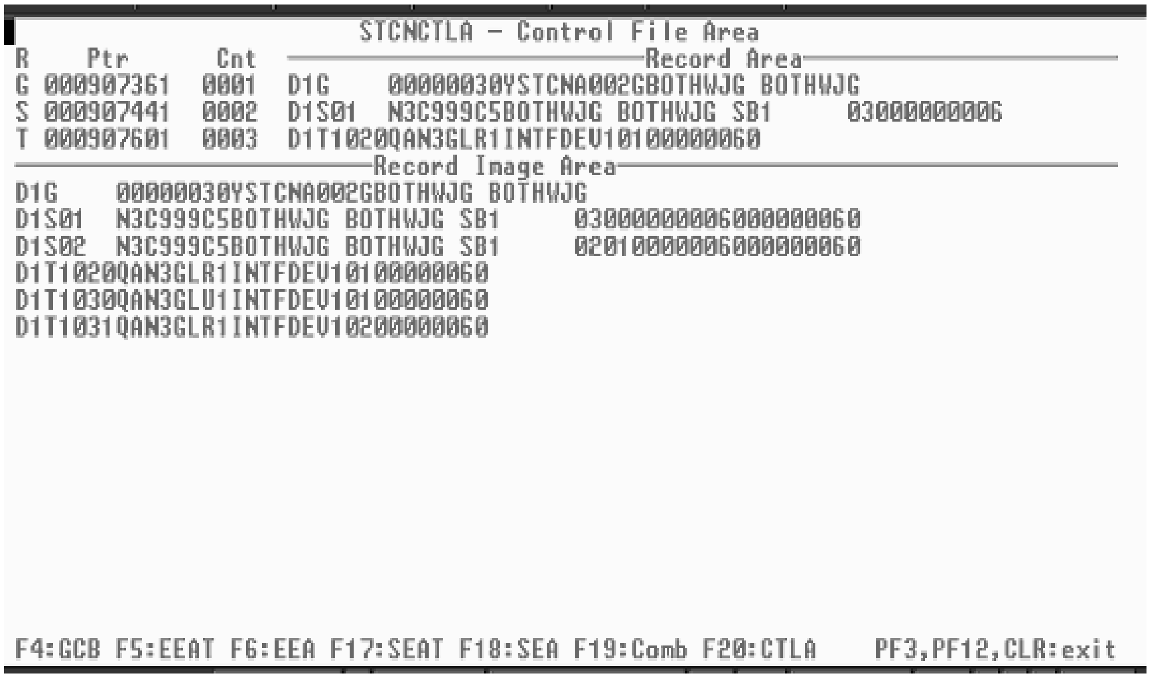


```
                         STCNEEA - E-Way ECB Areas
 ----Status----    date     time     BEP-ECB    P Tot   rc0  rc-1 rc-2 a-t l-t b-t
 1 available     10-31-01 16:40:38 000000000   0001   0000 0000 0000 000 000 000
 1 available     10-25-01 12:25:47 000000000   0001   0001 0000 0000 000 000 000
 2 initialized   10-25-01 11:34:43 000000000   0000   0000 0000 0000 000 000 000
 2 initialized   10-25-01 11:34:44 000000000   0000   0000 0000 0000 000 000 000




                         STCNSEA - Session ECB Areas
 ----Status----    date       time         FER-ECB     Task   Own Cur T En SC  Cnt
 A available     10-25-01 12:26:36.030 000000000 0001553 001 001   D1 01 0001
 A available     10-25-01 12:25:47.030 000000000 0001553 002 002   D1 02 0001




 F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (Figure 41) displays both external ADABAS Natural e*Way ECB Areas (EEAs) and back-end Natural session ECB Areas (SEAs). See the previous screen (**Figure 40 on page 46**) descriptions for the meaning of each field on the screen.

9 To view the internal memory area that contains the STCNCTL VSAM file records, press PF20 (SHIFT-F8). The screen shown in Figure 42 appears.

**Figure 42** CICS: Control File Area Screen



```
                          STCNCTLA - Control File Area
R    Ptr       Cnt    ----------------------------Record Area------------------------
G 000907361   0001   D1G     00000030YSTCNA002GBOTHWJG BOTHWJG
S 000907441   0002   D1S01   N3C999C5BOTHWJG BOTHWJG SB1      03000000006
T 000907601   0003   D1T1020QAN3GLR1INTFDEV10100000060
------------------------------Record Image Area-----------------------------------
D1G     00000030YSTCNA002GBOTHWJG BOTHWJG
D1S01   N3C999C5BOTHWJG BOTHWJG SB1      0300000000600000060
D1S02   N3C999C5BOTHWJG BOTHWJG SB1      0201000000600000060
D1T1020QAN3GLR1INTFDEV10100000060
D1T1030QAN3GLU1INTFDEV10100000060
D1T1031QAN3GLR1INTFDEV10200000060









F4:GCB F5:EEAT F6:EEA F17:SEAT F18:SEA F19:Comb F20:CTLA    PF3,PF12,CLR:exit
```

This screen (Figure 42) displays the internal memory area (STCNCTLA) that contains the records of the STCNCTL VSAM file. These records are used to control the processing of the Business Rules transaction messages coming from the external ADABAS Natural e*Ways, and also the starting up (or logging on) of back-end Natural sessions in CICS.

**Note:** *See the documentation of the STCNCTL VSAM file for a complete explanation of these record layouts.*

Chapter 3

# 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 properties for a new 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 50** and **"General Settings" on page 53** 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 e*Way Intelligent Adapter for ADABAS Natural, you must configure the Java Virtual Machine (JVM) settings. This section explains the configuration parameters in the e*Way Configuration Editor window, which control these settings.

### JNI DLL Absolute Pathname

#### Description

Specifies the absolute path name to where the JNI **.dll** (Windows) or shared library (UNIX) file is installed by the Java SDK on the Participating Host, for example:

**C:\eGate\client\bin\Jre** or **C:\jdk\jre\bin\server**

This parameter is *mandatory*.

#### Required Values

#### Required Values

A valid path name.

#### Additional Information

The JNI **.dll** or shared library file name varies, depending on the current operating system (OS). The following table lists the file names by OS:

| Operating System | Java 2 JNI .dll or Shared Library Name |
|---|---|
| Windows systems | jvm.dll |
| Solaris | libjvm.so |
| AIX | libjvm.a |

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 on different OS/platforms.

***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 SDK 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

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

#### Additional Information

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

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of "%" symbols, for example:

```
%MY_PRECLASSPATH%
```

### CLASSPATH Override

#### Description

Specifies the complete CLASSPATH variable to be used by the JVM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) is set. All necessary **.jar** and **.zip** files needed by both e*Gate and the JVM must be included. It is recommended that the **CLASSPATH Prepend** parameter be used.

#### Required Values

An absolute path or an environment 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 "%" symbols, 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 this value is set to 0 (zero), the preferred value for the initial heap size of the JVM is used.

#### Required Values

An integer from **0** to **2147483647**. This parameter is optional.

## Maximum Heap Size

### Description

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

### Required Values

An integer from **0** to **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 this value is set to 0 (zero), the default value is used.

### Required Values

An integer from **0** to **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 this value is set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

### Required Values

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

## Disable JIT

### Description

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

### Required Values

**YES** or **NO**.

## 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 the 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.

Chapter 4

# e*Way Connection Configuration

This chapter describes how to configure the parameters of the e*Way Intelligent Adapter for ADABAS Natural e*Way Connection in the e*Gate Integrator system.

## 4.1 Configuring e*Way Connections

Set the e*Way Connection properties using the e*Gate Schema Designer. This section provides procedures on how to set the properties you must configure before you can use an e*Way Connection with the ADABAS Natural e*Way.

**Note:** *For complete information on how to create and configure e*Way Connections, see the* ***e*Gate Integrator User's Guide****.*

See **Chapter 5** for detailed procedures on e*Way configuration, as part of explaining the e*Way's schema samples.

**To set necessary properties for a ADABAS Natural e*Way Connection**

1. Using the Schema Designer**,** 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 Component 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 **Natural**.
7. Set the **Event Type "get" interval** in the dialog box to the appropriate number of milliseconds.
8. From the **e*Way Connection Configuration File**, click **New**.

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

9. After you make your selection, click **OK**. The e*Way Configuration Editor window appears.
10. Use the e*Way Configuration Editor window to create a new configuration file for this e*Way Connection. Do this operation by selecting the appropriate configuration parameters available in the window (this chapter explains these parameters).
11. When you are finished, save the new configuration file and close the e*Way Configuration Editor.

The ADABAS Natural e*Way Connection configuration parameters are organized into the following sections:

- **"Connector" on page 55**
- **"CICS Gateway" on page 56**
- **"CICS Client" on page 57**
- **"Natural Settings" on page 59**
- **"Tracing" on page 61**

## 4.2 Connector

This section contains the following set of top-level parameters:

- **Type**
- **Class**
- **Property.Tag**

### Type

#### Description

Allows you to specify the type of e*Way Connection.

#### Required Values

A string. The value always defaults to **Natural** for ADABAS Natural connections.

### Class

#### Description

Allows you to specify the class name of the ADABAS Natural client connector object.

#### Required Values

A valid package name. The default is com.stc.eways.natural.NaturalClientConnector.

### Property.Tag

#### Description

Allows you to specify the data source identity. This parameter is required by the current **EBobConnectorFactory**.

#### Required Values

A valid data source package name. Accept the default.

## 4.3 CICS Gateway

This section assists in setting the following CICS Java Gateway parameters:

- **URL**
- **Port**
- **SSL KeyRing Class**
- **SSL KeyRing Password**

### URL

#### Description

Allows you to specify the URL for the remote/local Gateway to which to connect.

#### Required Values

A valid location.

### Port

#### Description

Allows you to specify the TCP/IP port to which to connect.

#### Required Values

A valid port number between **1** and **864,000**. The default is **8888.**

### SSL KeyRing Class

#### Description

Allows you to specify the full class name of the SSL KeyRing class.

#### Required Values

A valid class name.

### SSL KeyRing Password

#### Description

Allows you to specify the Password for the encrypted KeyRing class.

#### Required Values

A valid password for the encrypted KeyRing class.

## 4.4 CICS Client

The following parameters in this section assist in setting the CICS client information:

- **CICS UserId**
- **CICS Password**
- **ECI call type**
- **CICS Program**
- **CICS TransId**
- **COMMAREA Length**
- **ECI Extend Mode**
- **ECI LUW Token**
- **Message Qualifier**
- **Encoding**

### CICS UserId

#### Description

Allows you to specify the ID for the CICS user.

#### Required Values

A valid CICS user ID.

**Note:** *It is recommended that you include the CICS client user ID and password in the EWC configuration. Otherwise, you are prompted for this information, suspending all processing until valid values are entered. This recommendation is only for Windows versions of the client, because the UNIX versions generate an exception under the same conditions.*

### CICS Password

#### Description

Allows you to specify the password associated with the specified CICS user.

### Required Values

A valid CICS user password.

## ECI Call Type

### Description

Allows you to specify the ECI call type.

### Required Values

**Synchronous** or **Asynchronous**. The default is Synchronous.

## CICS Program

### Description

Allows you to specify CICS program to be run on the server.

### Required Values

A valid CICS program name in string format.

## CICS TransId

### Description

Allows you to specify the CICS TransId to be run on the server.

### Required Values

A valid CICS TransId name in string format.

## COMMAREA Length

### Description

Allows you to specify the length in bytes of the communication area (COMMAREA) passed to the ECI.

### Required Values

An integer between **1** and **32659**. The default is 1000.

## ECI Extend Mode

### Description

Allows you to specify whether to extend the extend mode.

### Required Values

**Yes** or **No**. The default is No.

### ECI LUW Token

#### Description

Allows you to specify whether the security feature-related Logical unit of work ID is only used on the same JavaGateway that created or assigned them.

#### Required Values

An integer between **0** and **1000**. The default is 0.

### Message Qualifier

#### Description

Allows you to specify whether the security feature-related ID is only used on the same JavaGateway that created or assigned them.

#### Required Values

An integer between **0** and **1000**. The default is 0.

### Encoding

#### Description

Allows you to specify the encoding type.

#### Required Values

**cp500** or **ASCII**. The default is ASCII.

## 4.5 Natural Settings

The following parameters in this section assist in setting the CICS information required to initialize a Natural Session and the Natural information:

- **CICS Program to Initialize Natural Session**
- **CICS Transaction to Initialize Natural Session**
- **CICS Program to Execute Business Rules**
- **CICS Transaction to Execute Business Rules**
- **CICS Program to Terminate Natural Session**
- **CICS Transaction to Terminate Natural Session**

### CICS Program to Initialize Natural Session

#### Description

Allows you to specify the name of the CICS program that initializes the Natural session.

#### Required Values

**STCNFEI**.

### CICS Transaction to Initialize Natural Session

#### Description

Allows you to specify the name of the CICS transaction that initializes the Natural session.

#### Required Values

**NFEI**.

### CICS Program to Execute Natural Business Rules

#### Description

Allows you to specify the name of the CICS program executes the Business Rules in the Natural session.

#### Required Values

**STCNFER**.

### CICS Transaction to Execute Natural Business Rules

#### Description

Allows you to specify the name of the CICS transaction that executes Natural Business Rules.

#### Required Values

**NFER**.

### CICS Program to Terminate Natural Session

#### Description

Allows you to specify the name of the CICS program that terminates the Natural session.

#### Required Values

**STCNFET**.

### CICS Transaction to Terminate Natural Session

#### Description

Allows you to specify the name of the CICS transaction that terminates the Natural Session.

#### Required Values

**NFET**.

## 4.6 Tracing

This section contains the following set of top-level parameters:

- **Level**
- **Filename**
- **Truncation Size**
- **Dump Offset**
- **Timing**

### Level

#### Description

Allows you to specify the level of tracing in place.

#### Required Values

**0**, **1**, **2**, or **3**; see Table 2.

**Table 2** Setting the Level Parameter

| Value | Description |
|---|---|
| 0 | **None**: No CICS Java client application tracing. |
| 1 | **Standard**: By default, displays only the first 128 bytes of any data blocks. For example, the COMMAREA, or Network flows. This trace level is equivalent to the Gateway trace set by the ctgstart -trace option. Can also set using System property “gateway.T.trace=on.” |
| 2 | **Full Debug**: By default, traces out the whole of any data blocks. The trace contains more information about CICS Transaction Gateway than the standard trace level. This trace level is equivalent to the Gateway debug trace set by the ctgstart -x option. Can also set using System property “gateway.T=on”. |
| 3 | **Exception Stacks**: Traces most Java exceptions, including exceptions which are expected during normal operation of the CICS Transaction Gateway. No other tracing is written. This trace level is equivalent to the Gateway stack trace set by the ctgstart -stack option. Can also set using System property “gateway.T.stack=on.” |

### Filename

#### Description

Allows you to specify a file location for writing the trace output. This is provided as an alternative to the default output on **stderr**.

#### Required Values

A valid file location. Long file names must be contained by quotation marks. For example, “trace output file.log”. This can also be set using System property “gateway.T.setTFile-xxx” where “xxx” is equal to a file name.

### Truncation Size

#### Description

Allows you to specify the maximum size of any data blocks to be written by the trace.

#### Required Values

An integer between **0** and **864000**. The default is **100**.

Specifying 0 causes no data blocks to be written in the trace. Leave blank to refrain from specifying the truncation size. This can also be set using System property “gateway.T.setTrancationSize=xxx” where “xxx” is equal to the size setting as an integer.

### Dump Offset

#### Description

Allows you to specify the offset from which displays of any data blocks start. If the offset is greater than the total length of data to be displayed, an offset of 0 is used.

#### Required Values

An integer between 0 and 864000. The default is 0.

This can also be set using System property “gateway.T.setDumpOffset=xxx”, where “xxx” is equal to a number indicating the offset amount.

### Timing

#### Description

Allows you to specify whether to display the time stamps in the trace.

#### Required Values

**On** or **Off**. The default is On.

This can also be set using System property “gateway.T.timing=on.”

Chapter 5

# Implementation

This chapter explains a sample schema to help you understand how to implement the e*Way Intelligent Adapter for ADABAS Natural in a production environment.

## 5.1 Implementation Overview

This section explains how to implement the ADABAS Natural 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/ewnatural/java**

This sample allows you to observe an end-to-end data-exchange scenario involving e*Gate, the e*Way, and sample interfaces. This chapter explains how to implement the this sample schema that uses the ADABAS Natural e*Way.

You can also use the procedures given in this chapter to create your own schema based on the sample provided. It is recommended that you use a combination of both methods, creating your own schema like the sample, then importing the sample into e*Gate to check your results.

***Note:*** *For complete information on how to set up an e*Gate schema, see **Creating an End-to-end Scenario with e*Gate Integrator**.*

### Before Importing or Running a Sample Schema

To import and/or run the sample schema, the ADABAS Natural e*Way must be installed, and you must also have access to an ADABAS Natural system.

### To import a sample schema

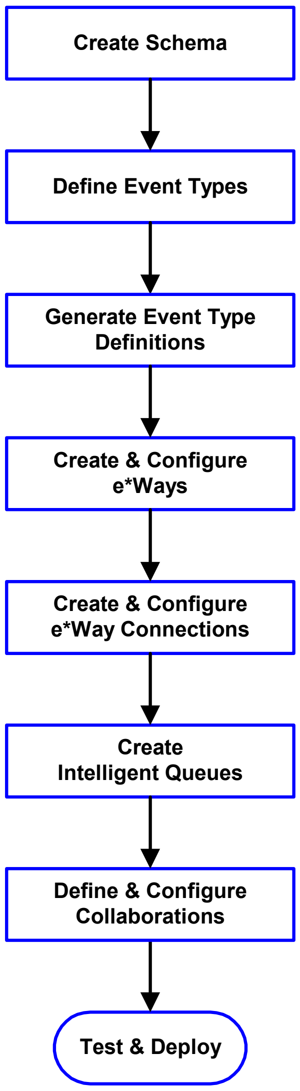
1. Copy the desired .**zip** file, for example, **NaturalClient.zip,** from the s**amples/ewnatural/java** directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
2. Start the e*Gate Schema Designer.
3. On the **Open Schema from Registry Host** dialog box, click **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 schema is installed.

### To create the sample schema

Use the following implementation sequence:


Create Schema
Define Event Types
Generate Event Type Definitions
Create & Configure e*Ways
Create & Configure e*Way Connections
Create Intelligent Queues
Define & Configure Collaborations
Test & Deploy

1 The first step is to create a new schema. The rest of these steps apply only to this schema.

2 The second step is to create and define the Event Types you are transporting and processing within the schema.

3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) you want to use in the schema.

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 Now you need to create Intelligent Queues (IQs) and IQ Managers (if necessary) to hold published Events.

7 You need to create the desired Collaboration Rules for your schema, along with their associated Business Rules.

8 Next, you need to define and configure the Collaborations between Event Types.

9 Finally, you must check and test your Schema. Once you have verified that it is working correctly, you can deploy it to your production environment.

### Chapter Organization

This chapter is set up sequentially, and you can use it as a tutorial to teach how to implement the ADABAS Natural e*Way. It is recommended that you use the steps listed previously, in that order, to implement the sample.

The chapter concludes with a section that explains how to use the e*Gate SAG wizard with the e*Way.

## 5.2 ADABAS Natural Sample Implementation

The ADABAS Natural e*Way sample schema illustrates the components to be created on Windows or UNIX. This section explains the basic structure, operation, and creation of the schema.

### 5.2.1 Schema Overview

The sample schema consists of the following components:

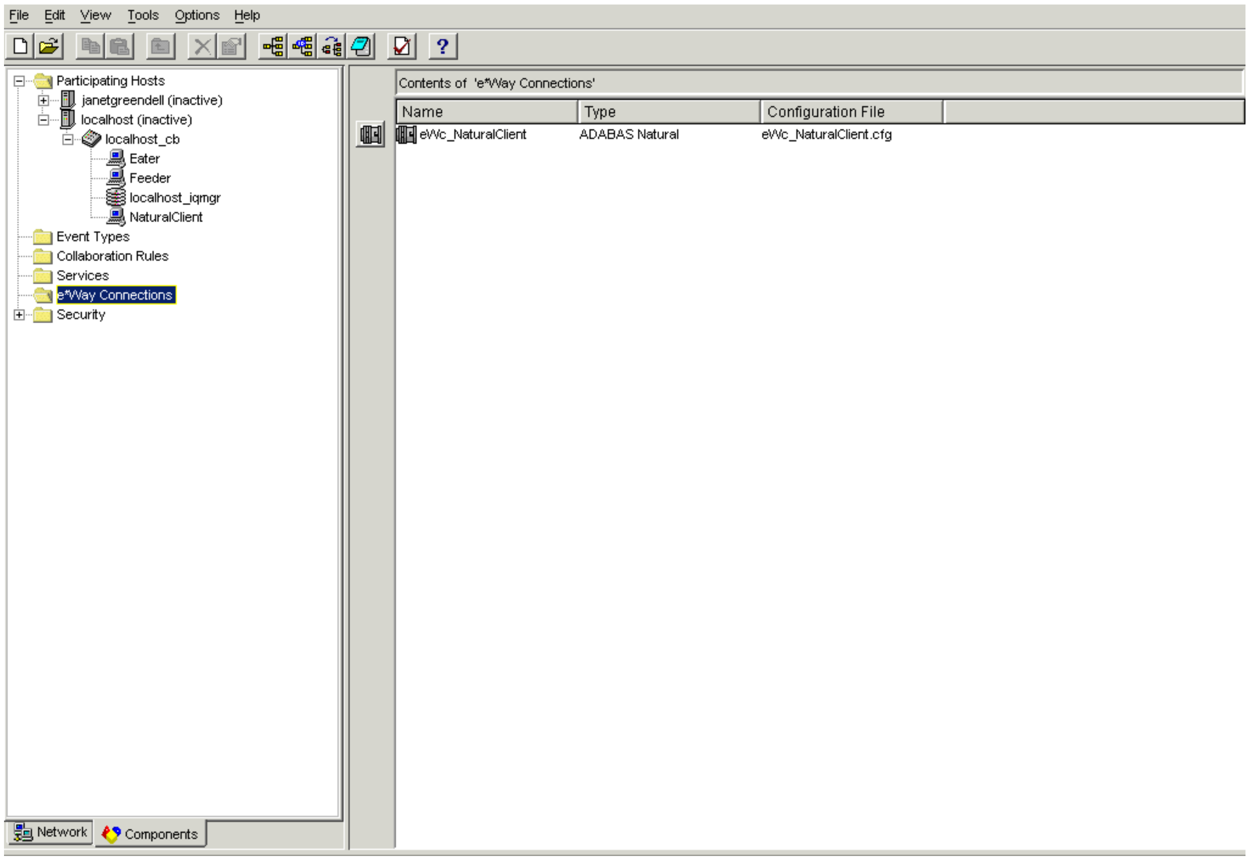
- Two file-based e*Ways (inbound/**Feeder** and outbound/**Eater**)
- One Multi-Mode e*Way (**NaturalClient**)
- One e*Way Connection (**eWc_NaturalClient**)
- One IQ Manager
- Two IQs
- Four Event Types (**GenericInEvent.ssc, GenericOutEvent.ssc, NaturalClient.xsc, NaturalTransInOut.xsc**)
- Two Collaboration Rules (**cr_PassThrough, cr_NaturalClient**)

You can create and configure all these components using the e*Gate Schema Designer.

***Note:*** *For complete information on how to use the e*Gate Schema Designer and all of its features, see the* ***e*Gate Integrator User's Guide****.*

Once the sample schema has been successfully imported, the Schema Designer appears as shown in **Figure 43 on page 66**.

**Figure 43** NaturalClient Sample Schema



| Name | Type | Configuration File |
|---|---|---|
| eWc_NaturalClient | ADABAS Natural | eWc_NaturalClient.cfg |

## 5.2.2 Schema Operation

The NaturalClient sample uses a file e*Way to send a file to the NaturalClient e*Way. The file contains a key lookup, which is then translated from ASCII to EBCDIC, before sending to the mainframe.

The mainframe receives the file, running a Natural sub-program to process the query, and returns the packet to the e*Gate, where it is translated back from EBCDIC to ASCII, before writing to a file on a local, external system.

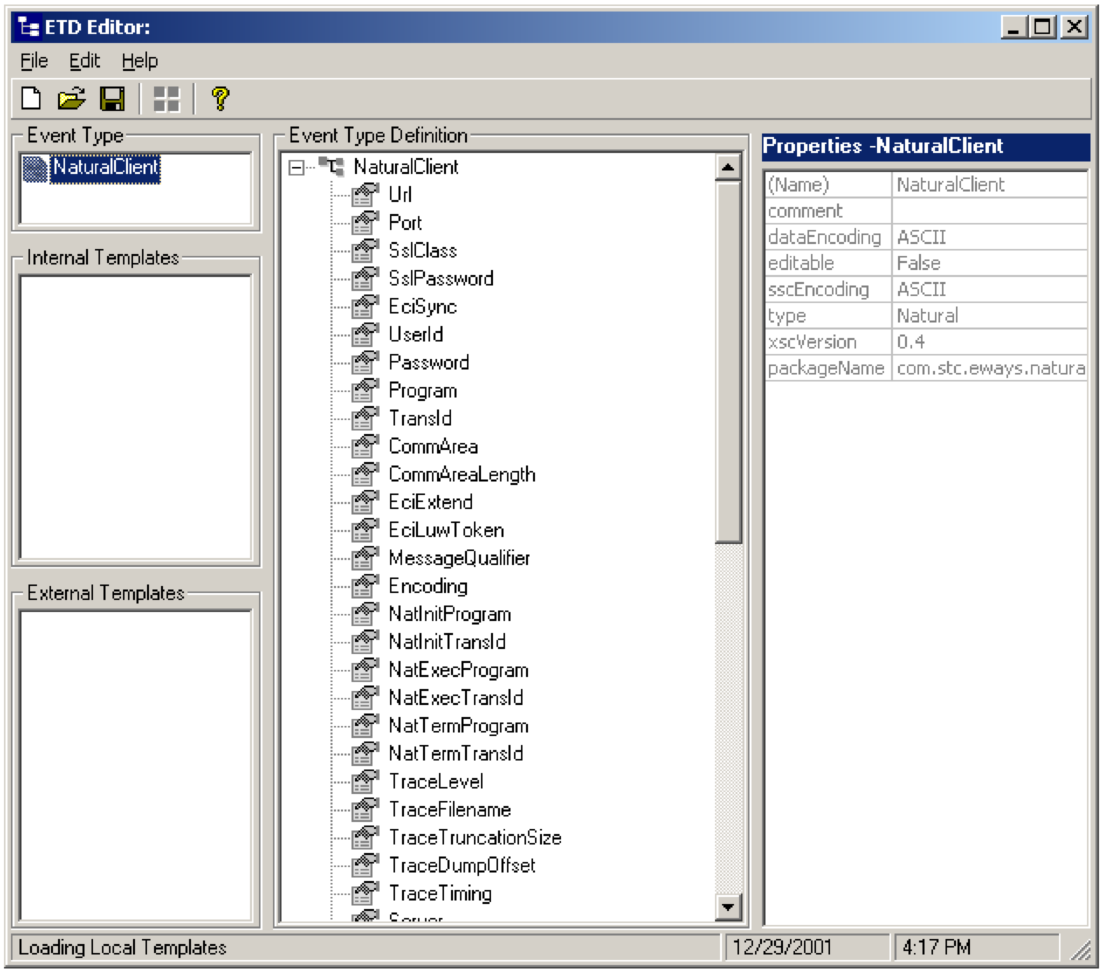
**Note:** *For more information on the e*Way and Natural sub-programs, see* **"Natural Sub-programs" on page 81**.

### 5.2.3 Creating Event Type Definitions

The **NaturalClient.xsc** file is the ADABAS Natural e*Way's basic ETD. It provides all the properties that pertain to the communication between e*Gate and the mainframe, and the definition of the COMMAREA.

Figure 44 shows this ETD, as it appears in the Schema Designer's ETD Editor Main window.

**Figure 44** ETD Editor Main Window: NaturalClient.xsc



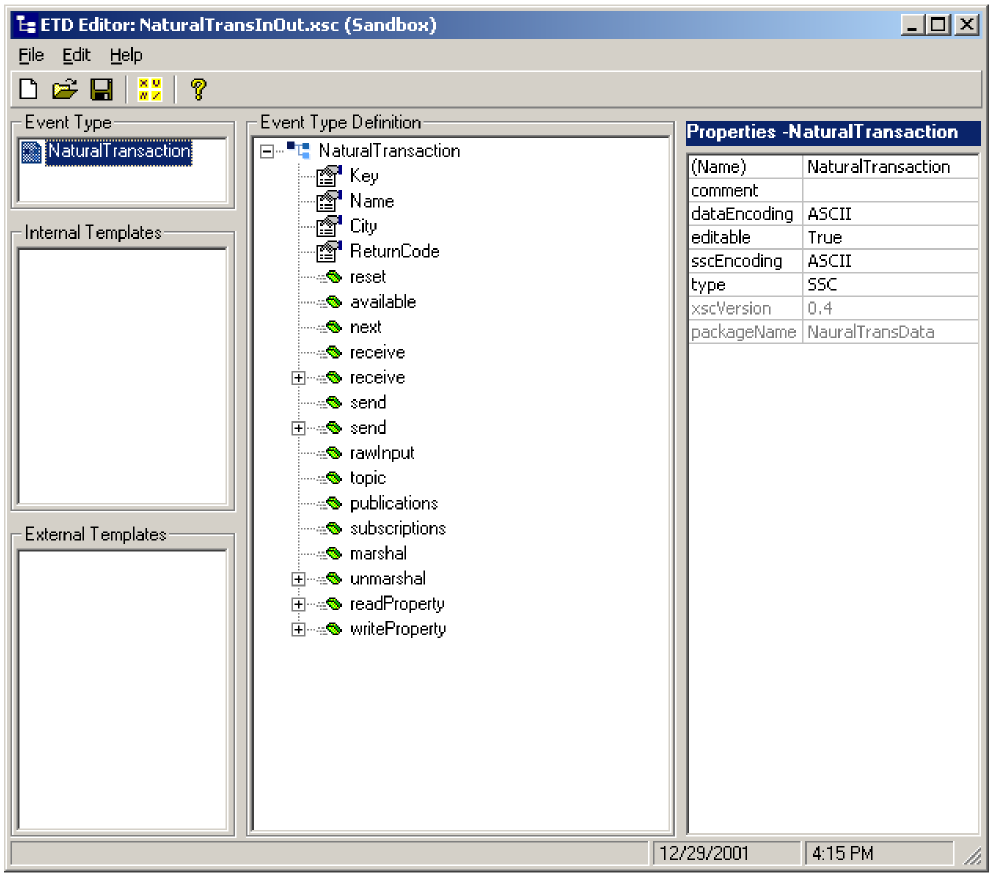
| | |
|---|---|
| (Name) | NaturalClient |
| comment | |
| dataEncoding | ASCII |
| editable | False |
| sscEncoding | ASCII |
| type | Natural |
| xscVersion | 0.4 |
| packageName | com.stc.eways.natura |

The sample also implements the **NaturalTransInOut.xsc**. This ETD has the following basic properties:

- Key
- Name
- City
- ReturnCode

The ETD was created based on the query to be performed by the external Natural program. You can use the ETD Editor's Custom ETD wizard to create a user-defined ETD that can vary based on specific needs (see Figure 45).

**Figure 45** EDT Editor Main Window: NaturalTransInOut.xsc



NaturalTransaction
- Key
- Name
- City
- ReturnCode
- reset
- available
- next
- receive
- receive
- send
- send
- rawInput
- topic
- publications
- subscriptions
- marshal
- unmarshal
- readProperty
- writeProperty

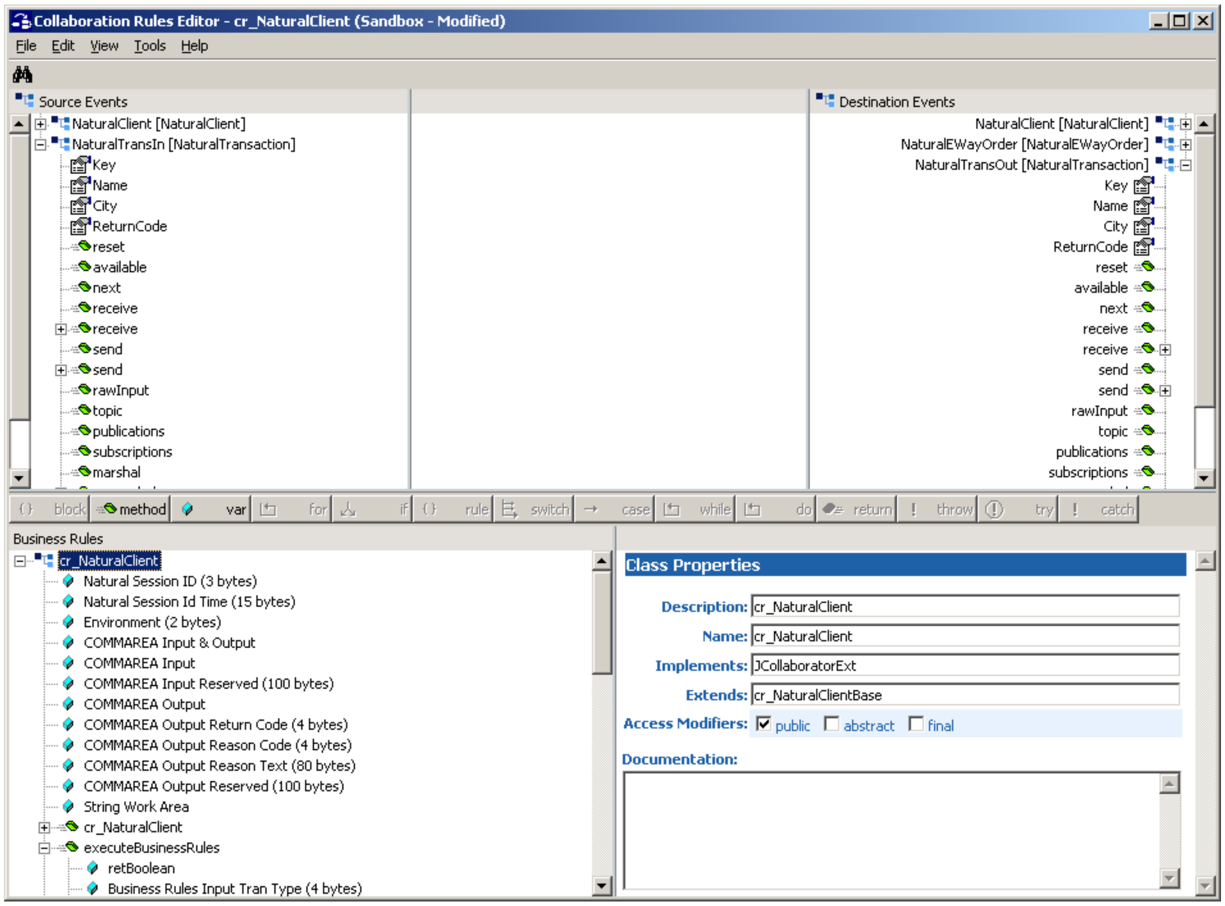
| | |
|---|---|
| (Name) | NaturalTransaction |
| comment | |
| dataEncoding | ASCII |
| editable | True |
| sscEncoding | ASCII |
| type | SSC |
| xscVersion | 0.4 |
| packageName | NauralTransData |

## 5.2.4 Creating Collaboration Rules

The Collaboration Rule performs the desired business logic. In this case, there are global variables defined, Initialization, Business, and Termination Rules, all being executed.

The Collaboration Rules Editor view appears in **Figure 46 on page 69**.

**Figure 46** Collaboration Rules Editor Main Window: cr_NaturalClient


Collaboration Rules Editor - cr_NaturalClient (Sandbox - Modified)
File Edit View Tools Help
Source Events
NaturalClient [NaturalClient]
NaturalTransIn [NaturalTransaction]
Key
Name
City
ReturnCode
reset
available
next
receive
receive
send
send
rawInput
topic
publications
subscriptions
marshal
Destination Events
NaturalClient [NaturalClient]
NaturalEWayOrder [NaturalEWayOrder]
NaturalTransOut [NaturalTransaction]
Key
Name
City
ReturnCode
reset
available
next
receive
receive
send
send
rawInput
topic
publications
subscriptions
block method var for if rule switch case while do return throw try catch
Business Rules
cr_NaturalClient
Natural Session ID (3 bytes)
Natural Session Id Time (15 bytes)
Environment (2 bytes)
COMMAREA Input & Output
COMMAREA Input
COMMAREA Input Reserved (100 bytes)
COMMAREA Output
COMMAREA Output Return Code (4 bytes)
COMMAREA Output Reason Code (4 bytes)
COMMAREA Output Reason Text (80 bytes)
COMMAREA Output Reserved (100 bytes)
String Work Area
cr_NaturalClient
executeBusinessRules
retBoolean
Business Rules Input Tran Type (4 bytes)
Class Properties
Documentation:

Description: cr_NaturalClient
Name: cr_NaturalClient
Implements: JCollaboratorExt
Extends: cr_NaturalClientBase
Access Modifiers: [✓ public] abstract final

The Java code for the Collaboration Rule shown in Figure 46 follows:

```
import com.stc.common.collabService.*;
import com.stc.jcsre.*;
import com.stc.eways.util.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import com.stc.eways.natural.*;
import NauralTransData.*;
import fsgsgf.*;


   class cr_NaturalClientBase extends JCollaboration
   {
      public cr_NaturalClientBase()
      {
         super();
      }

      com.stc.eways.natural.NaturalClient NaturalClient = null;

      public com.stc.eways.natural.NaturalClient getNaturalClient()
      {
         return this.NaturalClient;
      }
      NauralTransData.NauralTransaction NaturalTransIn = null;

      public NauralTransData.NauralTransaction getNaturalTransIn()
      {
         return this.NaturalTransIn;
      }

      fsgsgf.NaturalEWayOrder NaturalEWayOrder = null;

      public fsgsgf.NaturalEWayOrder getNaturalEWayOrder()
      {
         return this.NaturalEWayOrder;
      }

      NauralTransData.NauralTransaction NaturalTransOut = null;

      public NauralTransData.NauralTransaction getNaturalTransOut()
      {
         return this.NaturalTransOut;
      }

      public void resetData() throws CollabConnException, CollabDataE
xception
      {
         this.NaturalClient = (com.stc.eways.natural.NaturalClient) t
his.reset((ETD)this.getNaturalClient());
         this.NaturalTransIn = (NauralTransData.NauralTransaction) th
is.reset((ETD)this.getNaturalTransIn());
         this.NaturalEWayOrder = (fsgsgf.NaturalEWayOrder) this.reset
((ETD)this.getNaturalEWayOrder());
         this.NaturalTransOut = (NauralTransData.NauralTransaction) t
his.reset((ETD)this.getNaturalTransOut());
      }
      public void createInstances() throws CollabConnException
      {
         this.NaturalClient = (com.stc.eways.natural.NaturalClient) t
his.newInstance("com.stc.eways.natural.NaturalClient", "NaturalClient
", ETD.IN_OUT_MODE);
         this.NaturalTransIn = (NauralTransData.NauralTransaction) th
is.newInstance("NauralTransData.NauralTransaction", "NaturalTransIn",
 ETD.IN_MODE);
         this.NaturalEWayOrder = (fsgsgf.NaturalEWayOrder) this.newIn
stance("fsgsgf.NaturalEWayOrder", "NaturalEWayOrder", ETD.OUT_MODE);
         this.NaturalTransOut = (NauralTransData.NauralTransaction) t
his.newInstance("NauralTransData.NauralTransaction", "NaturalTransOut
", ETD.OUT_MODE);
      }
   }



   public class cr_NaturalClient extends cr_NaturalClientBase impleme
nts JCollaboratorExt
   {
```

```
=====================================================================

These two variables must be defined as public and available to
Initialize, Business Rules, and Termination:

      // IMPORTANT NOTE: NaturalSessionId must be initialized to
         3 spaces
      // IMPORTANT NOTE: NaturalSessionIdTime must be initialized to
         15 spaces

      public String NaturalSessionId = "   ";
      public String NaturalSessionIdTime = "               ";

=====================================================================

These variables can be defined locally to Initialize, Business Rules
and Termination. I decided to define them here
because the same area can be used in all three places.


This String is a 100 Byte reserved area in the input section of the
COMMAREA

      public String CAInReserved = "
                                                                   ";

This String is a 100 Byte reserved area in the output section of the
COMMAREA.

      public String CAOutReserved = "
                                                                    "
;

The Return Code needs to be a 4 byte String

      public String CAOutReturnCode = "    ";

The Reason Code needs to be a 4 byte String

      public String CAOutReasonCode = "    ";

The Reason Text needs to be a 80 byte String

      public String CAOutReasonText = "
                                                 ";

      public String CAInEnvironment = "  ";
      public String CAInputOutput;
      public String CAInput;
      public String CAOutput;
      public String StringWork;

=====================================================================

      public  cr_NaturalClient()
      {
         super();
      }

=====================================================================

      public boolean executeBusinessRules() throws Exception
      {
         boolean retBoolean = true;
```

```
          String CAInTranType = "    ";
          String CAInGuid = "                                    ";
          String CAPayload;
          String NaturalTranOut;

This is part of the actual Collaborition. In this example we’re
setting the transaction to 1020 and converting it from ASCII
to EBCDIC.

CAInTranType = "1020";

CAInTranType = new String (CAInTranType.getBytes("cp500"), "ISO-8859-
1");

This is part of the actual collaborition too. In this example
we've set the environment to QA and converted it from ASCII to EBCDIC

CAInEnvironment = "QA";

CAInEnvironment = new String (CAInEnvironment.getBytes("cp500"), "ISO
-8859-1");

This is part of the actual Collaboration also. In this example
we've constructed payload and converted it from ASCII to EBCDIC.

CAPayload = getNaturalTransIn().getKey() + getNaturalTransIn().getNam
e() + getNaturalTransIn().getCity() + getNaturalTransIn().getReturnCo
de();

CAPayload = new String (CAPayload.getBytes("cp500"), "ISO-8859-1");

This established a GUID for the Transaction we're about to send to
Natural.

CAInGuid = getNaturalClient().getGuidString();

The COMMAREA for Business Rules needs to be constructed as follows.

CAInput = NaturalSessionId + NaturalSessionIdTime + CAInTranType + CA
InEnvironment + CAInGuid + CAInReserved;

CAOutput = CAOutReturnCode + CAOutReasonCode + CAOutReasonText + CAOu
tReserved;

CAInputOutput = CAInput + CAOutput + CAPayload;

This methold sets up the length of the COMMAREA that we’re going to
send for a Natural Business Rule.

getNaturalClient().setCommAreaLength(CAInputOutput.length());

This method sends the COMMAREA that was just constructed to CICS to e
xecute a NATRUAL Business Rule

getNaturalClient().setCommArea(CAInputOutput.getBytes());

This method gets the list of available CICS Servers

getNaturalClient().getServerList(2);

These two methods set the CICS program name and Transaction for
Business Rules. These values would normally be picked from the
configuration file.
```

```
getNaturalClient().setProgram("STCNFER");

getNaturalClient().setTransId("NFER");

This Method calls the CICS Transaction to execute a Natural
Business Rule.

getNaturalClient().execute();

Obtain the returned payload back from Natural and convert it from
EBCDIC to ASCII.

NaturalTranOut  = getNaturalClient().getCommAreaString(350,50);

NaturalTranOut = new String (NaturalTranOut.getBytes(), "cp500");


return retBoolean;
      }


======================================================================

      public void userInitialize()
      {

The COMMAREA for Initialization needs to be constructed as follows.
The lengths have been established when these Strings were initially
defined.

    CAInput = CAInReserved;


CAOutput = CAOutReturnCode + CAOutReasonCode + CAOutReasonText + Natu
ralSessionId + NaturalSessionIdTime + CAOutReserved;

    CAInputOutput = CAInput + CAOutput;

This methold sets up the length of the COMMAREA that we’re going to
send at Natural Initialization.

getNaturalClient().setCommAreaLength(CAInputOutput.length());

This method sends the COMMAREA that was just constructed to CICS to
Initialize a Natural session.

getNaturalClient().setCommArea(CAInputOutput.getBytes());

This method gets the list of available CICS Servers.

getNaturalClient().getServerList(2);

These two methods set the CICS program name and Transaction for
Initialization. These values would normally be picked from the
configuration file.

getNaturalClient().setProgram("STCNFEI");

getNaturalClient().setTransId("NFEI");
         System.err.println ("About to initialize Natural Session und
er CICS");
```

```
This Method executes the CICS Transaction to Initialize a Natural
session.

getNaturalClient().execute();

This obtains the Natural session ID and session ID time which will be
used in Business Rules and Termination (this is required).

NaturalSessionId = getNaturalClient().getCommAreaString(188,3);

NaturalSessionIdTime = getNaturalClient().getCommAreaString(191,15);

      }

======================================================================


      public void userTerminate()
      {

The COMMAREA for Termination needs to be constructed as follows.

CAInput = NaturalSessionId + NaturalSessionIdTime + CAInEnvironment +
 CAInReserved;

CAOutput = CAOutReturnCode + CAOutReasonCode + CAOutReasonText + CAOu
tReserved;

CAInputOutput = CAInput + CAOutput;

This methold sets up the length of the COMMAREA that were going to
send at Natural Terminmation.

getNaturalClient().setCommAreaLength(CAInputOutput.length());

This method sends the COMMAREA that was just constructed to CICS to
Initialize a Natural session.

getNaturalClient().setCommArea(CAInputOutput.getBytes());

This method gets the list of available CICS Servers

getNaturalClient().getServerList(2);

These two methods set the CICS program name and Transaction for
Termination. These values would normally be picked from the
configuration file.

getNaturalClient().setProgram("STCNFET");

getNaturalClient().setTransId("NFET");

This Method executes the CICS Transaction to Terminate a Natural
session.

getNaturalClient().execute();
      }
   }


======================================================================
```

### Sample Input Data

For the sample schema, the following was passed in as a 50 byte file vertical bars have been added to show the contained byte area.

```
|50005500|                    |                    |  |
```

The first 8 bytes are the key, the next 20 are for the name, the next 20 are for the city, and the final 2 are for the return code.

### Sample Natural Program

The sample is designed to function with a sample Natural program, such as the following program:

```
DEFINE DATA
PARAMETER
* PAYLOAD area from e*Gate
1 #STCP-PARM-DATA  (A50)
END-DEFINE
*
RESET #STCL-PARM-DATA (A50)
MOVE #STCP-PARM-DATA TO #STCL-PARM-DATA
REDEFINE #STCL-PARM-DATA ( #STCL-KEY    (A8)
                           #STCL-NAME   (A20)
                           #STCL-CITY   (A20)
                           #STCL-RET    (A2))
IF #STCL-KEY = "50005500"
  DO
    MOVE "BLOND" TO #STCL-NAME
    MOVE "ST-ETIENNE" TO #STCL-CITY
    MOVE "OK" TO #STCL-RET
    MOVE #STCL-PARM-DATA TO #STCP-PARM-DATA
  DOEND
*
END
```

## 5.2.5 Running the Schema

**To run the NaturalClient schema**

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

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

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

2. Exit from the command line prompt, and start the Schema Manager GUI.

3. When prompted, specify the ***hostname*** which contains the Control Broker you started in Step 1 above.

4. Select the NaturalClient schema.

5. After you verify that the Control Broker is connected (the message in the **Control** tab of the console indicates the command *succeeded* and the status as *up*), highlight the IQ Manager, ***hostname***_igmgr, then click on the right button of the mouse, and select **Start**.

6. Select each of the e*Ways, right-click the mouse, and select **Start**.

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

**Note:** *While the schema is running, opening the destination file, can cause errors.*

## 5.3 SAG Wizard Operation

This section explains how to convert .**sag** files into ETDs usable by the ADABAS Natural e*Way. You can use the e*Gate Schema Designer's SAG wizard to do this operation.

### 5.3.1 Getting Started

Before you can use the SAG wizard to convert .**sag** files into ETDs, you must first unload the source code for the Natural Local Data Areas (LDAs) and/or Parameter Data Areas (PDAs) that their respective Natural applications are using.

#### Creation of .sag Files

Natural source-code objects of the type LDA and PDA are the input used to create .**sag** files. The .**sag** files are created by the Natural Unload utility on an OS/390 mainframe. Using this utility, you must unload each LDA or PDA Natural source-code object to an MVS Physical Sequential (PS) file with the extension .**sag**.

The Natural Unload utility creates the PS files and gives them the appropriate extension. You must ensure a one-to-one correspondence between each LDA or PDA object and its corresponding .**sag** file.

**Note:** *The Natural Unload utility is a Software AG Natural product. This utility is also referred to as the SYSTRANS utility.*

#### Converting .sag Files: The SAG Wizard

You can use the e*Gate SAG wizard to convert .**sag** files to e*Gate ETDs. To do so, you must first use the Natural Unload utility to unload one source-code object per each PS file, as described earlier. When this operation is done, you must then FTP each of the PS files, in the binary mode, to a Windows system workstation running the e*Gate Schema Designer.

**Note:** *Unload the Natural LDAs and PDAs via the Natural Unload utility with the EBCDIC-to-ASCII option set **Y** (Yes).*

Once these PS files reside on a machine available to the Schema Designer, you are ready to use the SAG wizard to convert them to ETDs. This wizard is a part of the Schema Designer's ETD Editor feature.

The SAG wizard generates an e*Gate-compatible ETD from each PS file containing an LDA or PDA object. The wizard does not add or delete anything from the source objects. Instead, it simply converts them to a file format that e*Gate and the ADABAS Natural e*Way can use.

You must use the SAG wizard convert .**sag** files in this way whenever you need to exchange data between a Natural sub-program and the ADABAS Natural e*Way. The ETDs generated by the SAG wizard are used by the e*Way only to exchange data between e*Gate and the Natural sub-program that is being called.

## 5.3.2 Using the SAG Wizard

Once you have unloaded the source objects and converted them to .**sag** files as explained earlier, you are ready to use the SAG wizard.

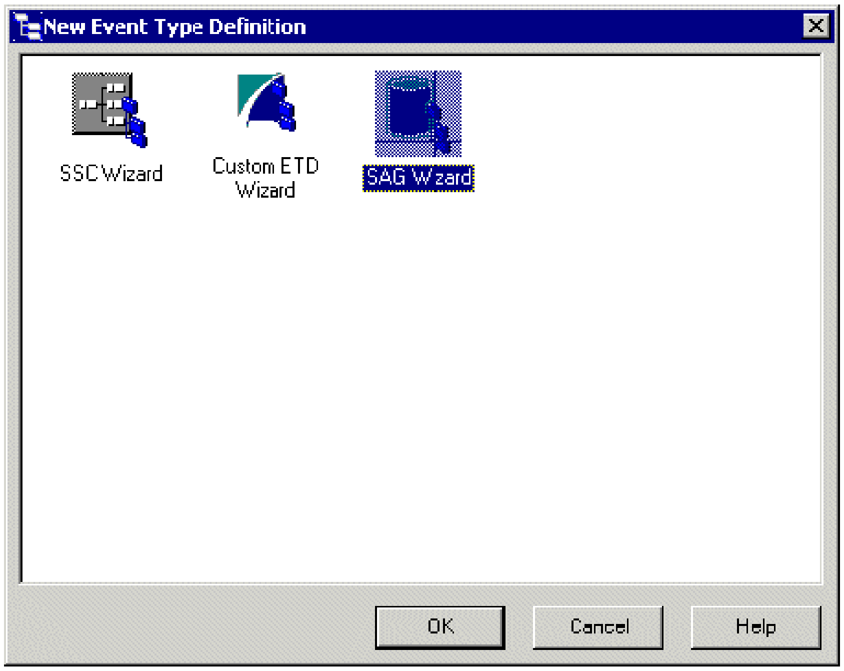
**To use the SAG wizard to convert an .sag file to an ETD**

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 SAG wizard, click **New** on the ETD Editor's **File** menu.

The New Event Type Definitions dialog box appears, displaying all installed ETD wizards (see the example in Figure 47).

**Figure 47** New Event Type Definition Dialog Box

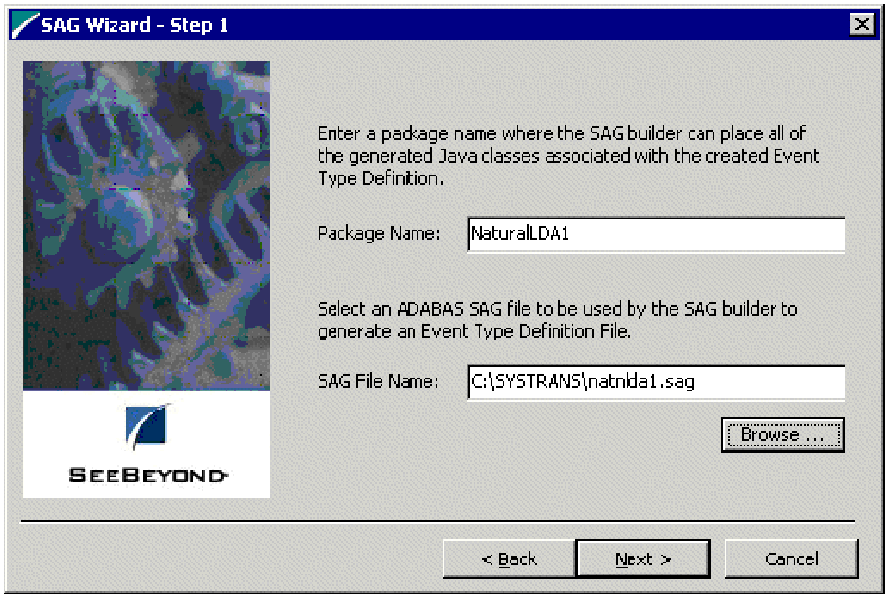


3 Double-click the **SAG Wizard** icon.

4 Review the **SAG Wizard - Introduction** dialog box, then click **Next**. This dialog box gives you brief instructions on how to use the wizard.

The **SAG Wizard - Step 1** dialog box appears (see Figure 48).

**Figure 48** SAG Wizard - Step 1 Dialog Box



Enter a package name where the SAG builder can place all of the generated Java classes associated with the created Event Type Definition.

Package Name: NaturalLDA1

Select an ADABAS SAG file to be used by the SAG builder to generate an Event Type Definition File.

SAG File Name: C:\SYSTRANS\natnlda1.sag

5 In the **SAG Wizard - Step 1** do the following actions:

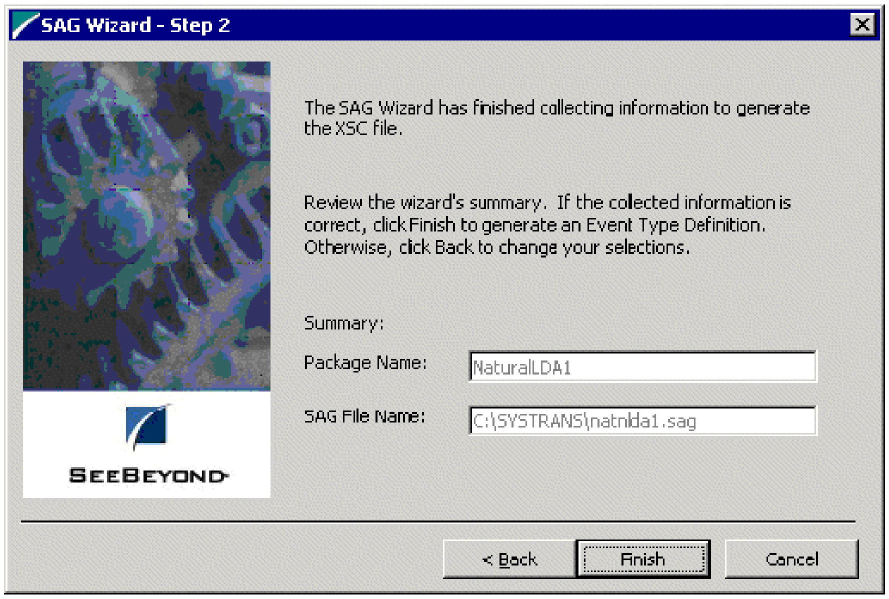
- Enter the desired package name for the container in which the wizard places the generated Java classes.
- Enter the desired **.sag** file name for the ETD. Be sure to include the full path location.

***Note:*** *Be sure to observe the required Java naming rules in these entries.*

6 Click **Next** to continue.

The **SAG Wizard - Step 2** dialog box appears (see Figure 49).

**Figure 49** SAG Wizard- Step 2 Dialog Box



The SAG 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. Otherwise, click Back to change your selections.

Summary:

Package Name: NaturalLDA1

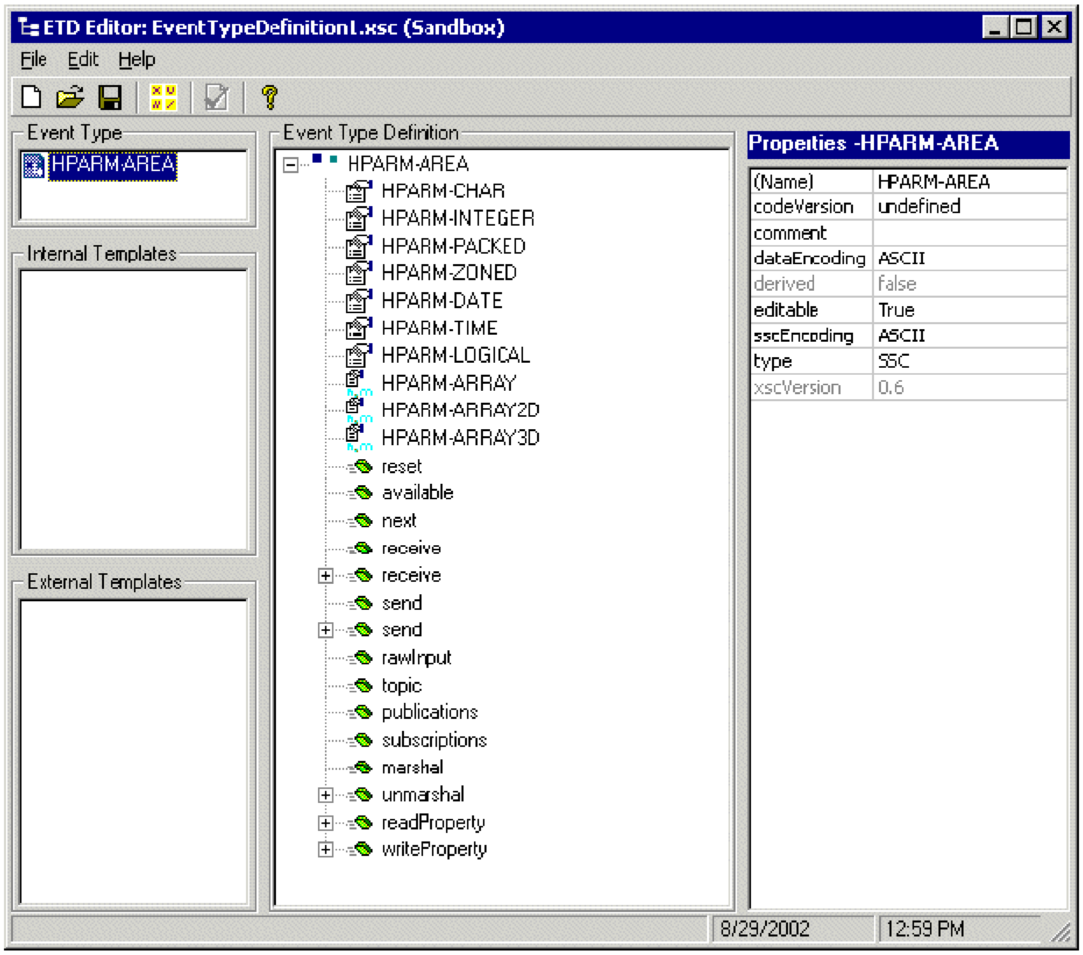
SAG File Name: C:\SYSTRANS\natnlda1.sag

7 Use the **SAG Wizard - Step 2** 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.

8 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 as shown in **Figure 50 on page 80**.

**Figure 50** ETD Editor Main Window: Sample .sag File



| | |
|---|---|
| (Name) | HPARM-AREA |
| codeVersion | undefined |
| comment | |
| dataEncoding | ASCII |
| derived | false |
| editable | True |
| sscEncoding | ASCII |
| type | SSC |
| xscVersion | 0.6 |

9 Click **Compile and Save** on the **File** menu to compile and save the ETD. In the resulting **Save** dialog box, you can enter your desired name. This is your new ETD (**.xsc**) file based on the input **.sag** file.

10 Close the ETD Editor and exit back to the Schema Designer. Be sure you promote the new file to run time.

## Example

The following text shows the source code for the Natural LDA NATNLDA2:

```
DEFINE DATA LOCAL
1 #PARM-AREA(A250)
* This is a comment line
1 REDEFINE #PARM-AREA
2 #PARM-CHAR(A36)
2 REDEFINE #PARM-CHAR
3 #PARM-CHAR2-ARRAY(A2/1:18)
2 #PARM-INTEGER(I4)
2 #PARM-PACKED(P7)
2 #PARM-ZONED(N7)
2 #PARM-DATE(D)
2 #PARM-TIME(T)
2 #PARM-LOGICAL(L)
```

```
2 #PARM-ARRAY(A3/1:2)
2 #PARM-ARRAY2D(A5/1:2,1:2)
2 #PARM-ARRAY3D(A7/1:2,1:2,1:2)
END-DEFINE
```

The following text shows an example of a .**sag** file based on the Natural LDA NATNLDA2:

```
*H**ANAT310320011010852126 MVS/ESA              0AE B
*C**                            INTFDEV1NATNLDA2
L                 *D01NAT3103L INTFDEV1NATNLDA2
MSCWT   MSCWT   TERM0235                   *D02
2001092109313692001092109313690000000683
*D03MVS/ESA TSO   NATTSO
*D04
*S****DF      0000A 250 1#PARM-AREA
*S****C       0000* THIS IS A COMMENT LINE
*S****DRR     0000    R1#PARM-AREA
*S****DFR     0000A  36 2#PARM-CHAR
*S****DRR     0000    R2#PARM-CHAR
*S****DFRI1   0000A   2 3#PARM-CHAR2-ARRAY       (18)
*S****DFR     0000I   4 2#PARM-INTEGER
*S****DFR     0000P   7 2#PARM-PACKED
*S****DFR     0000N   7 2#PARM-ZONED
*S****DFR     0000D     2#PARM-DATE
*S****DFR     0000T     2#PARM-TIME
*S****DFR     0000L     2#PARM-LOGICAL
*S****DFRI1   0000A   3 2#PARM-ARRAY            (2)
*S****DFRI2   0000A   5 2#PARM-ARRAY2D          (2,2)
*S****DFRI3   0000A   7 2#PARM-ARRAY3D         (2,2,2)
*E
```

## 5.4 Natural Sub-programs

This section explains how e*Gate communicates with Natural sub-programs.

### 5.4.1 Communication With e*Gate: Overview

Going from e*Gate to CICs, the ADABAS Natural e*Way in e*Gate communicates with Natural sub-programs via the 3GL call interface, via several components and programs. Upon completion of the Natural sub-programs, data is returned to e*Gate and the ADABAS Natural e*Way through the same sequence of events, except in reverse.

## 5.4.2 Communication With e*Gate: Basic Steps

The steps in the communication from e*Gate to CICS-OS\390 happen as follows:

### Within e*Gate

- The ADABAS Natural e*Way communicates with a SeeBeyond front-end component running inside CICS.

### Within CICS on OS/390

- The SeeBeyond CICS front-end component runs a CICS Natural session.
- This session runs a back-end COBOL program within CICS.
- This program calls Natural sub-programs via the 3GL call interface.
- Upon the completion of the sub-programs, control is returned to the SeeBeyond back-end program (running under the Natural session).
- Data is also returned to the SeeBeyond program, along with control.

***Note:*** *The Natural sub-programs can optionally call a Natural main program, as long as control is returned back to the Natural sub-programs (that is, FETCH RETURN).*

### Return Data

The steps in the return of data from CICS-OS\390 back to e*Gate happen in the reverse order as shown in the previous lists.

Upon completion of the Natural sub-programs, the control, along with any return data, is returned to the SeeBeyond back-end program running under the Natural session. This session then sends the return data to the SeeBeyond front end-component, which then returns the data to e*Gate and the ADABAS Natural e*Way.

Chapter 6

# CICS COBOL Function Prototypes

This chapter explains the CICS COBOL function prototypes that the e*Way Intelligent Adapter for ADABAS Natural supports, along with an explanation of how it supports each one.

## 6.1 Function Prototypes

The ADABAS Natural e*Way supports the following CICS COBOL function prototypes:

**CLOSE** on page 83

**OPEN** on page 84

**RECEIVE** on page 85

**SEND** on page 86

### CLOSE

#### Description

**CLOSE** shuts down the socket connection with the MUX server e*Way and frees any resources associated with it.

#### Syntax

```
call "MUXNATS" using
      MUXNAT-handle
      MUXNAT-errno
      MUXNAT-retcode.
```

#### Sample Working Storage Definitions

```
01  MUXNAT-handle    pic s9(8)   binary.
01  MUXNAT-errno     pic  9(8)   binary value 0.
01  MUXNAT-retcode   pic s9(8)   binary value +0.
```

#### Parameters Set by the Application

**MUXNAT-handle**

A 4-byte binary number containing the socket number returned by the OPEN.

**Return Value**
Unchanged.

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Return Value**
If MUXNAT-retcode is negative, this contains an error number. See **Appendix A**.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Return Value**
Negative value signifies an error.

## OPEN

### Description

**OPEN** creates a socket connection to the MUX server e*Way running on the specified remote host and TCP/IP port. This socket connection is defined by a unique identifier, or "handle", that is returned by the OPEN.

***Note:*** *This function prototype allows multiple connections to be opened and maintained by a single CICS application to a single or multiple MUX server e*Ways.*

### Syntax

```
call "MUXNAT" using
    MUXNAT-handle
    MUXNAT-remote-host
    MUXNAT-remote-port
    MUXNAT-errno
    MUXNAT-retcode.
```

### Sample working storage definitions

```
01  MUXNAT-handle       pic  s9(8)  binary value +0.
01  MUXNAT-remote-host  pic  x(24)  value 'remote.host.name'.
01  MUXNAT-remote-port  pic   9(8)  binary value 26051.
01  MUXNAT-errno        pic   9(8)  binary value 0.
01  MUXNAT-retcode      pic  s9(8)  binary value +0.
```

### Parameters set by the application

**MUXNAT-handle**
A 4-byte binary number, initialized to zero.

**Return Value**
TCP/IP socket number for the established connection.

**MUXNAT-remote-host**
A 24-byte character field, containing the DNS name of the remote host on which the MUX server e*Way is running.

**MUXNAT-remote-port**
A 4-byte binary number, containing the TCP/IP port number to which the MUX server e*Way is listening. The default is **26051**.

**Return Value**
Unchanged.

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Return Value**
See **Appendix A**.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Return Value**
Negative value signifies error.

## RECEIVE

### Description

**RECEIVE** receives a message or block of data from the MUX server e*Way. The call waits the specified time (expressed in hundredths of seconds) for a message to arrive on the socket connection identified by the passed handle.

### Syntax

```
call "MUXNATR" using
    MUXNAT-handle
    MUXNAT-returnmsg-len
    MUXNAT-returnmsg
    MUXNAT-hsecs-to-wait
    MUXNAT-errno
    MUXNAT-retcode.
```

### Sample Working Storage Definitions

```
01  MUXNAT-handle           pic s9(8)     binary.
01  MUXNAT-returnmsg-len    pic  9(8)     binary.
01  MUXNAT-returnmsg        pic x(32727)  value spaces.
01  MUXNAT-hsecs-to-wait    pic  9(8)     binary value 100.
01  MUXNAT-errno            pic  9(8)     binary value 0.
01  MUXNAT-retcode          pic s9(8)     binary value +0.
```

### Parameters Set by the Application

**MUXNAT-handle**
A 4-byte binary number containing the socket number returned by the OPEN.

**Return Value**
Unchanged.

**MUXNAT-returnmsg-len**
A 4-byte binary number, initialized to zero.

**Return Value**
The length, in bytes, of the data received from the MUX server e*Way.

**MUXNAT-returnmsg**
A 32727-byte character field.

**Return Value**
The data received from the MUX server e*Way.

**MUXNAT-hsecs-to-wait**
A 4-byte binary number, representing the hundredths of seconds to wait for a response from e*Gate.

**Return Value**
Unchanged.

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Return Value**
If MUXNAT-retcode is negative, this contains an error number. See **Appendix A**.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Return Value**
Negative value signifies an error.

## SEND

### Description

**SEND** sends a message or block of data to the MUX server e*Way.

### Syntax

```
call "MUXNATS" using
    MUXNAT-handle
    MUXNAT-message-len
    MUXNAT-message
    MUXNAT-secs-to-expire
    MUXNAT-errno
    MUXNAT-retcode.
```

### Sample Working Storage Definitions

```
01  MUXNAT-handle           pic   s9          binary.
01  MUXNAT-message-leng     pic    9(8)       binary.
```

```
    01  MUXNAT-message          pic   x(32703)   value spaces.
    01  MUXNAT-secs-to-expire   pic    9(8)      binary.
    01  MUXNAT-errno            pic    9(8)      binary value 0.
    01  MUXNAT-retcode          pic   s9(8)      binary value +0.
```

### Parameters Set by the Application

**MUXNAT-handle**

A 4-byte binary number containing the socket number returned by the OPEN.

**Return Value**

Unchanged.

**MUXNAT-message-len**

A 4-byte binary number containing the length, in bytes, of the message to be sent to the MUX server e*Way. The maximum size is 32K - 40 bytes, or 32727 bytes.

**Return Value**

Unchanged.

**MUXNAT-message**

A 32727-byte character field containing the actual data to be sent to the MUX server e*Way. The contents of this field are transmitted without a conversion of any kind.

**Return Value**

Unchanged.

**MUXNAT-secs-to-expire**

A 4-byte binary number, initialized to zero. For future use.

**Return Value**

Unchanged.

**MUXNAT-errno**

A 4-byte binary number, initialized to zero.

**Return Value**

If MUXNAT-retcode is negative, this contains an error number. See **Appendix A**.

**MUXNAT-retcode**

A 4-byte signed binary number, initialized to zero.

**Return Value**

Negative value signifies an error.

Chapter 7

# Using MUXNAT APIs

This chapter explains how to use the MUXNAT Application Programming Interfaces (APIs) with the e*Way Intelligent Adapter for ADABAS Natural.

## 7.1 MUXNAT APIs: Overview

MUXNAT APIs are used by Natural programs running on OS/390 (MVS) systems to perform calls to the MUXAPI modules, which communicate with the e*Gate MUX e*Way. The MUXNAT API works for Natural programs running in both CICS and batch environments.

The rest of this chapter provides a brief explanation of these APIs and how to use them.

## 7.2 Using MUXNAT APIs

The following code demonstrates a sample set of actions:

- Call **MUXNAT** with the appropriate parameters to establish a connection to the multiplexer e*Way.
- Call **MUXNATS** to SEND data to e*Gate, passing the data and its length as specified in the parameter list.
- Call **MUXNATR** to RECEIVE data from e*Gate; the length of the data received is returned by the API. Use the MUXNAT-hsecs-to-wait parameter to cause the execution to pause long enough for e*Gate to process and return the data.
- Repeat the SEND and RECEIVE as desired to continue passing and receiving data.
- Call **MUXNATC** to close the connection.

***Note:*** *Once the connection has been opened successfully, if any of the subsequent functions fails, the connection must be closed before continuing.*

The following Natural “client” program illustrates a simple Open-Send-Receive-Close scenario, in which a seventeen character text message (hard-coded in working storage in this example), is sent to the e*Gate “server,” and waits one second to receive a response:

```
* ===================================================================*
*  VARIABLES USED FOR THE MUXNAT FUNCTION CALLS                      *
* ===================================================================*
 RESET MUXNAT-HANDLE (B4)
* MOVE IN YOUR DNS NAME HERE
 MOVE 'YOUR.DNS.NAME' TO MUXNAT-REMOTE-HOST (A24)
* DEFAULT PORT:  YOU MAY NEED TO CHANGE THIS PER YOUR INSTALLATION
 MOVE 26051 TO MUXNAT-REMOTE-PORT (B4)
 RESET MUXNAT-MESSAGE-LEN (B4)
 RESET MUXNAT-HSECS-FOR-ACK (B4)
 RESET MUXNAT-RETURNMSG-LEN (B4)
 MOVE 100 TO MUXNAT-HSECS-TO-WAIT (B4)
 MOVE 100 TO MUXNAT-HSECS-FOR-ACK
 RESET MUXNAT-ERRNO (B4)
 REDEFINE MUXNAT-ERRNO (MUXNAT-ERRNO-I (I4))
 RESET MUXNAT-RETCODE (B4)
 REDEFINE MUXNAT-RETCODE (MUXNAT-RETCODE-I (I4))
* ===================================================================*
*  MISC                                                              *
* ===================================================================*
 MOVE 'HELLO FROM MUXCLI' TO  TEST-MESSAGE (A17)
 RESET  MUXNAT-MESSAGE (A1/32727)
 REDEFINE MUXNAT-MESSAGE (1) (MUXNAT-MESSAGE-A17 (A17))
 RESET  MUXNAT-RETURNMSG (A1/32727)
 REDEFINE MUXNAT-RETURNMSG (1) (MUXNAT-RETURNMSG-A25 (A25))
*
 PERFORM MUXNAT-OPEN-CONNECTION
 IF MUXNAT-RETCODE < 0
   ESCAPE ROUTINE
 MOVE TEST-MESSAGE TO MUXNAT-MESSAGE-A17
 MOVE 17 TO MUXNAT-MESSAGE-LEN
 PERFORM MUXNAT-SEND-MESSAGE
 IF MUXNAT-RETCODE >= 0
   PERFORM MUXNAT-RECEIVE-RESPONSE
 PERFORM MUXNAT-CLOSE-CONNECTION
*
DEFINE SUBROUTINE MUXNAT-OPEN-CONNECTION
 CALL "MUXNAT" USING
      MUXNAT-HANDLE
      MUXNAT-REMOTE-HOST
      MUXNAT-REMOTE-PORT
      MUXNAT-ERRNO
      MUXNAT-RETCODE
DISPLAY NOHDR 'RETURN FROM MUXNAT'
DISPLAY NOHDR 'MUXNAT-ERRNO'
DISPLAY NOHDR MUXNAT-ERRNO MUXNAT-ERRNO-I
DISPLAY NOHDR 'MUXNAT-RETCODE'
DISPLAY NOHDR MUXNAT-RETCODE MUXNAT-RETCODE-I
END-SUBROUTINE
*
DEFINE SUBROUTINE MUXNAT-SEND-MESSAGE
 CALL "MUXNATS" USING
      MUXNAT-HANDLE
      MUXNAT-MESSAGE-LEN
      MUXNAT-MESSAGE(1)
      MUXNAT-HSECS-FOR-ACK
      MUXNAT-ERRNO
      MUXNAT-RETCODE
 DISPLAY NOHDR 'RETURN FROM MUXNATS'
 DISPLAY NOHDR 'MUXNAT-ERRNO'
 DISPLAY NOHDR MUXNAT-ERRNO MUXNAT-ERRNO-I
 DISPLAY NOHDR 'MUXNAT-RETCODE'
 DISPLAY NOHDR MUXNAT-RETCODE MUXNAT-RETCODE-I
END-SUBROUTINE
*
DEFINE SUBROUTINE MUXNAT-RECEIVE-RESPONSE
 CALL "MUXNATR" USING
      MUXNAT-HANDLE
      MUXNAT-RETURNMSG-LEN
      MUXNAT-RETURNMSG(1)
      MUXNAT-HSECS-TO-WAIT
      MUXNAT-ERRNO
      MUXNAT-RETCODE
 DISPLAY NOHDR 'RETURN FROM MUXNATR'
 DISPLAY NOHDR 'MUXNAT-ERRNO'
 DISPLAY NOHDR MUXNAT-ERRNO MUXNAT-ERRNO-I
 DISPLAY NOHDR 'MUXNAT-RETCODE'
 DISPLAY NOHDR MUXNAT-RETCODE MUXNAT-RETCODE-I
 DISPLAY MUXNAT-RETURNMSG-A25
END-SUBROUTINE
*
DEFINE SUBROUTINE MUXNAT-CLOSE-CONNECTION
 CALL "MUXNATC" USING
      MUXNAT-HANDLE
      MUXNAT-ERRNO
      MUXNAT-RETCODE
 DISPLAY NOHDR 'RETURN FROM MUXNATC'
 DISPLAY NOHDR 'MUXNAT-ERRNO'
 DISPLAY NOHDR MUXNAT-ERRNO MUXNAT-ERRNO-I
 DISPLAY NOHDR 'MUXNAT-RETCODE'
 DISPLAY NOHDR MUXNAT-RETCODE MUXNAT-RETCODE-I
END-SUBROUTINE
```

## 7.3 MUXNAT API Function Sets

The MUXNAT APIs are contained in the following function sets:

- **Open** on page 90
- **Send** on page 92
- **Receive** on page 93
- **Close** on page 94

### Open

#### Syntax

```
call "MUXNAT" using
     MUXNAT-handle
     MUXNAT-remote-host
     MUXNAT-remote-port
     MUXNAT-errno
     MUXNAT-retcode.
```

### Description

This function creates a socket connection to the MUX server e*Way running on the specified remote host and TCP/IP port. This socket connection is defined by a unique identifier, or “handle,” that is returned by the OPEN. Note this allows multiple connections to be opened and maintained by a single Natural application to a single or multiple MUX server e*Ways.

### Sample

Sample working storage definitions:

```
01  MUXNAT-handle         pic s9(8)    binary value +0.
01  MUXNAT-remote-host    pic  x(24)   value 'remote.host.name'.
01  MUXNAT-remote-port    pic  9(8)    binary value 26051.
01  MUXNAT-errno          pic  9(8)    binary value 0.
01  MUXNAT-retcode        pic s9(8)    binary value +0.
```

### Parameters and returns set by the application

**MUXNAT-handle**
A 4-byte binary number, initialized to zero.

**Returns**
TCP/IP socket number for the established connection.

**MUXNAT-remote-host**
A 24-byte character field, containing the DNS name of the remote host on which the MUX server e*Way is listening.

**Returns**
Unchanged.

**MUXNAT-remote-port**
A 4-byte binary number, containing the TCP/IP port number to which the MUX server e*Way is listening.

**Returns**
Unchanged

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Returns**
If **MUXNAT-retcode** is negative (see below), **MUXNAT-errno** contains an error number.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Returns**
A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXNAT-errno**.

## Send

### Syntax

```
call "MUXNATS" using
     MUXNAT-handle
     MUXNAT-message-len
     MUXNAT-message
     MUXNAT-hsecs-for-ack
     MUXNAT-errno
     MUXNAT-retcode.
```

### Description

This function sends a message or block of data to the MUX server e*Way. The function then waits a specified time (expressed in hundredths of seconds) for an acknowledgment to arrive on the socket connection identified by the passed handle.

### Sample

Sample working storage definitions:

```
01   MUXNAT-handle              pic s9(8)      binary.
01   MUXNAT-message-len         pic  9(8)      binary.
01   MUXNAT-message             pic  x(32703)  value spaces.
01   MUXNAT-hsecs-for-ack       pic  9(8)      binary.
01   MUXNAT-errno               pic  9(8)      binary value 0.
01   MUXNAT-retcode             pic s9(8)      binary value +0.
```

### Parameters and returns set by the application

**MUXNAT-handle**
A 4-byte binary number containing the socket number returned by the OPEN.

**Returns**
Unchanged.

**MUXNAT-message-len**
A 4-byte binary number containing the length, in bytes, of the message to be sent to the MUX server e*Way. The maximum size is 32K to 40 bytes, or 32727 bytes.

**Returns**
Unchanged.

**MUXNAT-message**
A 32727-byte character field containing the actual data to be sent to the MUX server e*Way. The contents of this field are transmitted without a conversion of any kind.

**Returns**
Unchanged.

**MUXNAT-hsecs-for-ack**
A 4-byte binary number, initialized to zero. Hundredths of seconds to wait for an acknowledgment (ACK) from e*Gate after a SEND.

**Returns**
Unchanged.

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Returns**
If **MUXNAT-retcode** is negative (see below), **MUXNAT-errno** contains an error number.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Returns**
A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXNAT-errno**.

## Receive

### Syntax

```
call "MUXNATR" using
     MUXNAT-handle
     MUXNAT-returnmsg-len
     MUXNAT-returnmsg
     MUXNAT-hsecs-to-wait
     MUXNAT-errno
     MUXNAT-retcode.
```

### Description

This function receives a message or block of data from the MUX server e*Way. The function waits a specified time (expressed in hundredths of seconds) for a message to arrive on the socket connection identified by the passed handle.

### Sample

Sample working storage definitions:

```
01   MUXNAT-handle          pic s9(8)       binary.
01   MUXNAT-returnmsg-len   pic  9(8)       binary.
01   MUXNAT-returnmsg       pic  x(32727)   value spaces.
01   MUXNAT-hsecs-to-wait   pic  9(8)       binary value 100.
01   MUXNAT-errno           pic  9(8)       binary value 0.
01   MUXNAT-retcode         pic s9(8)       binary value +0.
```

### Parameters and returns set by the application

**MUXNAT-handle**
A 4-byte binary number, containing the socket number returned by the OPEN.

**Returns**
Unchanged.

**MUXNAT-returnmsg-len**
A 4-byte binary number, initialized to zero.

**Returns**
The length, in bytes, of the data received from the MUX server e*Way.

**MUXNAT-returnmsg**
A 32727-byte character field.

**Returns**
The data received from the MUX server e*Way.

**MUXNAT-hsecs-to-wait**
A 4-byte binary number, representing the hundredths of seconds to wait for a response from e*Gate.

**Returns**
Unchanged.

**MUXNAT-errno**
A 4-byte binary number, initialized to zero.

**Returns**
If **MUXNAT-retcode** is negative (see below), **MUXNAT-errno** contains an error number.

**MUXNAT-retcode**
A 4-byte signed binary number, initialized to zero.

**Returns**
A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXNAT-errno**.

## Close

### Syntax

```
call "MUXNATC" using
     MUXNAT-handle
     MUXNAT-errno
     MUXNAT-retcode.
```

### Description

The **Close** function shuts down the socket connection with the MUX server e*Way and frees any resources associated with it.

### Sample

Sample working storage definitions:

```
01   MUXNAT-handle    pic s9(8)   binary.
01   MUXNAT-errno     pic  9(8)   binary value 0.
01   MUXNAT-retcode   pic s9(8)   binary value +0.
```

### Parameters and returns set by the application

**MUXNAT-handle**
A 4-byte binary number, containing the socket number returned by the OPEN.

**Returns**

Unchanged.

**MUXNAT-errno**

A 4-byte binary number, initialized to zero.

**Returns**

If **MUXNAT-retcode** is negative (see below), **MUXNAT-errno** contains an error number.

**MUXNAT-retcode**

A 4-byte signed binary number, initialized to zero.

**Returns**

A value of zero or greater indicates a successful call. A negative value signifies an error; the error number is contained in **MUXNAT-errno**.

Chapter 8

# Java Methods and Classes

This chapter provides an overview of the Java classes and methods contained in the e*Way Intelligent Adapter for ADABAS Natural, which are used to extend the functionality of the e*Way.

## 8.1 e*Way Methods and Classes: Overview

For any e*Way, communication takes place both on the e*Gate Integrator 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 ADABAS Natural e*Way, the **stceway.exe** file (creates a Multi-Mode e*Way; see **Chapter 3**) is used to communicate between the e*Way and e*Gate. A Java Collaboration is utilized to keep the communication open between the e*Way and the external system or network.

## 8.2 Using Java Methods

Java methods have been added to make it easier to set information in the ADABAS Natural e*Way Event Type Definitions (ETDs), as well as get information from them. The nature of this data transfer depends on the configuration parameters (see **Chapter 4**) you set for the e*Way in the e*Gate Schema Designer's e*Way Configuration Editor window.

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

***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 or Collaboration Rules Editor windows.

For example, if the node name is **CommArea**, the associated **javaName** is **CommArea**. If you want to get the node value, use the Java method called **getCommArea()**. If you want to set the node value, use the Java method called **setCommArea()**.

These methods are contained in the following Java classes:

- **Cicsclient Class** on page 97
- **NaturalClient Class** on page 128

## 8.3 Cicsclient Class

The **Cicsclient** class is used by the CICS client.

### 8.3.1 Methods of the Cicsclient Class

These methods are described in detail on the following pages:

**commAreaToPackedDecimal** on page 99

**commAreaZonedToString** on page 100

**execute** on page 100

**getCommArea** on page 101

**getCommAreaLength** on page 102

**getCommAreaString** on page 102

**getEciCallbackable** on page 103

**getEciExtend** on page 103

**getEciLuwToken** on page 104

**getEciSync** on page 104

**getEncodedCommAreaString** on page 105

**getEncoding** on page 105

**getHexString** on page 106

**getMessageQualifier** on page 106

**getPassword** on page 106

**getPort** on page 107

**getProgram** on page 107

**getServer** on page 108

**getServerList** on page 108

**getSslClass** on page 108

**getSslPassword** on page 109

**getTraceDumpOffset** on page 109

**getTraceFilename** on page 110

**getTraceLevel** on page 110

**getTraceTiming** on page 110

**getTraceTruncationSize** on page 111

**getTransId** on page 111

**getUrl** on page 112

**getUserId** on page 112

**handleConfigValues** on page 112

**handleTrace** on page 113

**initialize** on page 113

**initJavaGateway** on page 114

**packedDecimalToString** on page 115

**reset** on page 115

**sendRequest** on page 116

**setCommArea** on page 116

**setCommAreaLength** on page 117

**setEciCallbackable** on page 117

**setEciExtend** on page 117

**setEciLuwToken** on page 118

**setEciSync** on page 118

**setEncoding** on page 119

**setMessageQualifier** on page 119

**setPassword** on page 120

**setPort** on page 120

**setProgram** on page 121

**setServer** on page 121

**setSslClass** on page 122

**setSslPassword** on page 122

**setTraceDumpOffset** on page 122

**setTraceFilename** on page 123

**setTraceLevel** on page 123

**setTraceTiming** on page 124

**setTraceTruncationSize** on page 124

These methods are explained in detail in this section.

## CicsClient

### Description

A constructor.

### Syntax

```
public CicsClient()
```

### Parameters

None.

### Return Values

None.

### Throws

None.

## commAreaToPackedDecimal

### Description

Builds a packed decimal from an existing CommArea object.

### Syntax

```
public com.stc.eways.cics.PackedDecimal commAreaToPackedDecimal
    (int offset, int intSize, int decSize)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| offset | integer | The offset value. |
| intSize | integer | The size of the CommArea. |
| decSize | integer | The decimal value. |

### Return Values

**com.stc.eways.cics.PackedDecimal**
The new packed decimal.

### Throws

None

## commAreaZonedToString

### Description

Builds a string from an existing CommArea zoned object.

### Syntax

```
public java.lang.String commAreaZonedToString(int offset, int len)

public java.lang.String commAreaZonedToString(int offset, int len,
    java.lang.String enc)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| offset | integer | The offset value. |
| len | integer | The length of the CommArea. |
| enc | java.lang.String | The character encoding type. |

### Return Values

**java.lang.String**
The new string.

### Throws

None.

## execute

### Description

Executes the CICS program.

### Syntax

```
public void execute()

public void execute(boolean eciSynCall, java.lang.String
     cicsServerName, java.lang.String cicsUserId, java.lang.String
     cicsPassword, java.lang.String cicsProgram, java.lang.String
     cicsTransId, byte[] ba, int len, boolean eciExtendMode, int
     eciLUWToken, int msgQualifier, com.stc.eways.cics.Callbackable
     eciCallbackableObj)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| eciSynCall | Boolean | A Boolean value indicating whether to use the ECI Synchronous Call mode. |
| cicsServerName | java.lang.String | The CICS server name. |
| cicsUserId | java.lang.String | The user ID. |
| cicsPassword | java.lang.String | The password associated with the specified user ID. |
| cicsProgram | java.lang.String | The CICS program name to be executed. |
| cicsTransId | java.lang.String | The CICS transaction ID. |
| ba | byte array | A byte array for the CommArea length. |
| len | integer | The CommArea length |
| eciExtendMode | Boolean | A Boolean value indicating whether to implement ECI extend mode. |
| eciLUWToken | integer | An ECI Logical Unit of Work (LUW) token. |
| msgQualifier | integer | Application-provided identifier. |
| eciCallbackableObj | com.stc.eways.cics.Callbackable | An ECI object that can be called back. This object can be null if no callback is required. |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## getCommArea

### Description

Constructs a CommArea.

### Syntax

```
public byte[] getCommArea()
```

#### Parameters

None.

#### Return Values

**byte array**
The CommArea byte array.

#### Throws

None.

## getCommAreaLength

#### Description

Constructs the CommArea length.

#### Syntax

```
public int getCommAreaLength()
```

#### Parameters

None

#### Return Values

**integer**
The CommArea length.

#### Throws

None.

## getCommAreaString

#### Description

Constructs a CommArea string using one of the following operations:

- Converting the CommArea array of bytes using the platform's default character encoding.
- Converting the CommArea array of bytes with offset and len using the platform's default character encoding.
- Converting the CommArea array of bytes with offset and len using the character encoding specified as an argument.
- Converting the CommArea array of bytes using the character encoding specified as an argument.

### Syntax

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

public java.lang.String getEncodedCommAreaString(int offset, int len)

public java.lang.String getCommAreaString(int offset, int len,
    java.lang.String enc)

public java.lang.String getCommAreaString(java.lang.String enc)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| offset | integer | The offset value. |
| len | integer | The length of the CommArea. |
| enc | java.lang.String | The character encoding type. |

### Return Values

**java.lang.String**
The CommArea string.

### Throws

None.

## getEciCallbackable

### Description

Gets the ECI object that can be called back.

### Syntax

```
public com.stc.eways.cics.Callbackable getEciCallbackable()
```

### Parameters

None.

### Return Values

**com.stc.eways.cics.Callbackable**
The ECI object that can be called back.

### Throws

None.

## getEciExtend

### Description

Determines whether the ECI LUW has been set to extended.

### Syntax

```
public boolean getEciExtend()
```

### Parameters

None

### Return Values

**Boolean**
Returns **true** to indicate that the extended request is implemented; otherwise, returns **false**.

### Throws

None.

## getEciLuwToken

### Description

Gets the ECI LUW token value.

### Syntax

```
public int getEciLuwToken()
```

### Parameters

None.

### Return Values

**integer**
The ECI LUW token value.

### Throws

None.

## getEciSync

### Description

Queries whether the state is set to synchronous.

### Syntax

```
public boolean getEciSync()
```

### Parameters

None.

### Return Values

**Boolean**
Returns **true** to indicate that the ECI state is set to synchronous.

### Throws

None.

## getEncodedCommAreaString

### Description

Constructs a CommArea String by doing one of the following operations:

- Converting the CommArea array of bytes using the character encoding specified previously for the ETD.
- Converting the CommArea array of bytes with offset and len using the character encoding specified previously for the ETD.

### Syntax

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

```
public java.lang.String getEncodedCommAreaString(int offset, int len)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| offset | integer | The offset value. |
| len | integer | The length of the CommArea |

### Return Values

**java.lang.String**
The encoded CommArea string value.

### Throws

**java.io.UnsupportedEncodingException**
Indicates an unsupported encoding.

## getEncoding

### Description

Gets the encoding key.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The encoding type.

### Throws

None.

## getHexString

### Description

Gets the hexadecimal string.

### Syntax

```
public static java.lang.String getHexString(byte[] ba)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| ba | byte array | The hexadecimal string to obtain. |

### Return Values

**java.lang.String**
The hexadecimal string.

### Throws

None

## getMessageQualifier

### Description

Gets the Message Qualifier information.

### Syntax

```
public int getMessageQualifier()
```

### Parameters

None

### Return Values

**integer**
The Message Qualifier information.

### Throws

None.

## getPassword

### Description

Gets the password and decrypts it.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The password.

### Throws

None.

## getPort

### Description

Gets the port information.

### Syntax

```
public int getPort()
```

### Parameters

None.

### Return Values

**integer**
The port information.

### Throws

None.

## getProgram

### Description

Gets the name of the CICS program.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The CICS program name.

### Throws

None.

## getServer

### Description

Gets the CICS server information.

### Syntax

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

### Parameters

None

### Return Values

**java.lang.String**
The name of the CICS server.

### Throws

None.

## getServerList

### Description

Gets a list of CICS servers defined.

### Syntax

```
public java.lang.String[] getServerList(int maxNumSystems)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| maxNumSystems | integer | The maximum number of systems. |

### Return Values

**java.lang.String[]**
Returns a list of the defined CICS servers.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## getSslClass

### Description

Gets the name of the Secure Sockets Layer (SSL) class.

### Syntax

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

### Parameters

None

### Return Values

**java.lang.String**
The name of the SSL class.

### Throws

None.

## getSslPassword

### Description

Gets the SSL password.

### Syntax

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

### Parameters

None

### Return Values

**java.lang.String**
The SSL password.

### Throws

None.

## getTraceDumpOffset

### Description

Gets the trace dump offset value.

### Syntax

```
public int getTraceDumpOffset()
```

### Parameters

None.

### Return Values

**integer**
The trace dump offset value.

### Throws

None.

## getTraceFilename

### Description

Gets the trace file name.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The trace file name.

### Throws

None.

## getTraceLevel

### Description

Gets the defined trace level value.

### Syntax

```
public int getTraceLevel()
```

### Parameters

None.

### Return Values

**integer**
The trace level.

### Throws

None.

## getTraceTiming

### Description

Gets the defined trace timing information.

### Syntax

```
public boolean getTraceTiming()
```

### Parameters

None

### Return Values

**Boolean**
Returns **true** to indicate the timing trace mask is implemented.

### Throws

None.

## getTraceTruncationSize

### Description

Gets the trace truncation size of the hexadecimal dumps.

### Syntax

```
public int getTraceTruncationSize()
```

### Parameters

None.

### Return Values

**integer**
The size of the trace truncation setting.

### Throws

None.

## getTransId

### Description

Gets the transaction ID of the current transaction.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The transaction ID for the current transaction.

### Throws

None.

## getUrl

### Description

Gets the URL of the CICS Transaction Gateway.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The URL of the CICS Transaction Gateway.

### Throws

None.

## getUserId

### Description

Gets the user ID associated with the terminal.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The user ID associated with the terminal, or a null if the user ID is set to null or it is a basic terminal.

### Throws

None.

## handleConfigValues

### Description

Implements the values assigned in the configuration file for the related e*Way Connection.

### Syntax

```
protected void handleConfigValues(java.util.Properties props)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| props | java.util.Properties | The configuration property values. |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a communication error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## handleTrace

### Description

Implements the trace flags based on parsed configuration values.

### Syntax

```
public void handleTrace()
```

### Parameters

None.

### Return Values

None.

### Throws

None.

## initialize

### Description

Initializes the ETD.

### Syntax

```
public void initialize(com.stc.common.collabService.JCollabController
      cntrCollab, java.lang.String sKey, int iMode)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| cntrCollab | com.stc.common.collabService.JCollabController | The Java CollabConroller object. |
| sKey | java.lang.String | The key to a JMsgObject. |
| iMode | integer | The mode for the ETD. The possible values are:<br>IN_MODE<br>OUT_MODE<br>IN_OUT_MODE |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

### Additional Information

Overrides **initialize()** in the class **com.stc.jcsre.SimpleETDImpl**.

## initJavaGateway

### Description

Initializes the Java Gateway object to allow the flow of data.

### Syntax

```
public void initJavaGateway()
```

### Parameters

None.

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## packedDecimalToString

### Description

Converts a packed decimal to a string.

### Syntax

```
public static java.lang.String packedDecimalToString
    (com.stc.eways.cics.PackedDecimal pd)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| pd | com.stc.eway.cics.PackedDecimal | The packed decimal to be converted. |

### Return Values

**java.lang.String**

### Throws

None.

## reset

### Description

Resets the data content of an ETD.

### Syntax

```
public boolean reset()
```

### Parameters

None.

### Return Values

**Boolean**
Returns **true** if the reset clears the data content of the ETD; otherwise, returns **false** if the ETD does not have a meaningful implementation of **reset()**, in which case it is necessary to create a new ETD.

### Throws

None.

### Additional Information

Overrides reset in class **com.stc.jcsre.SimpleETDImpl**

## sendRequest

### Description

Sends a flow of data contained in the ECI request object to the Java Gateway and determines whether the send has been successful by checking the return code.

### Syntax

```
public void sendRequest(com.stc.eways.cics.ECIRequest request)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| request | com.stc.eway.cics.ECI Request | The ECI Request object to be sent. |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## setCommArea

### Description

Sets the CommArea to be made available to CICS.

### Syntax

```
public void setCommArea(byte[] ba)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| ba | byte array | A byte array containing the information required to set the CommArea. |

### Return Values

None.

### Throws

None.

## setCommAreaLength

### Description

Sets the CommArea length.

### Syntax

```
public void setCommAreaLength(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The CommArea length to be set. |

### Return Values

None.

### Throws

None.

## setEciCallbackable

### Description

Sets the ECI value that can be called back.

### Syntax

```
public void setEciCallbackable(com.stc.eways.cics.Callbackable c)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | com.stc.eway.cics.Callbackable | The ECI value that can be called back. |

### Return Values

None.

### Throws

None.

## setEciExtend

### Description

Enables the ECI Extended Mode.

### Syntax

```
public void setEciExtend(boolean b)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | Boolean | A true enables the ECI Extended mode (sets the mode to Extended). |

### Return Values

None.

### Throws

None.

## setEciLuwToken

### Description

Sets the ECI LUW token value.

### Syntax

```
public void setEciLuwToken(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The application identifier. |

### Return Values

None.

### Throws

None.

## setEciSync

### Description

Enables the ECI Synchronous mode.

### Syntax

```
public void setEciSync(boolean b)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | Boolean | A true enables the ECI Synchronous mode (sets the mode to Synchronous). |

### Return Values

None.

### Throws

None.

## setEncoding

### Description

Sets the encryption type for encoding purposes.

### Syntax

```
public void setEncoding(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The encryption type. |

### Return Values

None.

### Throws

None.

## setMessageQualifier

### Description

Sets the Message Qualifier associated with the current request.

### Syntax

```
public void setMessageQualifier(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The application identifier. |

### Return Values

None.

### Throws

None.

## setPassword

### Description

Sets the password associated with the terminal.

### Syntax

```
public void setPassword(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The password to be set. |

### Return Values

None.

### Throws

None.

### Additional Information

Invoking this method automatically flags the terminal as an extended type of terminal. The password is not picked up until another send operation is completed or the terminal is connected.

## setPort

### Description

Sets the port number necessary to communicate with the Java Gateway.

### Syntax

```
public void setPort(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The Gateway port number. |

### Return Values

None.

### Throws

None.

## setProgram

### Description

Sets the CICS program identity.

### Syntax

```
public void setProgram(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The name of the CICS program. |

### Return Values

None.

### Throws

None.

## setServer

### Description

Sets the server identity on which the CICS program is running.

### Syntax

```
public void setServer(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The name of the server on which CICS resides. |

### Return Values

None.

### Throws

None.

## setSslClass

### Description

Sets the identity of the SSL class.

### Syntax

```
public void setSslClass(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The SSL class name. |

### Return Values

None.

### Throws

None.

## setSslPassword

### Description

Sets the password required to access SSL information.

### Syntax

```
public void setSslPassword(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The SSL password. |

### Return Values

None.

### Throws

None.

## setTraceDumpOffset

### Description

Sets the offset value for trace dumping.

### Syntax

```
public void setTraceDumpOffset(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The offset amount. |

### Return Values

None.

### Throws

None.

## setTraceFilename

### Description

Sets the name of the trace file to be used.

### Syntax

```
public void setTraceFilename(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The name of the trace file. |

### Return Values

None.

### Throws

None.

## setTraceLevel

### Description

Sets the debugging trace level.

### Syntax

```
public void setTraceLevel(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The trace level to be set. |

### Return Values

None.

### Throws

None.

## setTraceTiming

### Description

Enables the debugging trace timing.

### Syntax

```
public void setTraceTiming(boolean b)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| b | Boolean | A true enables the debugging trace timing (sets the trace timing to “on”). |

### Return Values

None.

### Throws

None.

## setTraceTruncationSize

### Description

Sets the trace truncation size.

### Syntax

```
public void setTraceTruncationSize(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | integer | The truncation size to be set. |

### Return Values

None.

### Throws

None.

## setTransId

### Description

Sets the CICS transaction ID.

### Syntax

```
public void setTransId(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The CICS transaction ID. |

### Return Values

None.

### Throws

None.

## setUrl

### Description

Sets the URL to the Java Transaction Gateway.

### Syntax

```
public void setUrl(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The URL for the Transaction Gateway. |

### Return Values

None.

### Throws

None.

## setUserId

### Description

Sets the user ID associated with the terminal.

#### Syntax

```
public void setUserId(java.lang.String s)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The terminal user ID. |

#### Return Values

None.

#### Throws

None.

## terminate

#### Description

Terminates the ETD.

#### Syntax

```
public void terminate()
```

#### Parameters

None

#### Return Values

None.

#### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

#### Additional Information

Overrides **terminate()** in the class **com.stc.jcsre.SimpleETDImpl**.

## toPackedDecimal

#### Description

Builds a packed decimal from a string number.

#### Syntax

```
public static com.stc.eways.cics.PackedDecimal toPackedDecimal
    (java.lang.String number, int intSize, int decSize)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| number | java.lang.String | Decimal string representation to be converted. |
| intSize | integer | The size of the package. |
| decSize | integer | The size of the package. |

### Return Values

**com.stc.eways.cics.PackedDecimal**
The packed decimal.

### Throws

**java.lang.NumberFormatException**

### Additional Information

Converts the integer string +-99999.99 in a packed decimal IBM data Flow -> Each digit is a 0..9 Numerical value last digit is the sign digit: A|C|E|F => +; B|D => -; the decimal point is virtual; its position is defined in the second byte of **dec_len**.

## toZoned

### Description

Converts a number to zoned data.

### Syntax

```
public static byte[] toZoned(java.lang.String number)

public static byte[] toZoned(java.lang.String number,
    java.lang.String enc)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| number | java.lang.String | The number to be converted. |
| enc | java.lang.String | The encryption type. |

### Return Values

**byte array**
Returns a byte array containing the new zoned data.

### Throws

**java.lang.NumberFormatException**
Indicates an error occurred as a result of a numeric format exception.

### zonedToString

#### Description

Converts zoned data to a string.

#### Syntax

```
public static java.lang.String zonedToString(byte[] zoned)

public static java.lang.String zonedToString(byte[] zoned,
    java.lang.String enc)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| zoned | byte array | Description |
| enc | java.lang.String | The encryption type. |

#### Return Values

**java.lang.String**
The new converted string.

#### Throws

**java.lang.NumberFormatException**

## 8.4 NaturalClient Class

```
java.lang.Object
    com.stc.jcsre.SimpleETDImpl
        com.stc.eways.cics.CicsClient
            com.stc.eways.natural.NaturalClient
```

The public class **NaturalClient** extends **com.stc.eways.cics.CicsClient**.

### 8.4.1 Methods of the NaturalClient Class

## NaturalClient

### Description

A constructor.

### Syntax

```
public NaturalClient()
```

### Parameters

None.

### Return Values

None.

### Throws

None.

## execute

### Description

Executes the Natural program.

### Syntax

```
public void execute()

public void execute(boolean eciSynCall, java.lang.String
     cicsServerName, java.lang.String cicsUserId, java.lang.String
     cicsPassword, java.lang.String cicsProgram, java.lang.String
     cicsTransId, byte[] ba, int len, boolean eciExtendMode, int
     eciLUWToken, int msgQualifier,
     com.stc.eways.natural.Callbackable eciCallbackableObj)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| eciSynCall | Boolean | A true sets the ECI mode to synchronous. |
| cicsServerName | java.lang.String | The CICS server name. |
| cicsUserId | java.lang.String | The CICS User ID. |
| cicsPassword | java.lang.String | The associated CICS password. |
| cicsProgram | java.lang.String | The CICS program name. |

| Name (Continued) | Type | Description |
|---|---|---|
| cicsTransId | java.lang.String | The CICS transaction ID. |
| ba | byte array | The data payload. |
| len | int | The length. |
| eciExtendMode | Boolean | A true sets the ECI mode to extended. |
| eciLUWToken | int | The ECI LUW token value. |
| msgQualifier | int | Application-provided identifier. |
| eciCallbackableObj | com.stc.eways.natural .Callbackable | The ECI object that can be called back. |

### Return Values

None.

### Throws

None.

## getNatExecProgram

### Description

Gets the name of the Natural execute program.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The name of the Natural execute program.

### Throws

None.

## getNatExecTransId

### Description

Gets the name of the Natural execute transaction ID.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The Natural execute transaction ID.

### Throws

None.

## getNatInitProgram

### Description

Gets the Natural initialize program.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The name of the Natural initialize program.

### Throws

None.

## getNatInitTransId

### Description

Gets the Natural Initialize transaction ID.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The Natural Initialize transaction ID.

### Throws

None.

## getNatTermProgram

### Description

Gets the Natural Terminate program information.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The Natural Terminate program information.

### Throws

None.

## getNatTermTransId

### Description

Gets the Natural Terminate transaction ID.

### Syntax

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

### Parameters

None.

### Return Values

**java.lang.String**
The Natural Terminate transaction ID.

### Throws

None.

## handleConfigValues

### Description

Implements the values assigned in the configuration file for the e*Way Connection.

### Syntax

```
protected void handleConfigValues(java.util.Properties props)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| props | java.util.Properties | The configuration property values. |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a connection error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

## initialize

### Description

Initializes the ETD.

### Syntax

```
public void initialize(com.stc.common.collabService.JCollabController
    cntrCollab, java.lang.String sKey, int iMode)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| cntrCollab | com.stc.common.coll abService.JCollabCon troller | The JCollabController object. |
| sKey | java.lang.String | The key to a JMSObject. |
| iMode | int | The mode for the ETD. The possible values are:<br>IN_MODE<br>OUT_MODE<br>IN_OUT_MODE |

### Return Values

None.

### Throws

**com.stc.common.collabService.CollabConnException**
Indicates a communication error.

**com.stc.common.collabService.CollabDataException**
Indicates a data error.

### Additional Information

Overrides initialize in **com.stc.eways.cics.CicsClient**.

## reset

### Description

Resets the data content of the ETD.

### Syntax

```
public boolean reset()
```

### Parameters

None.

### Return Values

**Boolean**

Returns **true** indicating the reset clears the data content of the ETD; otherwise, returns **false** if the ETD does not have a meaningful implementation of **reset()**. In such a case, a new creation of the ETD is required.

### Throws

None.

## setNatExecProgram

### Description

Sets the value for the Natural Execute program.

### Syntax

```
public void setNatExecProgram(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The name of the program. |

### Return Values

None.

### Throws

None.

## setNatExecTransId

### Description

Sets the value for the Natural Execute transaction ID.

### Syntax

```
public void setNatExecTransID(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The transaction ID. |

### Return Values

None.

### Throws

None.

## setNatInitProgram

### Description

Sets the value for the Natural Initialize program.

### Syntax

```
public void setNatuInitProgram(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The transaction ID. |

### Return Values

None.

### Throws

None.

## setNatInitTimeout

### Description

Sets the Natural Initialize time-out value.

### Syntax

```
public void setNatInitTimeout(int i)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| i | int | The time-out value. |

### Return Values

None.

### Throws

None.

## setNatInitTransId

### Description

Sets the value for the Natural Initialize transaction ID.

### Syntax

```
public void setNatInitTransId(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The transaction ID. |

### Return Values

None.

### Throws

None.

## setNatTermProgram

### Description

Sets the value for the Natural Terminate program.

### Syntax

```
public void setNatTermProgram(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The name of the program. |

### Return Values

None.

### Throws

None.

## setNatTermTransId

### Description

Sets the value of the Natural Terminate transaction ID.

### Syntax

```
public void setNatTermTransId(java.lang.String s)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| s | java.lang.String | The transaction ID. |

### Return Values

None.

### Throws

None.

Appendix A

# Error Return Codes

This appendix explains the error return codes for MUXNAT.

## 8.5 MUXNAT Error Return Codes

The following return codes can be found in the *IP CICS Sockets Guide Version 2 Release 8 and 9, OS/390 SecureWay Communications Server*. As of the date of the creation of this document, it can be downloaded from the following URL:

**http://www-1.ibm.com/servers/s390/os390/bkserv/r10pdf/commserv.html**

The rest of this appendix explains these codes.

### 8.5.1 TCP/IP for MVS Error Return Codes

Table 3 explains the TCP/IP for MVS error return codes.

**Table 3** TCP/IP for MVS Error Return Codes

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 1 | EPERM | All | Permission is denied. No owner exists. | Check that TCP/IP is still active; Check the protocol value of the socket call. |
| 1 | EDOM | All | Argument is too large. | Check parameter values of the function call. |
| 2 | ENOENT | All | The data set or directory was not found. | Check files used by the function call. |
| 2 | ERANGE | All | The result is too large. | Check parameter values of the function call. |
| 3 | ESRCH | All | The process was not found. A table entry was not located. | Check parameter values and structures pointed to by the function parameters. |
| 4 | EINTR | All | A system call was interrupted. | Check that the socket connection and TCP/IP are still active. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 5 | EIO | All | An I/O error occurred. | Check status and contents of source database if this occurred during a file access. |
| 6 | ENXIO | All | The device or driver was not found. | Check status of the device attempting to access. |
| 7 | E2BIG | All | The argument list is too long. | Check the number of function parameters. |
| 8 | ENOEXEC | All | An EXEC format error occurred. | Check that the target module on an exec call is a valid executable module. |
| 9 | EBADF | All | An incorrect socket descriptor was specified. | Check socket descriptor value. It might be currently not in use or incorrect. |
| 9 | EBADF | Givesocket | The socket has already been given. The socket domain is not AF_INET. | Check the validity of function parameters. |
| 9 | EBADF | Select | One of the specified descriptor sets is an incorrect socket descriptor. | Check the validity of function parameters. |
| 9 | EBADF | Takesocket | The socket has already been taken. | Check the validity of function parameters. |
| 10 | ECHILD | All | There are no children. | Check if created subtasks still exist. |
| 11 | EAGAIN | All | There are no more processes. | Retry the operation. Data or condition might not be available at this time. |
| 12 | ENOMEM | All | There is not enough storage. | Check validity of function parameters. |
| 13 | EACCES | All | Permission denied, caller not authorized. | Check access authority of file. |
| 13 | EACCES | Takesocket | The other application (listener) did not give the socket to your application. Permission denied, caller not authorized. | Check access authority of file. |
| 14 | EFAULT | All | An incorrect storage address or length was specified. | Check validity of function parameters. |
| 15 | ENOTBLK | All | A block device is required. | Check device status and characteristics. |
| 16 | EBUSY | All | Listen has already been called for this socket. Device or file to be accessed is busy. | Check if the device or file is in use. |
| 17 | EEXIST | All | The data set exists. | Remove or rename existing file. |
| 18 | EXDEV | All | This is a cross-device link. A link to a file on another file system was attempted. | Check file permissions. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 19 | ENODEV | All | The specified device does not exist. | Check file name and if it exists. |
| 20 | ENOTDIR | All | The specified device does not exist. | Use a valid file that is a directory. |
| 21 | EISDIR | All | The specified directory is a directory. | Use a valid file that is not a directory. |
| 22 | EINVAL | All types | An incorrect argument was specified. | Check validity of function parameters. |
| 23 | ENFILE | All | Data set table overflow occurred. | Reduce the number of open files. |
| 24 | EMFILE | All | The socket descriptor table is full. | Check the maximum sockets specified in MAXDESC(). |
| 25 | ENOTTY | All | An incorrect device call was specified. | Check specified IOCTL() values. |
| 26 | ETXTBSY | All | A text data set is busy. | Check the currrent use of the file. |
| 27 | EFBIG | All | The specified data set is too large. | Check size of accessed dataset. |
| 28 | ENOSPC | All | There is no space left on the device. | Increase the size of accessed file. |
| 29 | ESPIPE | All | An incorrect seek was attempted. | Check the offset parameter for seek operation. |
| 30 | EROFS | All | The data set system is Read only. | Access data set for read only operation. |
| 31 | EMLINK | All | There are too many links. | Reduce the number of links to the accessed file. |
| 32 | EPIPE | All | The connection is broken. For socket write/send, peer has shutdown one or both directions. | Reconnect with the peer. |
| 33 | EDOM | All | The specified argument is too large. | Check and correct function parameters. |
| 34 | ERANGE | All | The result is too large. | Check parameter values. |
| 35 | EWOULDBLOCK | Accept | The socket is in nonblocking mode and connections are not queued. This is not an error condition. | Reissue Accept( ). |
| 35 | EWOULDBLOCK | Read<br>Recvfrom | The socket is in nonblocking mode and read data is not available. This is not an error condition. | Issue a select on the socket to determine when data is available to be read or reissue the Read( )/Recvfrom( ). |
| 35 | EWOULDBLOCK | Send<br>Sendto<br>Write | The socket is in nonblocking mode and buffers are not available. | Issue a select on the socket to determine when data is available to be written or reissue the Send( ), Sendto( ), or Write( ). |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 36 | EINPROGRESS | Connect | The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition. | See the Connect( ) description for possible responses. |
| 37 | EALREADY | Connect | The socket is marked nonblocking and the previous connection has not been completed. | Reissue Connect( ). |
| 37 | EALREADY | Maxdesc | A socket has already been created calling Maxdesc( ) or multiple calls to Maxdesc( ). | Issue Getablesize( ) to query it. |
| 37 | EALREADY | Setibmopt | A connection already exists to a TCP/IP image. A call to SETIBMOPT (IBMTCP_IMAGE), has already been made. | Only call Setibmopt( ) once. |
| 38 | ENOTSOCK | All | A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid. | Correct the socket descriptor value and reissue the function call. |
| 39 | EDESTADDRREQ | All | A destination address is required. | Fill in the destination field in the correct parameter and reissue the function call. |
| 40 | EMSGSIZE | Sendto<br>Sendmsg<br>Send<br>Write | The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt( ) call. | Either correct the length parameter, or send the message in smaller pieces. |
| 41 | EPROTOTYPE | All | The specified protocol type is incorrect for this socket. | Correct the protocol type parameter. |
| 42 | ENOPROTOOPT | Getsockopt<br>Setsockopt | The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported. | Correct the level or optname. |
| 42 | ENOPROTOOPT | Getibmsocket opt<br>Setibmsocket opt | Either the level or the specified optname is not supported. | Correct the level or optname. |
| 43 | EPROTONOSUPPORT | Socket | The specified protocol is not supported. | Correct the protocol parameter. |
| 44 | ESOCKTNOSUPPORT | All | The specified socket type is not supported. | Correct the socket type parameter. |
| 45 | EOPNOTSUPP | Accept<br>Givesocket | The selected socket is not a stream socket. | Use a valid socket. |
| 45 | EOPNOTSUPP | Listen | The socket does not support the Listen call. | Change the type on the Socket( ) call when the socket was created. Listen( ) only supports a socket type of SOCK_STREAM. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 45 | EOPNOTSUPP | Getibmopt Setibmopt | The socket does not support this function call. This command is not supported for this function. | Correct the command parameter. See Getibmopt( ) for valid commands. Correct by ensuing a Listen( ) was not issued before the Connect( ). |
| 46 | EPFNOSUPPORT | All | The specified protocol family is not supported or the specified domain for the client identifier is not AF_INET=2. | Correct the protocol family. |
| 47 | EAFNOSUPPORT | Bind Connect Socket | The specified address family is not supported by this protocol family. | For Socket( ), set the domain parameter to AF_INET. For Bind( ), and Connect( ), set Sin_Family in the socket address structure to AF_INET. |
| 47 | EAFNOSUPPORT | Getclient Givesocket | The socket specified by the socket descriptor parameter was not created in the AF_INET domain. | The Socket( ) call used to create the socket must be changed to use AF_INET for the domain parameter. |
| 48 | EADDRINUSE | Bind | The address is in a timed wait because a LINGER delay from a previous close or another process is using the address. | If you want to reuse the same address, use Setsocketopt( ) with SO_REUSEADDR. See Setsockopt( ). Otherwise, use a different address or port in the socket address structure. |
| 49 | EADDRNOTAVAIL | Bind | The specified address is incorrect for this host. | Correct the function address parameter. |
| 49 | EADDRNOTAVAIL | Connect | The calling host cannot reach the specified destinations. | Correct the function address parameter. |
| 50 | ENETDOWN | All | The network is down. | Retry when the connection path is up. |
| 51 | ENETUNREACH | Connect | The network cannot be reached. | Ensure that the target application is active. |
| 52 | ENETRESET | All | The network dropped a connection on a reset. | Reestablish the connection between the applications. |
| 53 | ECONNABORTED | All | The software caused a connection abend. | Reestablish the connection between the applications. |
| 54 | ECONNRESET | All | The connection to the destination host is not available. | |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 54 | ECONNRESET | Send Write | The connection to the destination host is not available. | The socket is closing. Issue Send( ) or Write( ) before closing the socket. |
| 55 | ENOBUFS | All | No buffer space is available. | Check the application for massive storage allocation call. |
| 55 | ENOBUFS | Accept | Not enough buffer space is available to create the new socket. | Call your system administrator. |
| 55 | ENOBUFS | Send Sendto Write | Not enough buffer space is available to send the new message. | Call your system administrator. |
| 56 | EISCONN | Connect | The socket is already connected. | Correct the socket descriptor on Connect( ) or do not issue a Connect( ) twice for the socket. |
| 57 | ENOTCONN | All | The socket is not connected. | Connect the socket before communicating. |
| 58 | ESHUTDOWN | All | A Send cannot be processed after socket shutdown. | Issue read/receive before shutting down the read side of the socket. |
| 59 | ETOOMANYREFS | All | There are too many references. A splice cannot be completed. | Call your system administrator. |
| 60 | ETIMEDOUT | Connect | The connection timed out before it was completed. | Ensure the server application is available. |
| 61 | ECONNREFUSED | Connect | The requested connection was refused. | Ensure server application is available and at specified port. |
| 62 | ELOOP | All | There are too many symbolic loop levels. | Reduce symbolic links to specified file. |
| 63 | ENAMETOOLONG | All | The file name is too long. | Reduce size of specified file name. |
| 64 | EHOSTDOWN | All | The host is down. | Restart specified host. |
| 65 | EHOSTUNREACH | All | There is no route to the host. | Set up network path to specified host and verify that host name is valid. |
| 66 | ENOTEMPTY | All | The directory is not empty. | Clear out specified directory and reissue call. |
| 67 | EPROCLIM | All | There are too many processes in the system. | Decrease the number of processes or increase the process limit. |
| 68 | EUSERS | All | There are to many users on the system. | Decrease the number of users or increase the user limit. |
| 69 | EDQUOT | All | The disk quota has been exceeded. | Call your system administrator. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 70 | ESTALE | All | An old NFS** data set handle was found. | Call your system administrator. |
| 71 | EREMOTE | All | There are too many levels of remote in the path. | Call your system administrator. |
| 72 | ENOSTR | All | The device is not a stream device. | Call your system administrator. |
| 73 | ETIME | All | The timer has expired. | Increase timer values or reissue function. |
| 74 | ENOSR | All | There are no more stream resources. | Call your system administrator. |
| 75 | ENOMSG | All | There is no message of the desired type. | Call your system administrator. |
| 76 | EBADMSG | All | The system cannot read the file message. | Verify that CS for OS/390 installation was successful and that message files were properly loaded. |
| 77 | EIDRM | All | The identifier has been removed. | Call your system administrator. |
| 78 | EDEADLK | All | A deadlock condition has occurred. | Call your system administrator. |
| 78 | EDEADLK | Select<br>Selectex | None of the sockets in the socket descriptor sets is either AF_NET or AF_IUCV sockets, and there is no time-out or no ECB specified. The select/selectex would never complete. | Correct the socket descriptor sets so that an AF_NET or AF_IUCV socket is specified. A time-out of ECB value can also be added to avoid the select/selectex from waiting indefinitely. |
| 79 | ENOLCK | All | No record locks are available. | Call your system administrator. |
| 80 | ENONET | All | The requested machine is not on the network. | Call your system administrator. |
| 81 | ERREMOTE | All | The object is remote. | Call your system administrator. |
| 82 | ENOLINK | all | The link has been severed. | Release the sockets and reinitialize the client-server connection. |
| 83 | EADV | All | An ADVERTISE error has occurred. | Call your system administrator. |
| 84 | ESRMNT | All | AnSRMOUNT error has occurred. | Call your system administrator. |
| 85 | ECOMM | All | A communication error has occurred on a Send call. | Call your system administrator. |
| 86 | EPROTO | All | A protocol error has occurred. | Call your system administrator. |
| 87 | EMULTIHOP | All | A multihop address link was attempted. | Call your system administrator. |
| 88 | EDOTDOT | All | A cross-mount point was detected. This is not an error. | Call your system administrator. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 89 | EREMCHG | all | The remote address has changed. | Call your system administrator. |
| 90 | ECONNCLOSED | All | The connection was closed by a peer. | Check that the peer is running. |
| 113 | EBADF | All | Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC( ). The default range is 0 to 49. | Reissue function with corrected socket descriptor. |
| 113 | EBADF | Bind socket | The socket descriptor is already being used. | Correct the socket descriptor. |
| 113 | EBADF | Givesocket | The socket has already been given. The socket domain is not AF_INET. | Correct the socket descriptor. |
| 113 | EBADF | Select | One of the specified descriptor sets is an incorrect socket descriptor. | Correct the socket descriptor. Set on Select( ) or Selectex( ). |
| 113 | EBADF | Takesocket | The socket has already been taken. | Correct the socket descriptor. |
| 113 | EBADF | Accept | A Listen( ) has not been issued before the Accept( ) | Issue Listen( ) before Accept( ). |
| 121 | EINVAL | All | An incorrect argument was specified. | Check and correct all function parameters. |
| 145 | E2BIG | All | The argument list is too long. | Eliminate excessive number of arguments. |
| 156 | EMVSINITIAL | All | Process initialization error. | Attempt to initialize again. |
| 1002 | EIBMSOCKOUTOFRANGE | Socket | A socket number assigned by the client interface code is out of range. | check the socket descriptor parameter. |
| 1003 | EIBMSOCKINUSE | Socket | A socket number assigned by the client interface code is already in use. | Use a different socket descriptor. |
| 1004 | EIBMIUCVERR | All | The request failed because of an IUCV error. This error is generated by the client stub code. | Ensure IUCV/VMCF is functional. |
| 1008 | EIBMCONFLICT | All | This request conflicts with a request already queued on the same socket. | Cancel the existing call or wait for its completion before reissuing this call. |
| 1009 | EIMBCANCELLED | All | The request was cancelled by the CANCEL call. | Informational, no action needed. |
| 1011 | EIBMBADTCPNAME | All | A TCP/IP name that is not valid was detected. | Correct the name specified in the IBM_TCPIMAGE structure. |
| 1011 | EIBMBADTCPNAME | Setibmopt | A TCP/IP name that is not valid was detected. | Correct the name specified in the IBM_TCPIMAGE. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 1011 | EIBMBADTCPNAME | INITAPI | A TCP/IP name that is not valid was detected. | Correct the name specified on the IDENT option TCPNAME field. |
| 1012 | EIBMBADREQUESTCODE | All | A request code that is not valid was detected. | Contact your system administrator. |
| 1013 | EIBMBADCONNECTIONSTATE | All | A connection token that is not valid was detected; bad state. | Verify TCP/IP is active. |
| 1014 | EIBMUNAUTHORIZED CALLER | All | An unauthorized caller specified an authorized keyword. | Ensure user ID has authority for the specified operation. |
| 1015 | EIBMBADCONNECTIONMATCH | All | A connection token that is not valid was detected. There is no such connection. | Verify TCP/IP is active. |
| 1016 | EIBMTCPABEND | All | An abend occurred when TCP/IP was processing this request. | Verify that TCP/IP has restarted. |
| 1026 | EIBMINVDELETE | All | Delete requestor did not create the connection. | Delete the request from the process that created it. |
| 1027 | EIBMINVSOCKET | All | A connection token that is not valid was detected. No such socket exists. | Call your system programmer. |
| 1028 | EIBMINVTCPCONNECTION | All | Connection terminated by TCP/IP. The token was invalidated by TCP/IP. | Reestablish the connection to TCP/IP. |
| 1032 | EIBMCALLINPROGRESS | All | Another call was already in progress. | Reissue after previous call has completed. |
| 1036 | EIBMNOACTIVETCP | Getibmopt | No TCP/IP image was found. | Ensure TCP/IP is active. |
| 1037 | EIBMINVTSRBUSERDATA | All | The request control block contained data that is not valid. | check your function parameters and call your system programmer. |
| 1038 | EIBMINVUSERDATA | All | The request control block contained user data that is not valid. | Check your function parameters and call your system programmer. |
| 1040 | EIBMSELECTEXPOST | SELECTEX | SELECTEX passed an ECB that was already posted. | Check whether the user's ECB was already posted. |
| 2001 | EINVALIDRXSOCKETCALL | REXX | A syntax error occurred in the RXSOCKET parameter list. | Correct the parameter list passed to the REXX socket call. |
| 2002 | ECONSOLEINTERRUPT | REXX | A console interrupt occurred. | Retry the task. |
| 2003 | ESUBTASKINVALID | REXX | The subtask ID is incorrect. | Correct the subtask ID on the INITIALIZE call. |
| 2004 | ESUBTASKALREADYACTIVE | REXX | The subtask is already active. | Only issue the INITIALIZE call once in your program. |
| 2005 | ESUBTASKALNOTACTIVE | REXX | The subtask is not active. | Issue the INITALIZE call before any other socket call. |

**Table 3** TCP/IP for MVS Error Return Codes (Continued)

| Error Number | Message Name | Socket Type | Error Description | Programmer's Response |
|---|---|---|---|---|
| 2006 | ESOCKETNOTALLOCATED | REXX | The specified socket could not be allocated. | Increase the user storage allocation for this job. |
| 2007 | EMAXSOCKETSREACHED | REXX | The maximum number of sockets has been reached. | Increase the number of allocate sockets, or decrease the number of sockets used by your program. |
| 2009 | ESOCKETNOTDEFINED | REXX | The socket is not defined. | Issue the SOCKET call before the call that fails. |
| 2011 | EDOMAINSERVERFAILURE | REXX | A Domain Name Server failure occurred. | Call your MVS system programmer. |
| 2012 | EINVALIDNAME | REXX | An incorrect name was received from the TCP/IP server. | Call your MVS system programmer. |
| 2013 | EINVALIDCLIENTID | REXX | An incorrect client ID was received from the TCP/IP server. | Call your MVS server. |
| 2014 | EINVALIDFILENAME | REXX | An error occurred during NUCEXT processing. | Specify the correct translation table file name, or verify that the translation table is valid. |
| 2016 | EHOSTNOTFOUND | REXX | The host is not found. | Call your MVS system programmer. |
| 2017 | EIPADDRNOTFOUND | REXX | Address not found. | Call your MVS system programmer. |

## 8.5.2 Sockets Extended Error Return Codes

Table 4 explains the Sockets Extended error return codes.

**Table 4** Sockets Extended Error Return Codes

| Error Code | Problem Description | System Action | Programmer's Response |
|---|---|---|---|
| 10100 | An ESTATE macro did not complete normally. | End the call. | Call your MVS system programmer. |
| 10101 | A STORAGE OBTAIN failed. | End the call. | Increase MVS storage in the application's address space. |
| 10108 | The first call from TCP/IP was not INITAPI or TAKESOCKET. | End the call. | Change the first TCP/IP call to INITAPI or TAKESOCKET. |
| 10110 | LOAD of EZBSOH03 (alias EZASOH03) failed. | End the call. | Call the IBM Software Support Center. |
| 10154 | Errors were found in the parameter list for an IOCTL call. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the IOCTL call. You might have incorrect sequencing of socket calls. |
| 10155 | The length parameter for an IOCTL call is 3200 (32 x 100). | Disable the subtask for interrupts. Return an error code to the caller. | Correct the IOCTL call. You might have incorrect sequencing of socket calls. |

**Table 4** Sockets Extended Error Return Codes (Continued)

| Error Code | Problem Description | System Action | Programmer's Response |
|---|---|---|---|
| 10159 | A zero or negative data length was specified for a READ or READV call. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the length in the READ call. |
| 10161 | The REQARG parameter in the IOCTL parameter list is zero. | End the call. | Correct the program. |
| 10163 | A 0 or negative data length was found for a RECV, RECVFROM, or RECVMSG call. | Disable the subtask for interrupts. Sever the DLC path. Return an error code to the caller. | Correct the data length. |
| 10167 | The descriptor set size for SELECT or SELECTEX call is less than or equal to zero. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the SELECT or SELECTEX call. You might have incorrect sequencing of socket calls. |
| 10168 | The descriptor set size in bytes for a SELECT or SELECTEX call is greater than 252. A number greater than the maximum number of allowed sockets (2000 is maximum) has been specified. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the descriptor set size. |
| 10170 | A zero or negative data length was found for a SEND or SENDMSG call. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the data length in the SEND call. |
| 10174 | A zero or negative data length was found for a SENDTO call. | Disable the subtask for interrupts. Return an error code to the caller. | correct the data length in the SENDTO call. |
| 10178 | The SETSOCKOPT option length is less than the minimum length. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the OPTLEN parameter. |
| 10179 | The SETSOCKOPT option length is greater than the maximum length. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the OPTLEN parameter. |
| 10184 | A data length of zero was specified for a WRITE call. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the data length in the WRITE call. |
| 10186 | A negative data length was specified for a WRITE or WRITEV call. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the data length in the WRITE call. |
| 10190 | The GETHOSTNAME option length is less than 24 or greater than the maximum length. | Disable the subtask for interrupts. Return an error code to the caller. | Correct the length parameter. |
| 10193 | The GETSOCKOPT option length is less than the minimum or greater than the maximum length. | End the call. | Correct the length parameter. |
| 10197 | The application issued an INITAPI call after the connection was already established. | Bypass the call. | Correct the logic that produces the INITAPI call that is not valid. |

**Table 4** Sockets Extended Error Return Codes (Continued)

| Error Code | Problem Description | System Action | Programmer's Response |
|---|---|---|---|
| 10198 | The maximum number of sockets specified for an INITAPI exceeds 2000. | Return to the user. | Correct the INITAPI call. |
| 10200 | The first call issued was not a valid first call. | End the call. | For a list of valid first calls, refer to the section on special considerations in the chapter on general programming. |
| 10202 | The RETARG parameter in the IOCTL call is zero. | End the call. | Correct the parameter list. You might have incorrect sequencing of socket calls. |
| 10203 | The requested socket number is a negative value. | End the call. | Correct the requested socket number. |
| 10205 | The requested socket number is a negative value. | End the call. | Correct the requested socket number. |
| 10208 | the NAMELEN parameter for a GETHOSTYNAME call was not specified. | End the call. | Correct the NAMELEN parameter. You might have incorrect sequencing of socket calls. |
| 10209 | The NAME parameter on a GETHOSTBYNAME call was not specified. | End the call. | Correct the NAME parameter. You might have incorrect sequencing of socket calls. |
| 10210 | The HOSTENT parameter on a GETHOSTBYNAME call was not specified. | End the call. | Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls. |
| 10211 | The HOSTADDR parameter on a GETHOSTBYNAME or GETHOSTBYADDR call is incorrect. | End the call. | Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls. |
| 10212 | The resolver program failed to load correctly for GETHOSTBYNAME or GETHOSTBYADDR call. | End the call. | Check the JOBLIB, STEPLIB, and LINKLIB data sets and rerun the program. |
| 10213 | Not enough storage is available to allocate the HOSTENT structure. | End the call. | Increase the use storage allocation for this job. |
| 10214 | The HOSTENT structure was not returned by the resolver program. | End the call. | Ensure that the domain name server is available. This can be a nonerror condition indicating that the name or address specified in a GETHOSTBYADDR or GETHOSTBYNAME call could not be matched. |
| 10215 | The APITYPE parameter on an INITAPI call instruction was not 2 or 3. | End the call. | Correct the APITYPE parameter. |

**Table 4** Sockets Extended Error Return Codes (Continued)

| Error Code | Problem Description | System Action | Programmer's Response |
|---|---|---|---|
| 10218 | The application programming interface (API) cannot locate the specified TCP/IP. | End the call. | Ensure that an API that supports the performance improvements related to CPU conservation is installed on the system and verify that a valid TCP/IP name was specified on the INITAPI call. This error call might also mean that EZASOKIN could not be loaded. |
| 10219 | The NS parameter is greater than the maximum socket for this connection. | End the call. | Correct the NS parameter on the ACCEPT, SOCKET or TAKESOCKET call. |
| 10221 | The AF parameter of a SOCKET call is not AF_INET. | End the call. | Set the AF parameter equal of AF_INET. |
| 10222 | the SOCTYPE parameter of a SOCKET call must be stream, datagram, or raw (1, 2, or 3). | End the call. | Correct the SOCTYPE parameter. |
| 10223 | No ASYNC parameter specified for INITAPI with APITYPE=3 call. | End the call. | Add the ASYNC parameter to the INITAPI call. |
| 10224 | The IOVCNT parameter is less than or equal to zero, for a READV, RECVMSG, SENDMSG, or WRITEV call. | End the call. | correct the IOVCNT parameter. |
| 10225 | The IOVCNT parameter is greater than 120, for a READV, RECVMSG, SENDMSG, or WRITEV call. | End the call. | Correct the IOVCNT parameter. |
| 10226 | Invalid COMMAND parameter specified for a GETIBMOPT call. | End the call. | Correct the IOVCNT parameter. |
| 10229 | A call was issued on an APITYPE=3 connection without an ECB or REQAREA parameter. | End the call. | Add an ECB or REQAREA parameter to the call. |
| 10300 | Termination is in progress for either the CICS transaction or the sockets interface. | End the call. | None. |
| 10331 | A call that is not valid was issued while in SRB mode. | End the call. | Get out of SRB mode and reissue the call. |
| 10332 | A SELECT call is invoked with a MAXSOC value greater than that which was returned in the INITAPI function (MAXSNO field). | End the call. | Correct the MAXSOC parameter and reissue the call. |
| 10999 | An abend has occurred in the subtask. | Write message EZY1282E to the system console. End the subtask and post the TRUE ECB. | If the call is correct, call your system programmer. |
| 20000 | An unknown function code was found in the call. | End the call. | Correct the SOC-FUNCTION parameter. |

**Table 4** Sockets Extended Error Return Codes (Continued)

| Error Code | Problem Description | System Action | Programmer's Response |
|---|---|---|---|
| 20001 | The call passed an incorrect number of parameters. | End the call. | Correct the parameter list. |
| 20002 | The CICS Sockets Interface is not in operation. | End the call. | Start the CICS Sockets Interface before executing this call. |

## 8.5.3 MUXNAT API Error Return Codes

Table 5 explains the error return codes specific to the MUXNAT Application Programming Interface (API).

**Table 5** MUXNAT API Error Return Codes

| Error Code | Description | Programmer Response |
|---|---|---|
| 3001 | Get Host Name Error. | Verify that the host name and port number are correct. |
| 3002 | Error on Return from ezacic08. | Internal error. Call support. |
| 3003 | e*Gate returned something other than an ACK after a send. | Verify that the MUX e*Way is properly configured. |
| 3004 | Timed out waiting for an ACK. | Increase hsces-for-ack value. If the problem persists, call support. |
| 3005 | Timed out waiting for a RECEIVE. | Increase hsecs-to-wait. If the problem persists, call support. |

# Index