CICS® Transaction Gateway

Programming

Version 3.1

SC34-5594-00

CICS® Transaction Gateway

Programming

Version 3.1

SC34-5594-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix B. Notices" on page 63.

First edition (September 1999)

This edition applies to Version 3.1 of CICS Transaction Gateway, program number 5648-B43.

Material that was not in the CICS Transaction Gateway Version 3.0 books is indicated by vertical lines to the left of that material.

Contents

About this book

This book provides an introduction to Java™ programming with the CICS Transaction Gateway. It describes the Java classes, Java beans, and programming samples that are provided, and also how to use VisualAge for Java to develop applets for accessing CICS transactions.

Who should read this book

This book is intended for anyone involved with programming a CICS Transaction Gateway.

It is assumed that you are familiar with the operating system under which your CICS Transaction Gateway runs.

An understanding of Internet terminology would also be helpful.

Conventions and terminology used in this book

References to paths in this book use the OS/2® and Microsoft® Windows® convention of a backslash (\) as delimiter, instead of the / delimiter used on the AIX and Solaris platforms.

Prerequisite and related information

The following sections list books relevant to CICS Transaction Gateway.

IBM CICS Transaction Gateway publications

For information on the books in the CICS Transaction Gateway and CICS Universal Clients library, refer to “Appendix A. The CICS Transaction Gateway and CICS Universal Clients library” on page 57. That chapter also gives details of how to view and print the softcopy books supplied with CICS Universal Clients and how to order printed copies from IBM.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book, or any other CICS documentation:

- Visit our Web site at:

http://www.ibm.com/software/ts/cics/

and follow the **Library** link to our feedback form.

Here you will find the feedback page where you can enter and submit your comments.

- Send your comments by e-mail to idrcf@hursley.ibm.com
- Fax your comments to:

  +44-1962-870229 (if you are outside the UK)
  01962-870229 (if you are in the UK)

- Mail your comments to:

  Information Development
  Mail Point 095
  IBM United Kingdom Laboratories
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN
  United Kingdom

Whichever method you use, ensure that you include:

- The name of the book
- The form number of the book
- If applicable, the version of the product
- The specific location of the text you are commenting on, for example, a page number or table number.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Obtaining books from IBM

For information on books you can download, visit our Web site at:
http://www.ibm.com/software/ts/cics/

and follow the **Library** link.

You can order hardcopy books:

- Through your IBM representative or the IBM branch office serving your locality.
- By calling 1-800-879-2755 in the United States.

- From the Web site at:
  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

Chapter 1. CICS Transaction Gateway programming interface overview

This chapter introduces the classes, interfaces, and Java Beans that make up the public programming interface of CICS Transaction Gateway.

- **Java client program classes**
  - com.ibm.ctg.client.JavaGateway
  - com.ibm.ctg.client.ECIRequest
  - com.ibm.ctg.client.EPIRequest
  - com.ibm.ctg.client.ESIRequest
  - com.ibm.ctg.client.CicsCpRequest
  - com.ibm.ctg.client.Callbackable (interface)
  - com.ibm.ctg.client.GatewayRequest

  For more information, see "Chapter 2. Writing Java client programs" on page 3.
- **Interface definitions and certificate objects for writing Gateway security classes**
  - com.ibm.ctg.security.ClientSecurity
  - com.ibm.ctg.security.ServerSecurity
  - com.ibm.ctg.security.SSLightServerSecurity
  - com.ibm.sslight.SSLCert[]
  - com.ibm.ctg.security.SystemSSLServerSecurity (OS/390 only)
  - com.ibm.gskssl.SSLCertificate (OS/390 only)
  - com.ibm.ctg.util.RACF.Userid (OS/390 only)

  For more information, see "Chapter 3. CICS Transaction Gateway security classes" on page 11
- **CICS Transaction Gateway EPI support classes**
  - com.ibm.ctg.epi.AID
  - com.ibm.ctg.epi.EPIGateway
  - com.ibm.ctg.epi.Field
  - com.ibm.ctg.epi.FieldData
  - com.ibm.ctg.epi.Map
  - com.ibm.ctg.epi.MapData
  - com.ibm.ctg.epi.Screen
  - com.ibm.ctg.epi.Terminal
  - com.ibm.ctg.epi.TerminalSession (interface)
  - com.ibm.ctg.epi.TerminalInterface
  - com.ibm.ctg.epi.Session (interface)
  - com.ibm.ctg.epi.EPIException

These classes are not available for CICS Transaction Gateway for OS/390.

For more information, see "Chapter 4. EPI support classes" on page 15.

- **CICS Transaction Gateway EPI bean classes**
  - com.ibm.ctg.epi.EPIBasicScreenHandler Class
  - com.ibm.ctg.epi.EPITerminal Class
  - com.ibm.ctg.epi.ScreenEvent Class
  - com.ibm.ctg.epi.ScreenEventListener interface
  - com.ibm.ctg.epi.ScreenHandler Class
  - com.ibm.ctg.epi.TerminalEvent Class
  - com.ibm.ctg.epi.TerminalEventListener interface

  These classes are not available for CICS Transaction Gateway for OS/390.

  For more information, see "Chapter 5. EPI beans" on page 25.

Note: Although it may appear that there is a greater emphasis on EPI programming (support classes, beans) compared to ECI programming in CICS Transaction Gateway, this is a natural consequence of the nature of these interfaces. For more information on the ECI and EPI see the *CICS Family: Client/Server Programming* book.

Online information on CICS Transaction Gateway classes and interfaces is provided in Hypertext Markup Language (HTML) format. For more information, see "Chapter 8. Java class reference information" on page 55.

Chapter 2. Writing Java client programs

This chapter provides an introduction to writing Java client programs for the CICS Transaction Gateway.

The CICS Transaction Gateway provides the following basic classes for writing Java client programs:

com.ibm.ctg.client.JavaGateway
This class represents the logical connection between a program and a CICS Transaction Gateway. You need a JavaGateway object for each CICS Transaction Gateway that you wish to talk to.

com.ibm.ctg.client.ECIRequest
This class contains the details of an ECI request to the Gateway.

com.ibm.ctg.client.EPIRequest
This class contains the details of an EPI request to the Gateway.

com.ibm.ctg.client.ESIRequest
This class contains the details of an ESI request to the Gateway.

com.ibm.ctg.client.CicsCpRequest
This class contains the details of the codepage used by the CICS Universal Client.

com.ibm.ctg.client.Callbackable
The asynchronous model supported by the CICS Transaction Gateway allows the use of callback objects. This interface defines the methods that a Callbackable object must provide.

com.ibm.ctg.client.GatewayRequest
This is the root class that all the different types of Gateway request are built from. A user program cannot create a GatewayRequest object.

Flow of program control

At the simplest level, the flow of program control needed to write a simple CICS Transaction Gateway Java client program is as follows:

1. The Java program creates and opens an instance of an com.ibm.ctg.client.JavaGateway object.
   - The default JavaGateway constructor creates a blank JavaGateway object. You **must** then set the correct properties in this object using the relevant set.. methods. The JavaGateway is then opened by calling the open method.

- Two other JavaGateway constructors exist that simplify the creation of a JavaGateway by setting the relevant properties and implicitly calling the open method for you. On return from a successful call to one of these constructors, the resultant JavaGateway is open and connected to the requested CICS Transaction Gateway.

2. The Java program creates an instance of one of the Gateway request classes containing the request that it wishes to make, that is:
   - A com.ibm.ctg.client.ECIRequest is created for an ECI request
   - A com.ibm.ctg.client.EPIRequest is created for an EPI request
   - A com.ibm.ctg.client.ESIRequest is created for an ESI request
   - A com.ibm.ctg.client.CicsCpRequest is created for querying the codepage of the CICS Universal Client it is connected through.
3. The Java program then flows the request to the CICS Transaction Gateway using the flow method of the JavaGateway object.
4. The Java program checks the return code of the flow operation to see whether the request was successful.
5. The program continues to create request objects and flow them through the JavaGateway object, as appropriate.
6. The Java program then closes the JavaGateway object.

The CICS Transaction Gateway also provides a sample program TestECI, and a sample applet TestEPI to illustrate the use of these classes.

Note: There is no TestESI sample.

Setting up the CLASSPATH

Before you write any Java client programs, you must update the CLASSPATH environment variable to include the jar files supplied with the CICS Transaction Gateway. You must include the path to the ctgclient.jar file, and the ctgserver.jar file is required on the CLASSPATH statement if you are going to use the local CICS Transaction Gateway. For example, for CICS Transaction Gateway for Windows NT®:

```
CLASSPATH = C:\Program Files\IBM\CICS Transaction
Gateway\classes\ctgclient.jar;C:\Program Files\IBM\CICS
Transaction Gateway\classes\ctgserver.jar
```

For more information on setting CLASSPATH, see the *CICS Transaction Gateway Administration* book for your platform.

TestECI

TestECI is a sample program that allows you to test the functionality of the CICS Transaction Gateway. With TestECI you can connect to a Gateway and then send one or more ECI requests to a CICS server. If you specify more than one CICS program on the server, all the programs are run as one extended Logical Unit of Work (LUW). You can run TestECI either as an application, or as an applet.

The source for TestECI is provided in the directory:

`samples\java\com\ibm\ctg\test`

Running TestECI as an application

When running TestECI as an application, parameters are passed in via the command line and output appears in the console.

The syntax is:

```
java com.ibm.ctg.test.TestECI [jgate=jgate_URL]
                              [jgateport=jgate_port]
                              [clientsecurity=client_security_class]
                              [serversecurity=server_security_class]
                              [server=cics_server]
                              [userid=cics_userid]
                              [password=cics_password]
                              [prog<0..9>=prog_name]
                              [commarea=comm_area]
                              [commarealength=comm_area_length]
                              [status]
                              [trace]
```

Where :

- *jgate_URL* is the URL of the Gateway to connect to.
- *jgate_port* is the TCP/IP port to connect to on jgate_server, if it was not specified as part of the *jgate_URL*.
- *client_security_class* is the name of the class to use to provide client-side security.
- *server_security_class* is the name of the class to use to provide server-side security.
- *cics_server* is the name of the CICS server to receive ECI requests.
- *cics_userid* and *cics_password* are the userid and password.
- *prog_name* is the name of a CICS server program. You can specify up to ten program names.
- *comm_area* is the initial value of the COMMAREA, if any.

- *comm_area_length* is the length of the COMMAREA to send to each CICS server program.
- `status` causes the program to query the status of all the known CICS servers.
- `trace` causes tracing information to be produced.

For example:

```
java com.ibm.ctg.test.TestECI jgate=myjgate.here.com server=mycics
commarea="Hello World" prog0=testprog prog1=testprog2 status
```

Running TestECI as an applet

When running TestECI as an applet you pass in parameters via <param> tags within the <applet> tag. Output appears in a text area on the browser running the applet.

The parameters are the same as those used when running TestECI as an application (see "Running TestECI as an application" on page 5).

```
<applet
.....
<param name="jgate" value="jgate_URL">
<param name="jgateport" value="jgate_port">
<param name="clientsecurity" value="client_security_class">
<param name="serversecurity" value="server_security_class">
<param name="server" value="cics_server">
<param name="userid" value="cics_userid">
<param name="password" value="cics_password">
<param name="progn" value=prog_name">
<param name="commarea" value="comm_area">
<param name="commarealength" value="comm_area_length">
<param name="status" value="yes">
<param name="trace" value="yes">
</applet>
```

If *jgate_URL* is not specified, the browser connects to the applet host.

Sample HTML to invoke TestECI as an applet is provided in testeci.html, which is located with the TestECI source.

TestEPI

TestEPI is a sample applet that allows you to test the functionality of the CICS Transaction Gateway. With TestEPI you can connect to a Gateway, and then send one or more EPI requests to a CICS server.

TestEPI uses two other classes: RequestDetails and EPIStrings, which are also provided.

The source for TestEPI, RequestDetails, EPIStrings, and testepi.html is provided in the directory:

```
samples\java\com\ibm\ctg\test
```

Running TestEPI

As with other applets, TestEPI may be invoked from a web browser or an appletviewer using its HTML file: testepi.html.

TestEPI has a scrolling list that contains two EPI requests:
- List CICS Servers
- Run Transaction CECI

The default is List CICS Servers.

Before pressing the **Execute** button, fill in the **CICS Transaction Gateway Host Name** and the **CICS Transaction Gateway Port Number**.

If you are about to run the transaction CECI, and do not wish to use the CICS client's default server, you may fill in the **CICS Server name**. Before executing the transaction CECI, you are recommended to fill in the CICS Server device type; if you do not do so, the CICS server may select an inappropriate model. To obtain the name of an appropriate CICS Server device type you should contact your CICS server administrator.

When you press the **Execute** button, TestEPI starts a new thread on which to service your request. This Thread traces its progress in the TextArea entitled: `EPI Request Execution Details`. The latest thread to finish will display its success or failure in the TextField entitled: `Latest EPI Request Result`

Each Thread started as a result of pressing the Execute button starts a new JavaGateway connection, and closes it before ending.

TestEPI may be recompiled with Gateway tracing active, by changing the line

```
T.setOn(false);
```

to

```
T.setOn(true);
```

See "Tracing in Java client programs" on page 9 for more information.

Turning this tracing on affects the Java client only and does not affect tracing on the CICS Transaction Gateway itself.

Using a browser and CICS Transaction Gateway on the same workstation

If you intend to use a browser and CICS Transaction Gateway on the same workstation, you must remove ctgclient.jar and ctgserver.jar from the CLASSPATH setting. If you do not remove them, you are likely to receive the following error when running TestECI and TestEPI as an applet:

```
ERROR: java.io.IOException:
CCL6664E: Unable to load relevant class to support the tcp protocol
```

The reason for the error is that Java searches the CLASSPATH environment variable before downloading classes across the network. If the required class is local, Java attempts to use it. However, using class files from the local file system breaks Java applet security rules, therefore an exception is raised.

Performance issues

There are several performance issues to be considered when running Java client applications. The Java Virtual Machine (JVM) allocates a fixed size of stack space for each running thread in an application or applet. Hence, there are two categories of space that Java allocates, and these are controlled by the following options of `java`, the Java byte-code interpreter:

-ss The Native Stack Size, allocated when running native JIT (Just-In-Time) compiled code.

-oss The Java Stack Size, allocated when running Java Bytecode

The Java virtual machine can allocate a varying size of heap space for each running application or applet:

-ms The initial Java heap size

-mx The maximum Java heap size

It is recommended that in the case of a machine running many CICS Java applications or applets, the Native Stack Size and Java Stack Size be restricted to 32KB. Also, the inital heap size and maximum heap size should be restricted to 32 KB and 64 KB respectively. To do this, you must specify the following options to the java interpreter:

```
-ss 32k -os 32k -ms 32k -mx 64k
```

Tracing in Java client programs

You can control tracing in Java client programs using:

- calls to the com.ibm.ctg.client.T class
- Gateway.T system properties

  For example:

  ```
  java -DGateway.T=on com.ibm.ctg.test.TestECI
  ```

  to specify full debug for TestECI. For more information on the use of system properties, refer to your Java documentation.

The trace levels that you can specify for the two methods are:

Trace level	com.ibm.ctg.client.T call	System property
Product	T.setOn (true / false)	Gateway.T.trace=on
Full Debug	T.setDebugOn (true / false)	Gateway.T=on
Method entry	T.setEntryOn (true / false)	Gateway.T.entry=on
Method exit	T.setExitOn (true / false)	Gateway.T.exit=on
Arbitrary lines	T.setLinesOn (true / false)	Gateway.T.lines=on
Exception stacks	T.setStackOn (true / false)	Gateway.T.stack=on
Timing	T.setTimingOn (true / false)	Gateway.T.timing=on

Chapter 3. CICS Transaction Gateway security classes

The CICS Transaction Gateway provides the following classes for implementing security:

com.ibm.ctg.security.SSLightServerSecurity
: All implementations of CICS Transaction Gateway server-side security classes that require the exposure of SSL Client Certificates must implement the **SSLightServerSecurity** interface.

  Users of the pure Java (SSLight) protocol handlers should implement this interface. SSLight is available on all CICS Transaction Gateway platforms.

com.ibm.ctg.security.SystemSSLServerSecurity
: All implementations of CICS Transaction Gateway server-side security classes that require the exposure of SSL Client Certificates, and use System SSL protocol handlers must implement the **SystemSSLServerSecurity** interface.

  Users of System SSL should implement this interface. System SSL is only available for CICS Transaction Gateway for OS/390®.

com.ibm.ctg.security.ServerSecurity
: Implementations of CICS Transaction Gateway server-side security classes that do not require the exposure of SSL Client Certificates should implement the **ServerSecurity** interface.

com.ibm.ctg.security.ClientSecurity
: All implementations of CICS Transaction Gateway client-side security classes must implement the **ClientSecurity** interface.

com.ibm.ctg.util.RACFUserid
: This class attempts to map an X.509 client certificate to a RACF userid. The certificate must already be associated with a valid RACF userid.

The **SSLightServerSecurity**, **SystemSSLServerSecurity**, or **ServerSecurity** interfaces and partner **ClientSecurity** interface define a simple yet flexible model for providing security when using the CICS Transaction Gateway. Implementations of the interfaces can be as simple or as robust as deemed necessary; from simple XOR (eXclusive-OR) scrambling to use of the Java Cryptography Architecture.

The SSLightServerSecurity or SystemSSLServerSecurity interfaces have been designed to work in conjunction with the Secure Sockets Layer (SSL) protocol. The interfaces will allow server-side security objects access to a Client

Certificate passed during the initial SSL handshake. The exposure of the Client Certificate is dependent on the the CICS Transaction Gateway being configured to support Client Authentication.

An individual JavaGateway instance has an instance of a ClientSecurity class associated with it, until the JavaGateway is closed. Similarly, an instance of the partner SSLightServerSecurity, SystemSSLServerSecurity, or ServerSecurity class is associated with the connected Java client, until the connection is closed.

The basic model consists of:

- An initial handshake to exchange pertinent information. For example, this handshake could involve the exchange of public keys. However, at the interface level the flow consists of a simple byte-array, therefore an implementation has complete control over the contents of its handshake flows.
- The relevant ClientSecurity instance being called to encode outbound requests, and decode inbound replies.
- In the CICS Transaction Gateway, the partner SSLightServerSecurity, SystemSSLServerSecurity, or ServerSecurity instance being called to decode inbound requests and to encode outbound replies. The inbound request, and Client Certificate, is exposed via the **afterDecode()** method. For SSLight, the afterDecode() method exposes the GatewayRequest object, along with the **com.ibm.sslight.SSLCert[]** certificate chain object. For System SSL, the afterDecode() method exposes the GatewayRequest object, along with the **com.ibm.gskssl.SSLCertificate** certificate object.

ClientSecurity and SSLLightServerSecurity,SystemSSLServerSecurity, or ServerSecurity class instances should maintain as data members sufficient information from the initial handshake to correctly encode and decode the flows.

An example implementation of the ClientSecurity interface is supplied in the com.ibm.ctg.security.clientCompression.java file.

An example implementation of the SSLLightServerSecurity interface is supplied in the com.ibm.ctg.security.SSLLightServerCompression.java file.

An example implementation of the SystemSSLServerSecurity interface is supplied in the com.ibm.ctg.security.SystemSSLServerCompression.java file.

An example implementation of the ServerSecurity interface, which does not expose an SSL client certificate, is supplied in the com.ibm.ctg.security.ServerCompression.java file.

You can find the source for all of these examples in the \samples\java\com\ibm\ctg\security directory.

Chapter 4. EPI support classes

Many existing CICS server applications are written for 3270 terminal interfaces and CICS has some powerful capabilities for dealing with these data streams, including Basic Mapping Support (BMS).

The External Presentation Interface (EPI) provides a mechanism for CICS Clients to communicate with transactions on a server and to handle 3270 data streams. A CICS client using EPI connects to CICS as if it were a 3270 terminal, and must send and receive 3270 data streams.

The CICS Transaction Gateway EPI support classes make it simpler for a Java programmer to access the facilities that the EPI provides:

- Connection of 3270 sessions to CICS servers
- Starting CICS transactions
- Sending and receiving 3270 data streams.

Furthermore, a detailed knowledge of 3270 data streams is not required. The classes provide higher-level constructs for handling 3270 data streams as follows:

- General purpose Java classes for handling 3270 data streams, such as fields and attributes, and CICS transaction routing data, such as transaction ID.
- Generation of Java classes for specific CICS applications from BMS map source files. These classes allow client applications to access data on 3270 panels, using the same field names as used in the CICS server BMS application.

Note that these classes do not support Double Byte Character Set (DBCS) fields in 3270 data streams.

The BMS conversion utility is a tool for statically producing Java class source code from a CICS BMS mapset, see “Converting BMS maps and using the Map class” on page 20.

The EPI support classes are based on the CICS Client C++ EPI classes. C++ programmers who have used the CICS Client classes will find that the Java classes are very similar, apart from changes to class and method names.

The EPI support classes are as follows:

com.ibm.ctg.epi.AID
: Represents an AID (attention identifier) character to be sent to CICS.

com.ibm.ctg.epi.EPIGateway
Connects to the CICS Transaction Gateway. This class also has methods that obtain information on CICS servers accessible to the client.

com.ibm.ctg.epi.Field
Supports a single field on a virtual screen and provides access to field text and attributes.

com.ibm.ctg.epi.FieldData
Contains information about a single field, that is, its row and column position and length.

com.ibm.ctg.epi.Map
Provides access to Field objects using BMS map information. The BMSMapConvert utility generates classes derived from Map.

com.ibm.ctg.epi.MapData
Contains information about a BMS map, that is, its size, number of fields and so on.

com.ibm.ctg.epi.Screen
Each terminal Terminal object has a virtual screen associated with it. The Screen class contains a collection of Field objects and methods to access these objects. It also has methods for general screen handling.

com.ibm.ctg.epi.Terminal
Controls a 3270 terminal connection to CICS. The Terminal class handles CICS conversational, pseudo-conversational, and ATI transactions. One application can create many Terminal objects.

com.ibm.ctg.epi.TerminalInterface
The Terminal class implements this interface.

com.ibm.ctg.epi.TerminalSession (interface)
Defines the behavior of a terminal in session, that is, a terminal with an associated Session object. TerminalInterface extends this with functions that allow the Session object to be changed.

com.ibm.ctg.epi.Session (interface)
Controls communication with the server in synchronous and asynchronous modes. Application classes can implement the Session interface to handle replies from the server.

com.ibm.ctg.epi.EPIException
This exception may be thrown by the EPI classes in some error situations.

Using the EPI support classes

This section describes how to use the EPI support classes. Samples of code are given.

Connecting to CICS and starting a transaction

Before a terminal connection can be made to CICS, a connection to the CICS Transaction Gateway must be started by creating a JavaGateway object. The EPIGateway class provides methods to access information about CICS servers that are accessible to the CICS Transaction Gateway and can be used instead of the JavaGateway class if you wish.

To establish a 3270 terminal connection to CICS, a Terminal object is created. To start a transaction on the CICS server, use the send method on the Terminal object:

```
try {
    // Connect to CICS server
    Terminal terminal = new Terminal( jgate, "CICS1234", null, null );

    terminal.send( null, "CESN", null );
    ...

} catch ( EPIException exception ) {
    exception.printStackTrace();
}
```

Note the use of try and catch blocks to handle any exceptions thrown by the CICS classes.

Accessing fields on CICS 3270 screens

When a terminal connection to CICS has been established, the Terminal, Screen and Field objects are used to navigate through the screens presented by the CICS server application, reading and updating screen data as required.

The Screen object is created by the Terminal object and is obtained via the getScreen method on the Terminal object. It provides methods for obtaining general information about the 3270 screen (for example, cursor position) and for accessing individual fields (by row/column screen position or by index). The following example prints out field contents, then ends the CESN transaction by returning PF3:

```
// Get access to the Screen object
Screen screen = terminal.getScreen();

for ( int i=1; i <= screen.fieldCount(); i++ ) {
    Field field = screen.field(i); // get field by index
    if ( field.textLength() > 0 )
        System.out.println( "Field " + i + ": " + field.getText() );
}

// Return PF3 to CICS
screen.setAID( AID.PF3 );
terminal.send();

// Disconnect the terminal from CICS
terminal.disconnect();
```

The Field class provides access to the text and attributes of an individual 3270 field. You can use these in a variety of ways to locate and manipulate information on a 3270 screen:

```
for ( int i=1; i <= screen.fieldCount(); i++ ) {
    Field field = screen.field(i); // get field by index

    // Find unprotected (i.e. input) fields
    if ( field.inputProt() == Field.unprotect )
        ...
    // Find fields the same as a specific text string

    if ( field.getText().equals( "CICS Sign-on") )
        ...
    // Find red fields
    if ( field.foregroundColor() == Field.red )
        ...
}
```

Synchronization and sessions

The EPI classes support synchronous (blocking) and asynchronous (callback) protocols.

In the example above the default synchronization type (synchronous) is used. To use asynchronous calls, create a class that implements Session. An object of this type can be passed as the first parameter to the Terminal send method. Terminal calls getSyncType on this object to determine the type of synchronization required.

A synchronization type of sync means that the send method blocks until the CICS server has finished sending data. When the reply is received, updates are made to the Screen object according to the 3270 data stream received, then control is returned to the calling program.

A synchronization type of async means that when the call is made to CICS using the Terminal send method, control returns immediately to the client

application without waiting for a reply from CICS. When the Terminal object receives a reply from CICS, the handleReply method on the Session object is invoked.

The implementation of the handleReply method can process the screen data available in the Screen object, which will have been updated in line with the 3270 data stream sent from CICS:

```
void handleReply( Terminal term ) {
    // Check the state of the session
    switch( term.getState() ) {
    case Terminal.client:
    case Terminal.idle:
        // Output data from the screen
        for ( int i=1; i <= screen.fieldCount(); i++ ) {
            System.out.println( "Field " + i + ": " + screen.field(i).toString() );
        screen.setAID( AID.PF3 );
        ...
    } // end switch
}
```

The handleReply method is called for each transmission received from CICS. Depending on the design of the CICS server program, a Terminal send call may result in one or more replies. The Terminal state property indicates whether the server has finished sending replies:

Terminal.server
: Indicates that the CICS server program is still running and has further data to send. The client application can process the current screen contents immediately, or simply wait for further replies. The application can neither disconnect the terminal, send the screen to CICS, nor start a new transaction.

Terminal.client
: Indicates that the CICS server program is now waiting for a response. The client application should process the screen contents and send a reply. The application can neither disconnect the terminal, nor start a new transaction.

Terminal.idle
: Indicates that the CICS server program has completed. The client program should process the screen contents and either disconnect the terminal, or start a further transaction.

Most client applications will want to wait until the CICS server program has finished sending data (that is, the Terminal state is client, or idle) before processing the screen. However, some long-running server programs may send intermediate results or progress information that can usefully be accessed while the state is still server.

The implementation of the handleReply method can read and process data from the Screen object, update fields as required, and set the cursor position and AID key in preparation for the return transmission to CICS, then use the Terminal send method to drive the server application.

Note that the handleReply method is run on a separate thread when asynchronous calls are used.

Using Screen without Terminal

If you would like to use the Screen and Field classes to interpret 3270 data streams but would prefer to continue using the EPIRequest class, then you can do so:

1. Establish your session with CICS in the usual way and create a Screen object of the required size.
2. Pass your data stream to the analyze method in Screen.
3. You can then access the fields of the screen, set the AID to return and so on.
4. To prepare the data stream to return to CICS, call the format method in Screen.

Converting BMS maps and using the Map class

A large proportion of existing CICS applications use BMS maps for 3270 screen output. This means that the server application can use data structures corresponding to named fields in the BMS map rather than handling 3270 data streams directly. The EPI BMS conversion utility uses the information in the BMS map source to generate classes specific to individual maps, which allow fields to be accessed by their names.

The utility generates Java classes that applications can use to access the map data as named fields within a map object. A class is defined for each map, allowing field names and lengths to be known at compile time. The generated classes extend the class **Map**, which provides general functions required by all map classes.

Run the BMS map converter utility on the BMS source as follows:

```
java com.ibm.ctg.epi.BMSMapConvert -p package-name filename.BMS
```

The utility generates .java files containing the source for the map classes. Use the -p parameter to specify the package to put the new files into. This saves you having to edit the files to add the ″package″ statement.

After you have used the EPI BMS utility to generate the map class, use the base EPI classes to reach the required 3270 screens in the usual way. Then use the map classes to access fields by their names in the BMS map. The map classes are validated against the data in the current Screen object.

Using Map classes

The classes generated by the BMS Conversion Utility have the following features:

- The class name is derived from the map name in the BMS source.
- The class extends Map.
- Two constructors are provided. One constructor takes a Screen parameter and throws an EPIException, if the screen has not been produced by the relevant BMS map. The no argument constructor creates a Map that can be validated against a screen later by using the setScreen method.
- The method field provides access to fields in the map, using the BMS source field names (provided as constants within the class).

To use the generated Map class, create a Terminal and start a transaction as usual:

```
try {
   JavaGateway epi = new JavaGateway("jgate", 2006 );
   // Connect to CICS server
   Terminal terminal = new Terminal( epi, "CICS1234", null, null );
   // Start transaction on CICS server
   terminal.send( null, "EPIC", null );
```

In this example the server program uses a BMS map for its first panel, for which a map class ″MAPINQ1Map″ has been generated. When the map object is created, the constructor validates the screen contents with the fields defined in the map. If validation is successful, fields can then be accessed using their BMS field names instead of by index or position from the Screen object:

```
    MAPINQ1Map map = new MAPINQ1Map( terminal.getScreen() );
    Field field;
    // Output text from "PRODNAM" field
    field = map.field(MAPINQ1Map.PRODNAM);
    System.out.println( "Product Name: " + field.getText() );
    // Output text from "APPLID" field
    field = map.field(MAPINQ1Map.APPLID);
    System.out.println( "Applid : " + field.getText() );
} catch (Exception exception) {
    exception.printStackTrace();
}
```

BMS Map objects can also be used within the Session handleReply method.

For validation to succeed, the entire BMS map must be available on the current screen. A map class cannot therefore be used when some or all of the BMS map has been overlaid by another map or by individual 3270 fields.

EPI samples

The following samples are provides for EPI programming:

EPISample1

EPISample1 is a sample application illustrating basic use of the EPI classes. It connects to the CICS Transaction Gateway and creates a terminal, then starts the CESN transaction. A list of the fields present on the screen is output, then PF3 is sent back to the terminal to end the transaction. When the transaction has ended, the terminal is disconnected.

To run the sample, issue the command:

```
java com.ibm.ctg.epi.EPISample1 ctg-address cics-server
```

where *ctg-address* is the TCP/IP address of the CICS Transaction Gateway, and *cics-server* is the name of the CICS server to connect to.

The source for EPISample1 is in the

```
\samples\java\com\ibm\ctg\epi
```

directory.

EPISample2

EPISample2 is a simple applet that demonstrates:

- The use of the EPIGateway class to list all available servers
- the use of the Session interface for asynchronous calls.

To try the applet, use the applet viewer or a suitable browser to view the file episamp2.html in the directory:

```
\samples\java\com\ibm\ctg\epi
```

The source for EPISample2 is also in that directory.

Using the EPIRequest class

It is recommended that you use EPI beans or the CICS Transaction Gateway EPI support classes if you are writing programs using the EPI. However, if you intend to use the EPIRequest class, you should read this section.

When connecting to CICS using EPI, a Java application or applet is acting as a terminal to CICS. It is, therefore, important to be aware of the 3270 data streams that may flow from CICS, and the 3270 data streams that may be expected in reply. After an event has been returned to a Java application or applet, the EPIRequest object's **size** field indicates the size of the **data** array returned.

It is also important to be aware of the principles and restrictions governing EPI programming, and the fact that there may be minor differences in the working of the EPI code on different platforms. For example, if you are running a CICS Client on Windows NT, you will probably need to send Transaction identifiers in the **data** array of the EPIRequest object, rather than in the EPIRequest object's **Transid** field.

When getting events from CICS it is recommended that you use the EPI_WAIT option, and ensure that the EPIRequest object's **size** field is set to the maximum size of the 3270 data stream that CICS may return.

Generally, EPI programs written using the CICS Transaction Gateway should:

1. Open a connection to the Gateway.
2. Add a terminal.
3. Start a transaction.
4. Get an event until, either, the event received is an end transaction or a converse; or, the error received is severe.
5. If the event received is a converse, then send the reply and return to the get event loop.
6. If the event received is an end transaction, then delete the terminal and do a last get event to obtain the end terminal event.
7. Close the connection to the Gateway.

Chapter 5. EPI beans

This chapter describes the JavaBeans™ that you can use for EPI functions.

CICS Transaction Gateway EPI beans Overview

The CICS Transaction Gateway includes high-level EPI classes to allow you to easily write Java programs that access data from existing CICS 3270 applications.

However, the EPI beans, based on this function, go one step further—you can create applets and applications that access your existing CICS 3270 screens *without any programming at all.* Using any of the large number of new visual application builder tools, including VisualAge for Java, you can quickly and easily create new Java front-ends that can connect to CICS, run transactions, display data from 3270 screens, and send user input back to the server. Note that the EPI beans do not support DBCS fields in 3270 data streams.

There are now a large number of visual development tools that allow you to build Java applets and applications with JavaBeans. We have tested the EPI beans with VisualAge for Java and Sun's BDK Beanbox. The beans should work with any tool that provides comparable function.

To use the beans with a visual development tool, you may need to import or load the **ctgclient.jar** file into the tool. Refer to the tool's documentation to find out how to do this. Alternatively, you may need to include this file in the CLASSPATH setting.

If you are using VisualAge for Java, the information in "Getting started with VisualAge for Java" on page 29 may help.

Running EPIApplet

EPIApplet is a sample applet that uses the EPI beans. You may find it helpful to run it to get a quick idea of what can be done with the beans.

To run EPIApplet:

- From the command line, change to the directory where the CICS Transaction Gateway is installed.
- Enter the command:

```
cd samples\java\com\ibm\ctg\epi
```

for OS/2 and Windows, or

```
cd samples/java/com/ibm/ctg/epi
```

for AIX and Solaris.

- Edit the file epiapplet.html to set the parameters for the URL of the Gateway and the name of your CICS server.
- Start the Gateway if is not running.
- Run the JDK appletviewer against the file epiapplet.html.
- When the applet starts, select the **Connect** button to connect to CICS.

  The applet connects to CICS and tries to run the transaction CESN.
- A window should appear with a representation of the CICS terminal screen, and there will also be a panel of buttons labelled **Enter, Clear, PF1-PF24, PA1-PA3.**

  You can enter text in the terminal window, and you can use the buttons to drive the terminal.
- To disconnect from CICS, close the terminal window.

The source code for EPIApplet is supplied, and may be helpful if you want to program with the EPI beans, rather than using them in a visual builder tool.

Using the beans

Once you have loaded the beans into a suitable visual development tool, you are ready to start creating an applet that uses them.

Connecting to CICS

Before your applet can do anything else, it must connect to the CICS Transaction Gateway install a terminal at the CICS server. To do this, you need an EPITerminal bean. Set the properties of this bean to indicate the address of the Gateway you want to connect to and the name of the CICS server. To connect to CICS, you need to call the EPITerminal's **connect** method.

Starting a transaction

To start a transaction, you can call the EPITerminal's **startTran** method, having previously set the transaction property to the transaction ID you want to start. Some tools, such as VisualAge for Java, allow you to call the **setTransaction** method with a defined parameter when an event occurs. Alternatively, send an ActionEvent to EPITerminal with the action command set to the transaction ID that you want to start. The transaction property is set to that value and the transaction is started as soon as possible. ActionEvents are generated by buttons, menu items and entry fields (when you select **Enter**—the action command is the text).

Handling screens

Once you have started a transaction on the CICS server, CICS begins sending data back to the EPITerminal bean. When EPITerminal receives data from CICS, it sends a handleScreen event to all its registered event listeners. To handle the data sent by CICS, you create beans called screen handlers and connect them to the EPITerminal's handleScreen event.

The EPIBasicScreenHandler bean is an example of a screen handler. It handles screens by displaying the data in the form of text and entry fields that you can type into. To see what it looks like, run EPIApplet.

To handle specific screens differently, you need to create screen handlers for them. You can write your own or use one of the provided tools to generate screen handler beans automatically. The automatically generated screen handler beans give you access to some of the data in the screen in the form of bound properties. You then create your own user interface (from other beans) to allow those properties to be displayed and changed.

Sending data back to CICS

Once you have received a screen from CICS and viewed and edited it as required, you must send it back to CICS to continue processing. If you are using EPIBasicScreenhandler or an automatically generated bean, then this means:

- setting the screen handlers AID property to the required AID value
- calling the send method of the screen handler or EPITerminal

Another way to achieve the same result is to send an ActionEvent to the screen handler with the action command set to an AID value (for example: enter, PF3). The AID is then set to that value and the screen is sent to CICS. ActionEvents are generated by buttons, menu items and entry fields (when you select **Enter***mdash;the action command is the text).

Disconnecting from CICS

When your applet has finished, you should ensure that the EPITerminal's **disconnect** method is called to disconnect it from CICS. When you call the **disconnect** method, EPITerminal tries to disconnect the terminal. If the terminal is not in the right state to be disconnected, EPITerminal sends the handleScreen event to its registered event listeners, requesting that they exit any running transaction. The default behavior of EPIBasicScreenhandler and the generated screen handlers is to send the AID PF3 to the CICS server. If you know that this will not work for a particular transaction you are using, you should generate a screen handler for the relevant screen and customize it to ensure it can exit that screen.

Example

This example demonstrates the basic principles involved in using the beans. Following the steps below produces an applet similar to EPIApplet.

1. Using your preferred bean composer tool, create an EPITerminal, and look at its properties. Set the URL property to the address of the Gateway, for example: tcp://jbloggs.hursley.ibm.com:2006. The terminal settings property allows you to set further terminal settings, such as the name of your CICS server.
2. Create two buttons labelled **Connect** and **Disconnect**. Connect the **Connect** button's actionPerformed event to the **connect** method on EPITerminal. Similarly, connect the **Disconnect** button to the EPITerminal's **disconnect** method. You can now connect to the CICS server and disconnect again.
3. Create an EPIBasicScreenHandler. This is a visual part that is able to display data sent from CICS. Connect the EPITerminal's handleScreen event to the EPIBasicScreenHandler's handleScreen method. Depending on the tool you are using, check that the event data is passed to EPIBasicScreenHandler.
4. Connect the **Connect** buttons actionPerformed event to the **startTran** method on EPITerminal. Look at EPITerminal's properties and check that the transaction property is set to a transaction that you can run on the CICS server.
5. Create an EPIScreenButtons. This is a simple set of buttons that you can use to drive the terminal. Connect the EPIScreenButton's actionPerformed event to the EPIBasicScreenHandler's actionPerformed method. Depending on the tool you are using, check that the event data is passed to EPIBasicScreenHandler.
6. You should now be able to connect to CICS and run a transaction. The EPIBasicScreenHandler displays data sent from CICS and accepts user input. When you select one of the buttons, the screen is sent back to CICS. The screen is also sent if you select **Enter** while a text field has the input focus. Make sure you disconnect the terminal when you have finished with it.

Generating screen handler beans

You can automatically generate screen handler beans, from BMS definition files, see "Screen Handler beans" on page 34. These screen handler beans define bound properties for each labelled field in the corresponding BMS map so that you can easily map between data from CICS and user interface components.

To use the generated screen handlers you connect them to the EPITerminal's handleScreen event as above. Any number of screen handlers can be

connected to an EPITerminal. When the screen handler receives a new screen from CICS, it decides whether the screen is displaying the corresponding BMS map. If it recognizes the screen, it sends the **screenHandled** event. It also updates its bound properties with the new data from CICS.

Getting started with VisualAge for Java

This is not a VisualAge for Java user's guide—it should be helpful if you have not used VisualAge for Java before, but you also need to refer to the VisualAge for Java documentation.

Note: The examples given are for VisualAge for Java Version 2.0. There may be differences for other versions of VisualAge for Java.

Importing the CICS Transaction Gateway classes into VisualAge for Java

To use the CICS Transaction Gateway classes in VisualAge for Java, they must first be imported into the repository. From the Workbench menu, select **File—Import**. Type in a project name, for example: `CICS Transaction Gateway`, select **JAR file**, and select the **Next >** button. Select the file **ctgclient.jar** in the CICS Transaction Gateway **classes** subdirectory and import it.

VisualAge for Java may report some errors, but you can safely ignore these.

Once the file has been imported into VisualAge, a dialog titled Add to Palette appears. Select all the class names listed, then select **OK**. You are prompted to type in a name for the new category folder —something like `CICS Transaction Gateway EPI beans`. Select **OK** to continue.

The Demo applet

We supply two demonstration applets that you can also import into VisualAge for Java. Select **File—Import** again, and type a new project name, such as `Demo`. Select **Interchange File**. You must import the file vajdemo.dat from the CICS Transaction Gateway samples\java\com\ibm\ctg\epi subdirectory. The contents of the interchange file are imported into the repository, not the Workbench. To add the Demo project to the Workbench, go to the Projects page of the Workbench and select the **Add project** button on the toolbar. Select **Add project from the repository** and **Browse**. Select the project named **Demo** from the list. The Editions list should contain one edition - select it and select **>>** to move it to the list of editions to add. Now select **OK** and **Finish** to add the project to the Workbench.

Now you can have a look at the Demo applets. From the Projects page of the Workbench, expand the Demo project and the demo package within it. There are two applets, called Demo and Demo2. Right-click on **Demo**, select **Open**

To -> Visual Composition. The Visual Composition Editor starts, and you can see how the Demo applet is assembled.

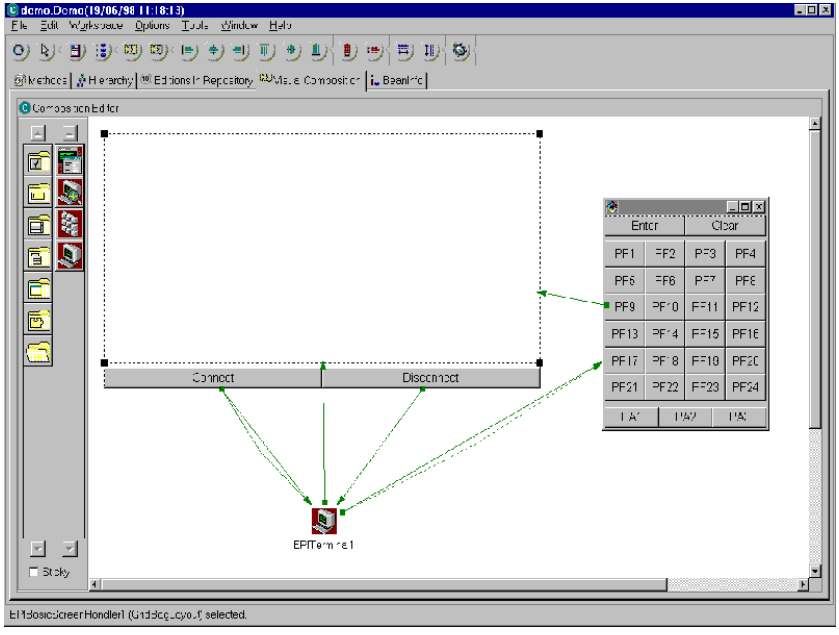

Figure 1. Demo applet

To run Demo, right-click on the EPITerminal bean in the display area, and select **Properties**. The properties of a bean are the settings you can change to alter its behavior. In this case, the EPITerminal bean must know how to connect to the CICS Transaction Gateway and your CICS server. In the Properties window, set the Gateway URL to the address of the CICS Transaction Gateway. This is normally of the form tcp://<tcp/ip name>:2006. Select the **...** button by the label **terminal settings** and set the name of your CICS server, as defined to the CICS Transaction Gateway. Check that the transaction property is a valid transaction ID for your CICS server.

Now select the **Test** button on the toolbar—it is the first one. VisualAge for Java generates the source code for the Demo applet, then another dialog appears. Select **OK**, and the applet should start. Select the **Connect** button to connect to CICS. You should see a CICS screen displayed. You can enter text, and use the **AID** button panel to drive the terminal. Select **Disconnect** to disconnect from CICS.

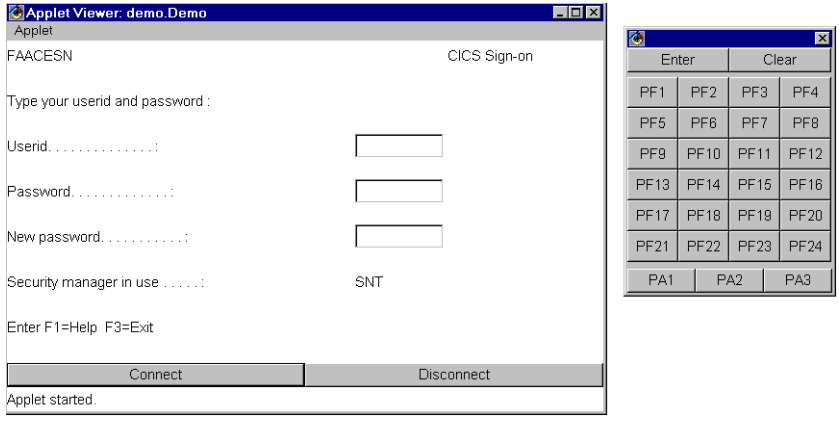

Figure 2. Demo applet running

Now go back to the Visual Composition Editor to see how the applet has been put together. It uses the EPITerminal, EPIBasicScreenHandler and EPIScreenButtons beans. The links between the beans are what make the applet work. The important ones are:

- The **Connect** button's **actionPerformed** event is connected to the **connect** method on EPITerminal. This means that when you select the button, EPITerminal connects to CICS.
- The **Connect** button's **actionPerformed** event is also connected to the **startTran** method on EPITerminal. This means that after connecting to CICS, EPITerminal starts the transaction that is set as its transaction property.
- EPITerminal's **handleScreen** event is connected to EPIBasicScreenHandler's **handleScreen** method. This means that when data arrives from the CICS server, it is passed to EPIBasicScreenHandler, which displays it as you have just seen.
- EPIScreenButton's **actionPerformed** event is connected to the **actionPerformed** method on EPIBasicScreenHandler. This means that when you select one of the buttons, EPIBasicScreenHandler is told what AID to send to CICS.
- Finally, the **Disconnect** button is connected to the **disconnect** method on EPITerminal

The Demo2 applet

The EPIBasicScreenHandler bean can display any screen sent from CICS, and allows you to edit fields, position the cursor, send different AID keys and so on. But what if you want to design your own user interface? Demo2 shows you the basic principles.

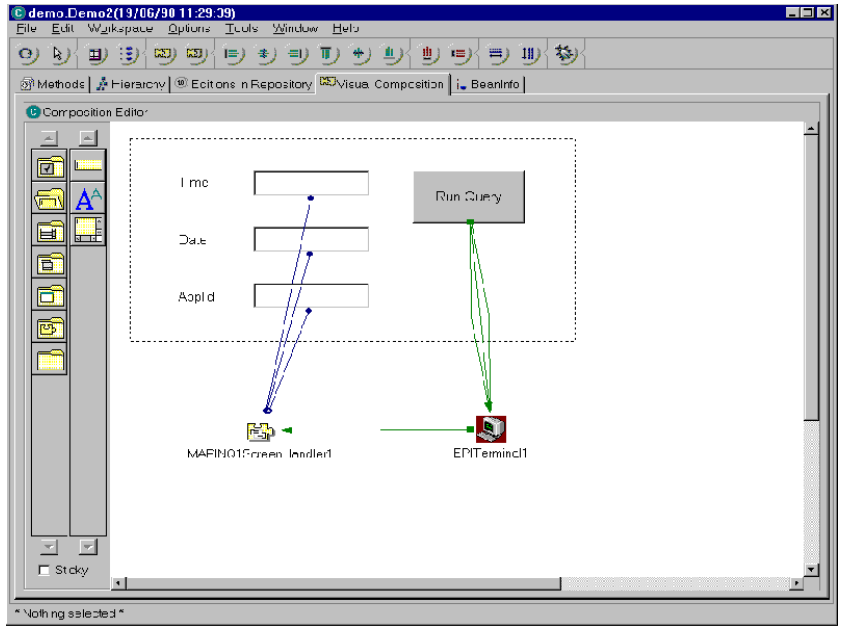

Figure 3. Demo2 applet

Demo2 uses the EPIINQ sample transaction that comes with the CICS Transaction Gateway. The source code for the transaction and the BMS map definition file can be found in the CICS Transaction Gateway **samples\server** subdirectory. You may want to compile this sample and set it up on your CICS server. It also uses the bean MAPINQ1ScreenHandler that is included in the Demo project. This was generated automatically from the EPIINQ.BMS file, using the BMS Map Conversion utility.

In Demo2, EPITerminal is used as before. The **Run Query** button is connected to the methods **connect**, **startTran** and **disconnect** on EPITerminal. The order of the connections does matter! MAPINQ1ScreenHandler replaces EPIBasicScreenHandler; it is connected to the **handleScreen** event from EPITerminal in the same way. MAPINQ1ScreenHandler defines properties that are in fact particular fields on the CICS screen —they are the named fields from the MAPINQ1 BMS map. Each of the properties we want to see is connected to the text property of an entry field.

When the screen created by the MAPINQ1 BMS map arrives from CICS, the MAPINQ1ScreenHandler bean recognizes it, and the entry fields connected to it are updated with the contents of the associated fields from the screen.

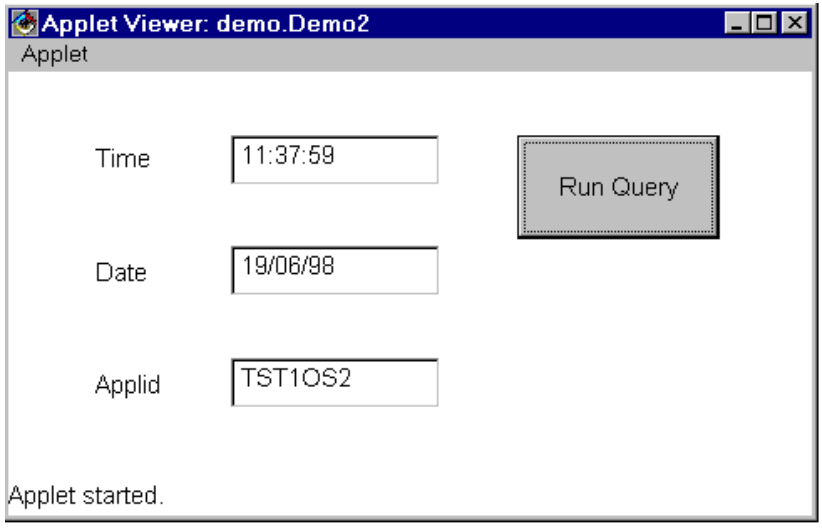

Figure 4. Demo2 applet running

Hints and Tips

This section provides some hints on how to make better use of EPI beans.

My applet does not work—and some of my connections are shown with dotted lines

An event can be connected to a method simply in order to make something happen, for example: connect to CICS when a button is selected. But sometimes events are used to pass information between two beans, for example: the **handleScreen** event from EPITerminal passes the screen data received from CICS to the EPIBasicScreenHandler—or MAPINQ1ScreenHandler. When you connect an event to a method, VisualAge for Java assumes you do not want to pass the event data to the method. If the method could be passed some data, the connection is shown with a dotted line. To make VisualAge for Java pass the event data to the method, right-click on the connection and select **Properties**. In the Event-to-Method Properties dialog, make sure the checkbox labelled **Pass event data** is checked. Now the connection should be shown with a solid line.

How do I set a Screen Handler property with an entry field ?

The **text** property of an entry field is not a bound property, so you cannot just connect it to the Screen Handler property. Instead, you must call the **setXXX** method on the Screen handler when you want the field to be set. Use an **actionPerformed** event as the trigger (from a button or the entry field itself). Connect the **actionPerformed** event to the Screen Handler's **setXXX** method.

Now connect the **text** property on the entry field **to the connection** you have just made. A menu should pop up with one item in it—the parameter to the method **setXXX**. Select it, and there should be a link from the entry field to the connection.

Adding generated Screen Handler beans to the Visual Composition Editor

First, import the generated files into VisualAge for Java—both the Map and ScreenHandler classes. You should have created the files in a package (using the -p parameter to BMSMapConvert). If you forgot, copy the files into a package (not the Default package!) Now, from the Visual Composition Editor menu, select **Options—Add bean**. Type in the first few letters of your beans name—you do not have to specify the package name. Select **Browse**, and from the list of classes select your new bean class. Select **OK**, and you can now drop the bean onto the display area.

Screen Handler beans

A screen handler bean is a Java class that can:

- Recognize a CICS screen
- Make information from the screen available as bean properties
- Exit the screen

Screen handlers can be used with the EPI beans to create Java front-ends to existing CICS 3270 applications. They are also used by the CICS Transaction Gateway Terminal Servlet both to exit transactions and to allow fields on the screen to be accessed by symbolic names.

You do not have to write screen handler classes—they can be generated automatically from BMS map definitions.

Generating screen handler beans

To generate screen handler classes, you use the BMS Map Conversion utility supplied with the CICS Transaction Gateway. For example, to generate beans for the screens defined by the BMS map file called TEST1.BMS, you use the command:

```
java com.ibm.ctg.epi.BMSMapConvert
-b test1.bms
```

The parameters you can supply to the BMS Map Conversion tool are (in any order):

-b to generate screen handler beans. The default is not to generate beans.

-p *package name*
to generate the Java source code with the statement package *package name*, for example `-p testing.beans`. The default is not to add a package statement.

-k *exit* to set the key that the screen handler will use to exit the screen, for example, `-k clear`. The default is PF3.

BMS map definition filenames
to select the BMS files to convert. Use your platform's standard filename conventions. Files are assumed to have the filetype .BMS if you do not give the full filename.

The BMS Map Conversion tool generates two Java source files for each map defined in a BMS file:

- A Map class
- A ScreenHandler class

The names of the source files are derived from the map name, for example, if a map called MAP1 is defined in the BMS file, then Java source files called MAP1Map.java and MAP1ScreenHandler.java would be produced.

The ScreenHandler class uses the Map class, so keep these files together.

If you are going to load the files into VisualAge for Java generate them with package names, otherwise they may not be recognized as beans.

Customizing and writing Screen Handlers

You should customize the generated ScreenHandler classes if:

- The screen handler cannot exit the screen it is associated with simply by sending an AID, such as PF3, to CICS. In this case, you will need to change the **exitScreen** method to do whatever is appropriate.
- You want to change the way the screen handler recognizes the screen it is associated with. Normally, it checks that the screen contains the same fields that were defined by the BMS map.

If you want to write your own screen handlers for some reason, you should bear the following points in mind:

- They must be Java beans. In general, this means they should have a constructor that takes no arguments, and they should implement the java.io.Serializable interface.
- Generated screen handlers extend the class ScreenHandler. This is not necessary; in fact for use with the EPI beans, screen handlers need only provide a method that accepts a TerminalEvent parameter.

For use with the Terminal Servlet, screen handlers should implement the TerminalEventListener interface and should use the TerminalEvent **isHandled** method to signal that they have recognized a screen.

EPI beans reference

This section provides reference information on the EPI beans:

EPITerminal bean

This bean acts as a 3270 terminal connected to the CICS server through the CICS Transaction Gateway.

To use this bean:

- Set the bean properties to appropriate values—you need to at least set the URL of the CICS Transaction Gateway.
- Connect screen handler beans to the terminal handleScreen event.
- Call the connect method to make the terminal connect to CICS.
- Start a transaction by setting the transaction property to the transaction ID then calling startTran.
- Make sure that the terminal is disconnected before your applet or application ends, by calling disconnect or terminate.

Properties

Gateway URL
The URL of the CICS Transaction Gateway to connect to, for example: setting the URL to: tcp://buster:2006 would result in the terminal trying to connect to a CICS Transaction Gateway at the TCP/IP address buster and the port number 2006.

Gateway Client Security class
The client security class that is used to connect to the CICS Transaction Gateway. This is an expert property.

Gateway Server Security class
The server security class that is used by the CICS Transaction Gateway. This is an expert property.

terminal settings
The terminal settings are: the name of the CICS server, the terminal model definition, and the terminal netname. In general, you only need to set the CICS server name. The default server is used if you do not set the server name. Changing these values while the terminal is connected to the server has no effect. You must disconnect and reconnect the terminal to use different settings.

ATI enabled
Set this property to **true** if you want to allow the CICS server to start transactions on the terminal.

transaction
A transaction ID. This is the transaction that is started when the method startTran is called.

transactionData
Transaction parameters. When startTran is called this string is passed as a parameter to the transaction, for example: if the transaction ID is CESN and the transaction data is set to `USERID=fred PS=pswd` then the effect of calling startTran would be the same as typing `CESN USERID=fred PS=pswd` at a CICS terminal.

timeout
The terminal timeout interval in milliseconds. If the terminal is idle (no data received from CICS) for longer than this period, the terminal attempts to disconnect from CICS. You can set this value to 0 if you do not want the terminal to time out. You should ensure that this timeout value is less than the CICS Transaction Gateway connection timeout value, otherwise the terminal may lose its connection to the CICS Gateway for Java without being able to disconnect from CICS.

connected
This boolean value is set to true if the terminal is connected to CICS.

Events

TerminalEvent

terminalConnected
when the terminal has connected to CICS.

terminalDisconnected
when the terminal has disconnected from CICS.

handleScreen
when the terminal receives a screen from CICS.

exceptionOccurred
if an exception occurs.

Methods

connect()
Connects the terminal to the server, if not already connected.

startTran()
Starts a transaction on the terminal using the transaction ID and transaction data already set.

send() Sends the current screen to CICS.

disconnect()
Disconnects the terminal from the server. The terminal cannot disconnect from the server while a transaction is running. If a transaction is running the terminal attempts to end it before disconnecting. The terminal remains connected to the CICS Transaction Gateway until the terminate method is called or the application ends.

terminate()
Waits for the terminal to finish disconnecting, then closes the connection to the CICS Transaction Gateway.

actionPerformed(ActionEvent e)
Sets the transaction ID to the event's action command, then calls the **startTran** method. Can be used to make a button push (or other ActionEvent) trigger the starting of a transaction.

EPIBasicScreenHandler bean

This bean is a simple screen handler that can deal with any screen. It can:

- Display screens sent from CICS using Java user interface components
- Accept user input
- Send the screen back to CICS with the user's changes
- Set the AID to be sent to CICS
- Set the screen cursor position.

To use this bean:

- Add it to your user interface—it can be treated like a Panel.
- Connect the terminal handleScreen event to the handleScreen method. If you are using VisualAge, make sure that the event data is passed to this method.

Properties

AID the AID key that is sent to CICS

exit AID
the AID key used to exit a running transaction

handling
true if the EPIBasicScreenHandler is handling the screen

minimum width
the minimum width of the screen panel

minimum height
the minimum height of the screen panel

Events

ScreenEvent

screenHandled
when the screen handler starts handling screens

screenUnhandled
when the screen handler stops handling screens

Methods

terminalConnected(TerminalEvent evt)
No action is taken when the terminal connected event is received

terminalDisconnected(TerminalEvent evt)
Clears the display

handleScreen(TerminalEvent evt)
Displays the screen

actionPerformed(ActionEvent evt)
If the source of the event is a button, the button action command is mapped to the screen AID if possible, and the screen is sent to CICS. If the source of the event is a TextField, the associated screen field is updated, the AID is set to **enter**, and the screen is sent to CICS.

send( )
Sends the screen to CICS

EPIMonitor bean

This bean is a simple status monitor for an EPITerminal. It handles terminal events by displaying a status message. You may find it useful while developing to keep track of what the terminal is doing.

To use this bean:

- Add it to your user interface.
- Connect the terminal events—terminalConnected, terminalDisconnected, handleScreen and exceptionOccurred—to the appropriate EPIMonitor methods; if you are using VisualAge, make sure that the event data is passed.

Methods

terminalConnected(TerminalEvent evt)
Updates the status message.

terminalDisconnected(TerminalEvent evt)
Updates the status message.

handleScreen(TerminalEvent evt)
Updates the status message.

exceptionOccurred(TerminalEvent evt)
Sets the status message to the exception text.

EPIScreenButtons bean

This bean is a set of buttons that can be used to send AIDs to the EPIBasicScreenHandler. Each button's action command is set to an AID value, for example: the button labelled **Enter** has the action command **enter**. When you select a button, the EPIScreenButtons bean forwards the button's action event to its own listeners.

To use this bean:

- Add it to your user interface.
- Connect the actionPerformed event to the EPIBasicScreenHandlers actionPerformed method—if you are using VisualAge for Java, make sure that the event data is passed.

Any screen handler based on ScreenHandler responds to action events from EPIScreenButtons, but you can also use your own buttons—any button with an action command equivalent to an AID value has the same effect.

Events

ActionEvent

actionPerformed
When a button is selected.

Methods

actionPerformed(ActionEvent evt)
Forwards the event to any registered listeners.

Chapter 6. CICS Transaction Gateway programming samples

This chapter summarizes the programming samples provided with CICS Transaction Gateway

The samples listed in the following sections are provided on each platform.

EPI samples

The following samples are provided in the samples\java\com\ibm\ctg\epi directory:

DefaultScreenHandler.java
A Default screen handler.

epiapplet.html
Applet to demonstrate use of EPI Beans, It performs simple terminal emulation using the EPIBasicScreenHandler bean.

EPIApplet.java
Source code for the EPI Beans applet.

EPIMonitor.java
Demonstrates a simple way to generate a status message to tell the user what the terminal is doing.

EPISample1.java
Demonstrates basic functions of the CICS Transaction Gateway EPI support classes; namely how to connect to CICS Transaction Gateway, create a terminal, run the CESN transaction, end the transaction, and disconnect the terminal.

episamp2.html
Applet that demonstrates the use of the EPIGateway class to list all available servers, and the use of the Session interface for asynchronous calls.

EPISample2.java
The source code for the EPISample2 applet.

EPIScreenButtons.java
A simple “Control panel” that you can use to send appropriate ActionEvents to a ScreenHandler such as EPIBasicScreenHandler.

vajdemo.dat
Interchange file for the Demo and Demo2 EPI bean applets.

Security samples

The following samples are provided in the samples\java\com\ibm\ctg\security directory:

ClientCompression.java
Demonstrates compression of the client/server dataflows using the java.util.zip package.

ServerCompression.java
Demonstrates compression of the client/server dataflows using the java.util.zip package.

clientDesign.java
Demonstrates PublicKey cryptography.

SSLightServerCompression.java
Demonstrates compression of the client/server dataflows using the java.util.zip package, and exposes an SSL client certificate. This sample is for SSLight users.

SystemSSLServerCompression.java
Demonstrates compression of the client/server dataflows using the java.util.zip package, and exposes an SSL client certificate. This sample is for System SSL users (on OS/390 only)

serverDesign.java
Demonstrates PublicKey cryptography.

Terminal Servlet samples

The following samples are provided in the samples\java\com\ibm\ctg\servlet directory:

epissam.html
Instructions on the use of the Terminal Servlet samples.

epissam1.shtml
Demonstrates how to use the servlet as a replacement for a CICS terminal.

epissam2.shtml
Demonstrates how to use server-side includes to display information from a CICS screen and to drive the terminal.

epissam3.html
Demonstrates how to use an HTML template file.

episssam4.shtml
Demonstrates how to display information that might be useful to an administrator.

servlet.properties
Sample properties file for the Terminal Servlet.

For more information about the Terminal Servlet, refer to the *CICS Transaction Gateway Administration* book for your platform.

Test samples

The following samples are provided in the samples\java\com\ibm\ctg\test directory:

EPIStrings.java
A class required by the TestEpi applet.

RequestDetails.java
A class required by the TestEpi applet.

testeci.html
An applet that demonstrates basic CICS Transaction Gateway functionality. It allows connection to a Gateway, and sending of one or more ECI requests to a CICS server.

TestECI.java
Source code for the TestEci applet.

testepi.html
An applet that demonstrates basic CICS Transaction Gateway functionality. It allows connection to a Gateway, and sending of one or more EPI requests to a CICS server.

TestEPI.java
Source code for the TestEpi applet.

For more information on TestECI and TestEPI see “Chapter 2. Writing Java client programs” on page 3.

Chapter 7. Using VisualAge for Java

This section describes how you can use VisualAge for Java to build a simple applet for accessing CICS transactions. It describes how to set up VisualAge for Java to use the CICS Transaction Gateway classes, how to use the VisualAge for Java Visual Composition editor to construct an applet, and how to use the applet outside the VisualAge for Java environment.

The examples given are for VisualAge for Java Version 2.0 Entry Version. There may be differences for other versions of VisualAge for Java.

VisualAge for Java and the CICS Transaction Gateway classes

To build CICS applications or applets with VisualAge for Java, you must import the class package, ctgclient.jar, that is included with the CICS Transaction Gateway The package contains the Java classes and beans necessary for building your applet.

To import the CICS Transaction Gateway class package into VisualAge for Java:

1. Start VisualAge for Java and go to the workbench.
2. Select **Import** from the **File** menu.
3. Select the **Jar file** button in the SmartGuide Import dialog and select **Next**.
4. In the SmartGuide import from jar/zip file dialog, select the **Filename Browse** button and navigate to the directory containing the ctgclient.jar file.
5. Select the ctgclient.jar file and select **Open** to return to the SmartGuide import from jar/zip file dialog.
6. Ensure that the check boxes for **.class** and **resource** are checked.
7. In the **Project** field enter `CICS Transaction Gateway`. Figure 5 on page 46 shows the Smartguide import.

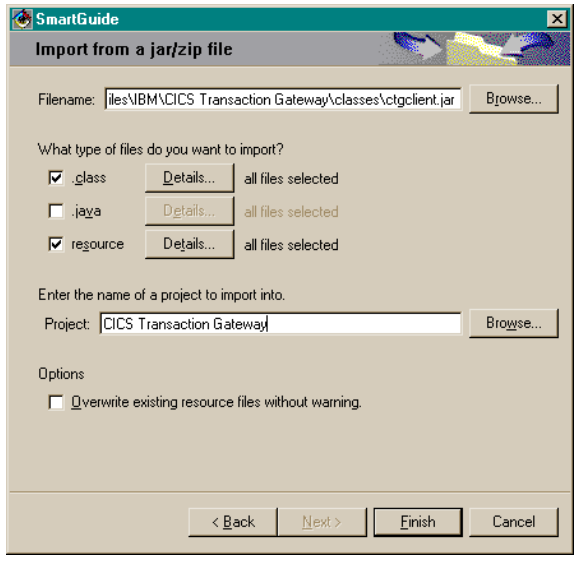

Figure 5. Importing CICS Transaction Gateway classes into VisualAge for Java

8. Select **Finish**. A question box appears stating that the CICS Transaction Gateway project does not exist and asks if you want to create it. Select **Yes**.

   The import of classes processing may report problems, but you may safely ignore these. When processing is complete, the Modify Palette dialog is displayed.

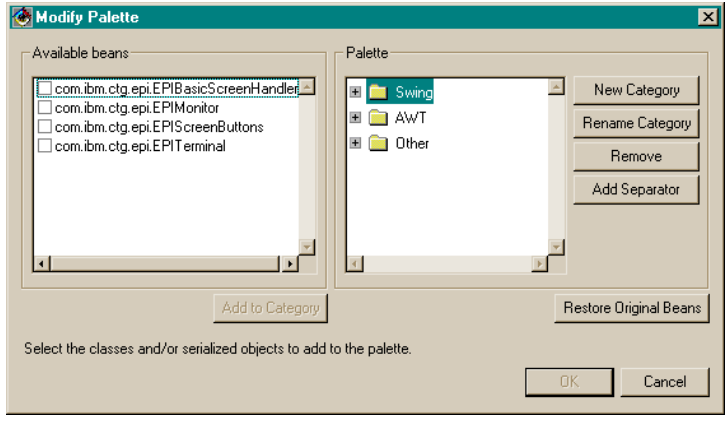

Figure 6. Modify Palette dialog

9. Select **New Category** and type CTG beans as name for the new category.

10. Ensure that all the available beans in the left panel are selected and that the new category, CTG beans, is selected.
11. Select **Add to Category**. Figure 7 shows the Modify Palette dialog including the CICS Transaction Gateway beans.

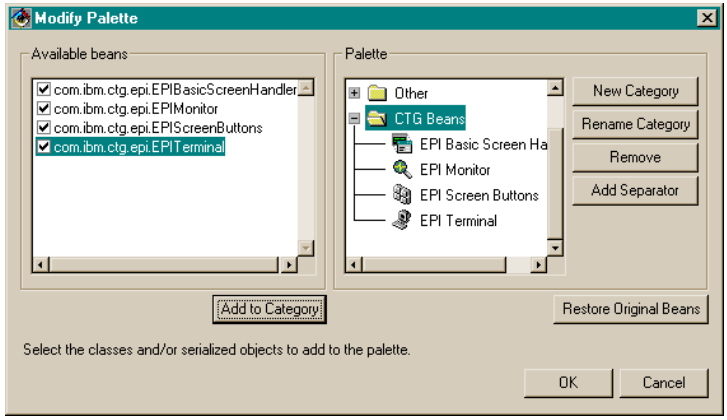

Figure 7. Modify Palette dialog and CICS Transaction Gateway beans

12. Select **OK**.

The CICS Transaction Gateway classes and beans are then imported and made available for program development in VisualAge for Java.

The newly created VisualAge for Java project, CICS Transaction Gateway, contains several samples with which you may want to experiment. See "Chapter 6. CICS Transaction Gateway programming samples" on page 41 for details on using the samples.

Note: If you are using the Enterprise edition of VisualAge for Java, you should either remove the **CICS Connector** project from the workspace, or import the CICS Transaction Gateway classes into that project instead. This avoids having the same classes at different levels in different projects in the workspace at the same time, which could cause problems.

Building an EPI Applet

To create an EPI applet, you can use the VisualAge for Java Quick Start feature to generate the framework of your applet. You can then use the VisualAge for Java Visual Composition editor to lay out an applet canvas, add buttons and CICS Transaction Gateway beans, configure the bean properties, and connect the buttons and beans.

Creating an Applet

To create a new applet:

1. From the **File** menu select **Quick Start** to display the Quick Start dialog.
2. In the left panel select **Basic** and in the right panel select **Create Applet**.
3. Select **OK** to open the SmartGuide Create Applet dialog.
4. In the **Project field**, enter a suitable name.
5. In the **Applet name** field, enter a name for your applet.
6. Select **Finish**.

When processing has completed, a blank applet canvas is displayed in the VisualAge for Java Visual Composition editor.

Creating an EPI Terminal

To add an EPI Terminal bean to your applet and configure the bean to access the CICS server:

1. Select **CTG beans** from the pull-down menu in the upper-left corner of the Visual Composition editor screen.
2. Select the icon labeled **EPI Terminal** on the left of the screen. (Move the mouse over the icons to display their labels.)
3. Move the mouse pointer to a position on the canvas that lies outside the rectangular outline so that the cursor changes to crosshairs. Click with the left mouse button to create an icon labeled **EPITerminal1** at that position.
4. Select the newly created **EPI Terminal** icon with the right mouse button and select **Properties** from the pop-up menu.
5. Modify the **GatewayURL** property to point to the host on which your CICS Transaction Gateway is running, for example, tcp://dilbert:2006
6. Select the right of the **terminal settings** input field to pop up a Terminal settings dialog. Type the name of your remote CICS server in the **CICS server name** field and then select **OK**.

The EPI Terminal bean, EPITerminal1, is now configured.

Creating a Logon Button

To add a logon button bean to the applet and connect the button bean to the EPI Terminal bean:

1. Select **AWT** from the pull-down menu in the upper left-corner of the Visual Composition editor screen.
2. Select the icon labeled **Button** on the left of the screen. (Move the mouse over the icons to display their labels.)
3. Move the mouse pointer to a position on the canvas that lies inside the rectangular outline so that the cursor changes to crosshairs. Click with the left mouse button to create a button bean labeled **Button1** at that position.
4. Select the newly created button bean with the right mouse button and select **Properties** from the pop-up menu.
5. Change the button bean label property to **Logon** and close the properties dialog.
6. Select the newly created button bean (now labeled **Logon**) with the right mouse button and select **Connect** and **actionPerformed** from the pop-up menu. The cursor changes to a spider attached, with a dotted line, to the **Logon** button.
7. Use the spider to select the **EPI Terminal** bean. Select **Connectable Features** from the pop-up menu. This opens the End connection to dialog.
8. In the End connection to dialog, select the **Method** button, located near the top of the display, select the **connect()** method in the panel, and select **OK**.

Creating an EPI Basic Screen Handler

To add an EPI Basic Screen Handler bean to the applet and connect it to the EPI Terminal bean:

1. Select **CTG beans** from the pull down menu in the upper left corner of the Visual Composition editor screen.
2. Select the icon labeled **EPI Basic Screen Handler** on the left of the screen. (Move the mouse over the icons to display their labels.)
3. Move the mouse pointer to a position on the canvas that lies inside the rectangular outline so that the cursor changes to crosshairs. Click near the top left corner with the left mouse button to create a new rectangular outline at the selected location. The rectangular outline represents the EPI Basic Screen Handler bean.
4. Select the **EPI Terminal** bean with the right mouse button and select **Connect** and **Connectable Features** from the pop-up menu. This opens the Start connection from dialog.
5. In the Start connection from dialog, select **Event** near the top of the screen, select **handleScreen** in the panel, and select **OK**.

6. The cursor changes to a spider attached, with a dotted line, to the Logon button. Use the spider to select the **EPI Basic Screen Handler** rectangle. Select **Connectable Features** from the pop-up menu. This opens the End connection to dialog.
7. In the End connection to dialog, select the **Method** button, located near the top of the display, select the **handleScreen(com.ibm.ctg.terminalEvent)** method in the panel, and select **OK**.
8. Using the right mouse button, select the dotted line between the EPI Basic Screen Handler rectangle and the EPI Terminal bean to open the Event-to-method connection - Properties dialog.
9. In the Event-to-method connection - Properties, select the checkbox **Pass event data** and select **OK**.

Connecting the Logon Button to EPI Terminal

The Logon button bean requires a second connection to the EPI Terminal so that the button can also start a transaction once a connection is established with the CICS server.

To create a second Logon button connection:

1. Select the **Logon button bean** with the right mouse button and select **Connect** and **actionPerformed** from the pop-up menu. The cursor changes to a spider attached, with a dotted line, to the Logon button.
2. Use the spider to select the **EPI Terminal bean**. Select **Connectable Features** from the pop-up menu. This opens the End connection to dialog.
3. In the End connection to dialog, select the **Method** button, located near the top of the display, select the **startTran()** method in the panel, and select **OK**.

Creating EPI Screen Buttons

To add an EPI Screen Buttons bean to your applet:

1. Select **CTG Beans** from the pull-down menu in the upper left corner of the Visual Composition editor screen.
2. Select the icon labeled **EPI Screen Buttons** on the left of the screen. (Move the mouse over the icons to display their labels.)
3. Move the mouse pointer to a position on the canvas that lies inside the rectangular outline so that the cursor changes to crosshairs. Select near the lower right corner with the left mouse button to create an EPI Screen Buttons bean at the selected location.
4. Select the EPI Screen Buttons bean with the right mouse button and select **Connect** and **Connectable Features** from the pop-up menu. This opens the Start connection from dialog.

5. In the Start connection from dialog, select the **Event** button, located near the top of the display, select **actionPerformed** in the panel, and select **OK**.
6. The cursor changes to a spider attached, with a dotted line, to the EPI Screen Buttons bean. Use the spider to select the **EPI Basic Screen Handler** rectangle. Select **Connectable Features** from the pop-up menu. This opens the End connection to dialog.
7. In the End connection to dialog, select **Method** near the top of the screen, select the **actionPerformed(java.awt.event.ActionEvent)** method in the panel, and select **OK**.
8. Using the right mouse button, select the dotted line between the EPI Basic Screen Handler rectangle and the EPI Screen Buttons bean to open the Event-to-method connection - Properties dialog.
9. In the Event-to-method connection - Properties, select the checkbox **Pass event data** and select **OK**.

Testing the Applet within VisualAge for Java

To test your applet from within VisualAge for Java:

1. Go to the workbench **Bean** menu and select **Run** and **In Applet Viewer**.
2. Resize the applet, by dragging the lower-left corner of the applet, until all components are visible.
3. Select the applet **Logon** button. The CESN transaction runs on the CICS server and the EPI Screen Handler applet displays the results.
4. Type in your Userid and Password for the CICS server and press Enter.
5. Select **Clear** to clear the screen and display a text input box.
6. Type in any CICS transaction name and operands and press Enter.
7. Resize the EPI Basic Screen Handler and rearrange the applet beans in the Visual Composition editor by dragging them with the mouse to make all the applet components visible in the Applet Viewer.

Exporting the Applet

When you have built the applet within VisualAge for Java, you must export it to use it outside VisualAge for Java.

You may experience a problem using the VisualAge for Java export function. The export function selects all of the CICS Transaction Gateway static classes required by the applet but omits the dynamic classes. Selecting the required dynamic classes by hand during export is one solution to this problem.

It is recommended that you export only your applet class, and copy the applet class, the HTML file generated by VisualAge for Java, and the CICS

Transaction Gateway ctgclient.jar file to your server. Although you get an applet distribution that is greater than 1 MB, you will be certain that any required classes are present.

To export and distribute your applet:

1. Go to the VisualAge for Java workbench, select the **Projects** tab, and navigate to the your applet class.
2. Select your applet class using the right mouse button. In the pop-up menu select **Export** to open the SmartGuide Export dialog.
3. In the SmartGuide Export dialog, select **Directory** and then **Next** to open the SmartGuide Export to a directory dialog (see Figure 8).

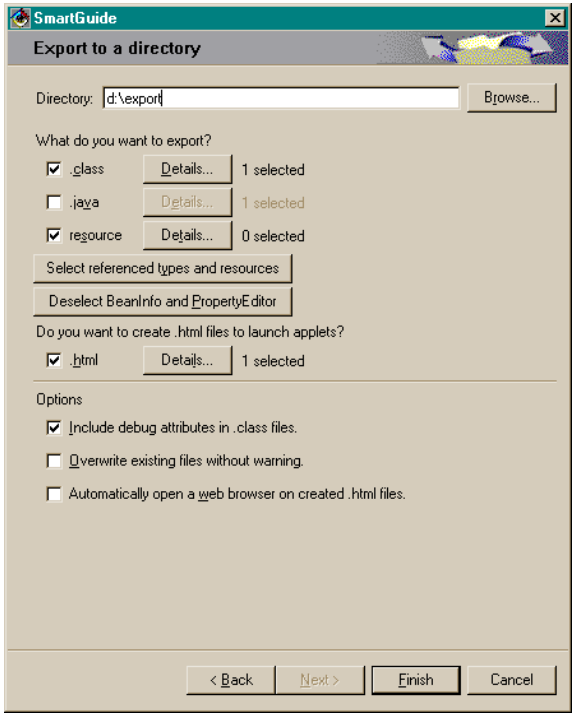

Figure 8. Export to a directory dialog

4. Now, select a suitable Directory, select **.class** and **.html**, and then select **Finish**. When asked if you want to create the directory you have selected, select **Yes**. VisualAge for Java then creates .class and .html files for your applet, and places them in the directory you specified.
5. Open the .html file in an editor, modify the applet width and height tags, and insert an archive tag in the applet tag. For example:

```
<HTML>
<HEAD>
<TITLE>MyApplet</TITLE>
</HEAD>
<BODY>
<H1>MyApplet</H1>
<APPLET CODE=MyApplet.class
        ARCHIVE=ctgclient.jar
        WIDTH=650
        HEIGHT=500>
</APPLET>
</BODY>
</HTML>
```

6. Copy the ctgclient.jar file to the directory on the CICS Transaction Gateway server you specified. This directory must be accessible by the server for serving HTML documents.

Note: File ctgclient.jar does not necessarily need to be placed in the same directory as the .class and .html files for your applet. This was shown only for simplicity.

Running the Applet

You run the applet you have built by pointing your browser to the URL of the .html file on your CICS Transaction Gateway server, for example:

```
http://azov.almaden.ibm.com/local/apache/share/htdocs/MyApplet.html
```

Chapter 8. Java class reference information

Online programming reference information is provided for the Java classes and interfaces provided with CICS Transaction Gateway.

The reference information is in HTML format and is generated using the Javadoc tool provided with the JDK.

To get to the reference information:

- On Windows and OS/2, select the **Documentation** icon.
- On AIX, Solaris, and OS/390, run the **ctgdoc** script.

and the library home page is displayed. You can then follow the links to the reference information.

The following sections describe the different kinds of HTML pages that are provided within the reference information.

Note: You may need to refer to the README file for the latest information on using the programming reference information.

Class/interface page

In the reference pages, each class and interface has its own page. In each of these pages, there are three sections:

1. Class/interface description:
   - Class inheritance diagram
   - Direct Subclasses
   - All Known Subinterfaces
   - All Known Implementing Classes
   - Class/interface declaration
   - Class/interface description
2. Summary tables:
   - Inner Class Summary
   - Field Summary
   - Constructor Summary
   - Method Summary
3. Class/interface description:
   - Field Detail

- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Use page

Each documented class and interface has its own Use page. This page describes what packages, classes, methods, constructors and fields use any part of the given class. The Use page for a package or interface A includes:

- Subclasses of A
- Fields declared as A
- Methods that return A
- Methods and constructors with parameters of type A.

You can access this page by first going to the class or interface, then clicking on the **Use** link in the navigation bar.

Tree (Class Hierarchy)

When viewing a particular class or interface page, selecting **Tree** displays the class and interface hierarchy for CICS Transaction Gateway.

Deprecated API

The Deprecated API page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements. Deprecated APIs may be removed in future implementations.

Index page

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Appendix A. The CICS Transaction Gateway and CICS Universal Clients library

This chapter lists all the CICS Transaction Gateway, CICS Universal Clients, and related books, and discusses the various forms in which they are available.

The headings in this chapter are:

CICS Transaction Gateway books

- *CICS Transaction Gateway for OS/2 Administration, SC34-5590*

  This book describes the administration of the CICS Transaction Gateway for OS/2.
- *CICS Transaction Gateway for Windows Administration, SC34-5589*

  This book describes the administration of CICS Transaction Gateway for Windows 98 and CICS Transaction Gateway for Windows NT.
- *CICS Transaction Gateway for AIX Administration, SC34-5591*

  This book describes the administration of the CICS Transaction Gateway for AIX.
- *CICS Transaction Gateway for Solaris Administration, SC34-5592*

  This book describes the administration of the CICS Transaction Gateway for Solaris.
- *CICS Transaction Gateway for OS/390 Administration, SC34-5528*

  This book describes the administration of the CICS Transaction Gateway for OS/390.
- *CICS Transaction Gateway Messages*

  This online book lists and explains the error messages that can be generated by CICS Transaction Gateway.

  You cannot order this book.

- *CICS Transaction Gateway Programming*, SC34-5594

  This book provides an introduction to Java programming with the CICS Transaction Gateway.

  There are also additional HTML pages that contain programming reference information.

CICS Universal Clients books

- *CICS Universal Client for OS/2 Administration, SC34-5450*

  This book describes the administration of the CICS Universal Client for OS/2.

- *CICS Universal Client for Windows Administration, SC34-5449*

  This book describes the administration of the CICS Universal Client for Windows 98 and CICS Universal Client for Windows NT.

- *CICS Universal Client for AIX Administration, SC34-5348*

  This book describes the administration of the CICS Universal Client for AIX.

- *CICS Universal Client for Solaris Administration, SC34-5451*

  This book describes the administration of the CICS Universal Client for Solaris.

- *CICS Universal Clients Messages*

  This online book lists and explains the error and trace messages that can be generated by CICS Universal Clients.

  You cannot order this book.

- *CICS Universal Clients C++ Programming*, SC33-1923

  This book describes how to write object oriented programs for the ECI and EPI in the C++ language.

- *CICS Universal Clients COM Automation Programming*, SC33-1924

  This book describes how to write object oriented programs for the ECI and EPI according to the Component Object Model (COM) standard.

CICS Family publications

- *CICS Family: Client/Server Programming*, SC33-1435

  This book describes the programming interfaces associated with CICS client/server Programming— the External Call Interface (ECI), the External Presentation Interface (EPI), and the External Security Interface (ESI). It is intended for application designers and programmers who wish to develop client applications to communicate with CICS server systems.

Book filenames

Table 1 show the softcopy filenames of the CICS Transaction Gateway and CICS Universal Client books.

Table 1. CICS Transaction Gateway and CICS Universal Clients books and file names

Book title	File name
CICS Universal Clients Messages	CCLHAB
CICS Universal Client for AIX Administration	CCLHAD
CICS Universal Client for OS/2 Administration	CCLHAE
CICS Universal Client for Windows Administration	CCLHAF
CICS Universal Client for Solaris Administration	CCLHAG
CICS Transaction Gateway for OS/390 Administration	CCLHAI
CICS Transaction Gateway Messages	CCLHAJ
CICS Transaction Gateway Programming	CCLHAK
CICS Transaction Gateway for Windows Administration	CCLHAL
CICS Transaction Gateway for OS/2 Administration	CCLHAM
CICS Transaction Gateway for AIX Administration	CCLHAN
CICS Transaction Gateway for Solaris Administration	CCLHAO
CICS Universal Clients C++ Programming	CCLHAP
CICS Universal Clients COM Automation Programming	CCLHAQ
CICS Family: Client/Server Programming	DFHZAD
Note: The File names in this table do not include the 2-digit suffix.	

Sample configuration documents

A number of sample configuration documents are available in the Portable Document Format (PDF) format.

These documents provide step-by-step guidance to help you, for example, in configuring your CICS Universal Clients for communication with CICS servers, using various protocols. They provide detailed instructions that extend the information in the CICS Transaction Gateway and CICS Universal Client libraries.

As more sample configuration documents become available, you can download them from our Web site; go to:

`http://www.ibm.com/software/ts/cics/`

and follow the **Library** link.

Other publications

The following International Technical Support Organization (ITSO) Redbook publication contains many examples of client/server configurations:

- *Revealed! CICS Transaction Gateway with more CICS Clients Unmasked, SG24-5277*

  This book supersedes the following book:

- *CICS Clients Unmasked, GG24-2534*

You can obtain ITSO Redbooks from a number of sources. For the latest information, see:

`http://www.ibm.com/redbooks/`

You can find information on CICS products at:

`http://www.ibm.com/software/ts/cics/`

Viewing the online documentation

You can access all of the documentation provided with CICS Transaction Gateway and CICS Universal Client in our online library. You need Adobe Acrobat Reader and a suitable Web browser to use the online library (and you may need to configure these).

To get to the online library:

- On Windows and OS/2, select the **Documentation** icon.
- On AIX and Solaris, run the **ctgdoc** script.

and the library home page is displayed.

The online library allows you to link to:

- CICS Transaction Gateway and CICS Universal Clients books in PDF format.
- Programming reference documentation in HyperText Markup Language (HTML) files (provided for CICS Transaction Gateway only).
- README files.
- Sample configuration documents in PDF format.
- Translated books in PDF format. (You may find that not all books are translated for your language.)
- The CICS Web site.

Guidance information on using Acrobat Reader is also provided.

Updated versions of the books may be provided from time to time, check our Web site at:

`http://www.ibm.com/software/ts/cics/`

and follow the **Library** link.

Viewing PDF books

The PDF information provides powerful functions for:

- Navigating through the information. There are hypertext links within PDF documents, and to other PDF documents and Web pages.
- Searching for specific information.
- Printing all or part of PDF documents on a PostScript printer.

You can find out more about Acrobat Reader at the Adobe Web site:

`http://www.adobe.com/acrobat/`

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
IBM
OS/390
VisualAge
CICS
OS/2

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, or other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Index

Program Number: 5648-B43

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5594-00