

**Microsoft Visual C++ V5™  
Front End to  
CICS applications  
using the ECI**

Document: CA06  
**Issued:** 9th April, 1998  
**Revision Date:** 11th February, 1999  
**Previous Revision Date:** 9th April, 1998  
**Next Review:** None

Jim MacNair  
IBM Transaction Systems  
MS 1272  
Route 100  
Somers, NY 10589

**Unclassified**

**Take Note!**

Before using this User's Guide and the product it supports, be sure to read the general information under "Notices".

**Second Edition, February 1999**

This edition applies to Version 1.0.1 of Microsoft Visual C++ front end application using the CICS ECI to access CICS applications and to all subsequent releases and modifications until otherwise indicated in new editions.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM United Kingdom Laboratories  
Transaction Systems Marketing Support (MP207)  
Hursley Park  
Hursley  
Hampshire, SO21 2JN, England

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. You may continue to use the information that you supply.

© **Copyright International Business Machines Corporation 1998. All rights reserved.**

Note to U.S. Government Users - Documentation related to restricted right Use, duplication or disclosure is subject to restrictions set forth in GS Schedule Contract with IBM Corporation.

---

# Contents

|   |    |
|---|----|
| <b>Introduction</b>                                     | 1  |
| Objective   | 1  |
| Software environment and prerequisites                  | 1  |
| CICS Clients overview                                   | 1  |
| CICS External Call Interface (ECI) Overview             | 2  |
| CICS ECI Programming Overview                           | 2  |
| Using the ECI   | 2  |
| CICS Client operating environment                       | 2  |
| Asynchronous calls                                      | 3  |
| Multiple calls in the same unit of work                 | 3  |
| CICS Client Support for Microsoft Visual C++            | 3  |
| Preparing to use the CICS Client Support for Visual C++ | 3  |
| Client Classes  | 3  |
| Working with CICS COMMAREAs in Visual C++               | 4  |
| COBOL vs C considerations                               | 4  |
| BMS considerations                                      | 5  |
| Login dialog  | 5  |
| Using the CICS Client Support in a Visual C++ Program   | 5  |
| ECI Error handling in Microsoft Visual C++              | 5  |
| Overview of the Microsoft Visual Studio                 | 6  |
| ECI versus "Screen Scraping" tradeoffs                  | 7  |
| Complex and Error Prone Programming Environment         | 7  |
| Reuse with Multiple client environments                 | 8  |
| Stateless nature of the Web                             | 8  |
| Combine multiple transactions into a single transaction | 9  |
| Conversion options                                      | 9  |
| <br>  |    |
| <b>Installation and Use</b>                             | 10 |
| Pre-requisites  | 10 |
| Installing the SupportPac                               | 10 |
| Execution of the program                                | 10 |
| Preparing the programs for recompilation                | 11 |
| Preparing the Visual Studio                             | 11 |
| <br>  |    |
| <b>Microsoft Visual C++ Programming details</b>         | 12 |
| Application overview                                    | 12 |
| CICS COMMAREA considerations                            | 12 |
| CICS ECI considerations                                 | 12 |
| Main Application Class (CREDGUIC.CPP)                   | 13 |
| Instance variables in the main application              | 13 |
| Common Subroutines in the main application              | 13 |
| CICS Logon dialog Class (LOGIN.CPP)                     | 14 |
| Getting the CICS region names                           | 14 |
| Finding the CICS Client name                            | 15 |
| Main Account Detail dialog Class (CREDGUICDLG.CPP)      | 15 |
| Browse dialog Class (BROWSE.CPP)                        | 15 |
| CICS COMMAREA classes                                   | 15 |
| Inquiry, Change and Add COMMAREA object (INQYCOMM.CPP)  | 15 |
| Browse COMMAREA object (BRWCOMM.CPP)                    | 16 |
| <br>  |    |
| <b>Server Program Overview</b>                          | 18 |

|                                     |    |
|-------------------------------------|----|
| Server Programs Overview            | 18 |
| Password verification (VERIFYPW)    | 18 |
| Inquire, Add and Update (NEWCALL)   | 19 |
| Browse (NEWCBRW)                    | 19 |
| <b>Debugging techniques</b>         | 21 |
| Debugging overview                  | 21 |
| C++ GUI program debugging           | 21 |
| CICS Client listState method        | 21 |
| Debugging the CICS back end program | 21 |
| Capturing an input COMMAREA in CICS | 22 |
| Debugging the CICS program          | 23 |
| CICS Client Trace                   | 23 |
| Starting the Client Trace           | 23 |
| Using the CICS client trace         | 24 |
| <b>Advanced Topics</b>              | 25 |
| State Management                    | 25 |
| Client names                        | 25 |
| Multi-threading considerations      | 26 |
| Extended units of work              | 26 |
| Support for multiple CICS systems   | 26 |
| <b>SupportPac Contents</b>          | 27 |
| <b>Glossary</b>                     | 29 |

## **Notices.**

**The following paragraph does not apply in any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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

CICS

CICS/ESA

IBM

IBM Transaction Server for Windows NT

VisualAge

The following terms are trademarks of Microsoft Corporation:

Microsoft  
Windows  
Microsoft Windows NT  
Microsoft Windows NT Advanced Server  
Microsoft Windows NT Workstation  
Microsoft Windows 95  
Microsoft Windows 98  
Microsoft Visual Basic  
Microsoft Visual C++  
Microsoft Visual J++  
ActiveX  
Component Object Model

The following terms are trademarks of other companies:

PowerBuilder, Sybase Corporation  
Java, Sun Microsystems

## **Acknowledgments**

The author would like to acknowledge the help graciously provided by Neil Kolban from the IBM Dallas Systems Center and Richard Chamberlain of the IBM Hursley Lab.

## **Summary of Changes**

| <b>Date</b>                | <b>Changes</b>                  |
|----------------------------|---------------------------------|
| <b>11th February, 1999</b> | Fixed bug in ECI error handling |
| <b>9th April, 1998</b>     | Initial release                 |

## **Bibliography**

- *CICS Family: Client/Server Programming, IBM Corp. SC33-1435*
- *CICS Clients Unmasked, IBM Corp. SG24-2534*
- *OO Programming in C++ for CICS Clients, IBM Corp. SC33-1923*
- *Inside Visual C++ Fourth Edition, by David J. Kruglinski, Microsoft Press*

---

## Introduction

This SupportPac is written to provide a working sample of using Microsoft Visual C++ and the IBM Clients ECI interface to access to CICS applications. The CICS server applications are provided in the companion CA03 SupportPac, which should be used in conjunction with this offering to provide a complete solution.

This SupportPac provides a GUI interface to an existing 3270 based application, and is a client-side companion to the CICS application sample provided in the CA03 SupportPac. It was written and tested in the Microsoft Windows NT™ environment.

The programs in this SupportPac are written as a complete sample application, to illustrate various techniques involved in accessing a CICS application from a C++ application.

---

## Objective

The objective of this SupportPac is to provide a working sample application which provides a GUI (C++) based front end to the sample CICS applications provided in SupportPac CA03. This includes working client programs written in C++. This document provides additional hints and tips, as well as instructions for installing and using the sample application. The samples are written to allow reuse as a template for customer applications.

---

## Software environment and prerequisites

This SupportPac was developed using Microsoft Windows NT Advanced Server V4.0™, and should work equally well on the Workstation version of Windows NT. In addition, the following additional products were used:

- IBM Transaction Server for Windows NT V4.01™
- Microsoft Visual C++ V5™

Executable versions of the GUI programs, compiled with Microsoft Visual C++ V5™ are provided. However, this SupportPac does not include the MFC runtime libraries. These runtime libraries are included with the compiler, and may be installed on the system by other products as well.

---

## CICS Clients overview

CICS clients support access from non-CICS environments to CICS applications. The clients provide three key functions, two programming interfaces and a terminal emulator function.

From a programming point of view, the clients offer two interfaces, known as the External Call Interface (ECI) and the External Programming Interface (EPI). The ECI uses a Call/Return model, and is a CICS specific form of remote procedure call. The EPI is a 3270 oriented protocol, to support *screen scraping* applications.

IBM offers CICS clients on a variety of popular Client platforms, including most forms of Microsoft™ Windows (Windows 3.11, Windows 95, Windows NT, etc), OS/2, Apple MacIntosh™, PC/DOS, etc. The clients support a number of protocols to connect to CICS server systems, with the most popular being TCP/IP and SNA. The clients can connect to multiple CICS server systems at the same time.

The clients are small and efficient, and do not require a full CICS environment on the client system. They must be connected to a CICS server. CICS servers are available on a variety of platforms, including smaller server systems

running OS/2 or Microsoft Windows NT™, mid range systems running popular versions of Unix and IBM mainframe environments (MVS and VSE).

---

## CICS External Call Interface (ECI) Overview

The CICS External Call Interface (ECI) is one of the two CICS client programming interfaces. It allows a program running outside of the CICS environment to access CICS programs by calling them as subroutines. The ECI includes support for CICS security, units of work, etc. Unlike most Remote Procedure Call (RPC) environments, the ECI does not require stubs, interface definition language (idl) compilers, etc, and is therefore quite easy to use.

---

## CICS ECI Programming Overview

A CICS application which is called from a CICS client using the ECI can be any CICS program which can be called as a subroutine with EXEC CICS LINK, and which does not depend on a terminal. The application receives a user COMMAREA as input and returns the updated COMMAREA to the calling application.

An ECI program must build two data areas and then issue the actual ECI call. The CICS client sends the request to the CICS server. The server receives the request, performs any necessary data translation and starts a transaction to process the request. When the transaction is complete, the CICS server will return the updated COMMAREA to the requesting client. The ECI call will generate a return code, to indicate any errors. The return code should be checked, and if the call was successful, the updated COMMAREA can then be used.

The ECI supports a variety of options, including synchronous and asynchronous (non-blocking) invocations, multiple calls within a single unit of work, user defined transaction codes, and other options. The programs in this SupportPac use simple synchronous calls.

### Using the ECI

The COMMAREA contains all user data to be passed to the CICS application. The called application must place any data it is returning to the ECI application in the COMMAREA, and the updated COMMAREA is returned to the ECI application. COMMAREAs can be up to 32,500 bytes long. Larger data areas must be segmented.

The second data area that the ECI program must build is the ECIBLOCK. This area contains the parameters for the ECI call. For example, it contains the name of the CICS program to be executed, the user id and password to be used, the address and length of the COMMAREA and the type of call.

The ECI area should be initialized to binary zeros. The necessary fields should be filled in, and the ECI call issued. The return code should be checked. For normal return codes, the COMMAREA should have been updated by the CICS application, and no information is returned in the ECIBLOCK. For two exceptional return codes, useful debugging data are returned in the ECIBLOCK. For a sample of a typical ECI call, see the performECI routine in the ECIMSG.C program.

### CICS Client operating environment

The CICS client requires a separate daemon process on the client system. The daemon process has no user interface. This process can be started from a command line, as an NT service or will be started automatically when the first client call is issued. Once started the client process remains running. It can be shut down from a command line program. The client offers powerful trace and debugging options, which can be turned on and off with a command line utility. The client uses a client definition file (CICSCLI.INI) to specify many configuration options.

## Asynchronous calls

ECI calls can be either synchronous (blocking) or asynchronous (non-blocking). For asynchronous calls, there are several ways for the calling application to receive notification that a call has completed, including the use of a semaphore, a windows message or a call back routine. The application receives two return codes. When the call is first made, the CICS client will validate the call parameters, and if no problem is found, will initiate the request and return a normal return code. If a problem is found, then the return code will be set to reflect the error and no attempt is made to send the request to the CICS server. When the request is complete, the application will receive a second return code, indicating if the call was successful or not.

## Multiple calls in the same unit of work

The CICS clients support combining more than one ECI call as a single CICS transaction. If a later call in the same unit of work fails, then the entire transaction will be rolled back, including the effects of any earlier calls in the same unit of work. This facility is known as extended units of work.

---

## CICS Client Support for Microsoft Visual C++

The CICS clients provide support for Microsoft Visual C++ The support is documented in the book *OO Programming in C++ for CICS clients* referred to in the supporting publications section.

## Preparing to use the CICS Client Support for Visual C++

To use the CICS Client support in Microsoft Visual C++, the CICS Client support must be installed. The CICS\_W32 variable should be defined in the project settings. To do this, select the settings option from the project menu. Select the C++ tab and then preprocessor category. Add the CICS\_W32 variable to the end of the preprocessor definitions list.

## Client Classes

To use this support, five objects must be defined. The five objects are:

- ECI
- Connection
- Flow
- Buffer
- UOW

There should only be one ECI object in an application. The ECI object will issue the ECI\_Initialize call in its constructor. The ECI\_Initialize call will start the CICS client process if the client daemon is not already running, and will start a thread in the application process to listen for client replies. The ECI object has methods to determine the number of CICS servers which have been configured in the CICSCLI.INI file, and their names.

The Connection object provides methods to use a CICS server connection. In general, there should be one connection object for each CICS server that the application uses. The client will establish the actual connection to the server the first time that any application on the client system makes a request to that particular server. Once a connection to a server is established, then the connection will persist. The connection can only be stopped by running the command line utility CICSCLI.EXE with the /X=servername parameter. Creating or destroying a connection object is therefore a relatively inexpensive operation. The Connection object has methods which allow the

server name, user id and password to be set. At a minimum, the *Details* method should be called to set a user id and password, and optionally a CICS server name. If the server name is not set, then the connection will use the default server. The default server is the first server defined in the client definition file (CICSCLI.INI).

The Flow object is used for sending an actual ECI request to the server. The flow object allows synchronous or asynchronous calls to be made to the server. In most cases, synchronous calls should be used. Flow objects can be reused for many subsequent calls. A flow object is busy for the duration of a CICS transaction, and cannot be reused until the current transaction has completed. If asynchronous calls or extended units of work are used, then the program must be careful to not try to reuse a flow object which is already in use for a new transaction.

The buffer object (CclBuf) passes the CICS COMMAREA to the client. The CICS COMMAREA is the data area that the CICS application receives data from and returns data in. The COMMAREA can either be contained within the buffer object or can be external to the buffer object.

The UOW object is used for cases where multiple ECI calls will be made as part of a single transaction. It is not required for single calls. In most cases, it is not needed, and a Null pointer should be used on the connect.link method calls.

## Scoping considerations

The ECI object should be allocated first, and should persist for the duration of the application. There must be one and only one ECI object. If a second ECI object is created, the constructor will throw an error. A good place to put the ECI object is in the main application class as a global variable. The ECI can then be accessed if needed and should persist for the length of the application.

The other objects are only needed for the duration of a particular CICS transaction. They are relatively inexpensive to create and destroy. Although the connection object would appear to involve considerable overhead in establishing a connection to a CICS server, the connection is actually held by the client in a separate process, and there is no need to keep an instance of the connection allocated. On the other hand, the connection object(s) can be allocated in long term storage and reused for multiple transactions.

## Working with CICS COMMAREAs in Visual C++

A CICS COMMAREA is an application specific user data structure, usually consisting of many individual fields. The approach taken in this SupportPac is to encapsulate all explicit COMMAREA handling, as well as the CICS ECI calls, in a separate user defined subclass of the COBJ class. The COMMAREA class could also have been defined as a subclass of the CclBuf class. However, in this case, the afxDump support could not be used, since the CclBuf class is not derived from the standard COBJ. Instance variables are used to make the appropriate data available to the rest of the application, including the user data fields (such as name and address) as well as error messages and information. The CICS ECI calls are encapsulated in appropriate logical methods (such as executeInquiry).

## COBOL vs C considerations

Most CICS applications are written in COBOL and run on IBM mainframes. Most web servers run on Unix or PC servers. Most CGI applications will probably be written in a language other than COBOL. There are some significant differences in the formats of some of the primitive data types, especially strings. In the case of COBOL, strings are fixed length memory areas, and are usually padded with spaces (blank characters) on the right to their defined length. In the case of most Unix or PC languages, such as C, a string is variable length and is delimited by a character containing a binary zero. There is no automatic padding with blanks. These differences are handled by subroutines supplied with this SupportPac.

## BMS considerations

When a 3270 transmits data to a host, it only sends changed fields. If a user simply presses the enter key with typing any characters into any fields on the screen, then no user data is sent, and a CICS MAPFAIL error will be raised when a BMS RECEIVE is issued. To handle this properly, BMS input areas contain input data plus additional input fields which indicate the length of any data in an individual field. If applications are checking the length fields, then the length fields must also be filled in. Finally, to properly simulate the MAPFAIL condition, the total length of any input data must be calculated and sent with the screen data.

To support input into BMS-like fields in a COMMAREA, two additional subroutines are provided. These routines fill in the user input data and the length field, and add the length of the field data to the total length field.

## Login dialog

The login dialog is contained in a separate class, and this class should be largely reusable in customer applications. It performs several useful functions, including getting a user id and password from the end user and validating them, assigning a unique name to the client and selecting the CICS region that the user will be using. The login dialog uses its own CICS COMMAREA to communicate with a CICS program called VERIFYPW, which is supplied with the companion CA03 SupportPac.

## Using the CICS Client Support in a Visual C++ Program

The ECI object is created in the InitInstance routine of the main application object, and a pointer to the ECI object is stored as a public variable in the application object. The ECI object is used in two ways. First, it is used by the Login dialog to get the names of all defined CICS servers, to load into a combo box control. Secondly, the ECI object is used to retrieve error information (error message and CICS abend code).

The ECI object that is used (fixCclECI) is a subclass of the CclECI class, and is found in a header file named *clnterr.h*. The reason for using this subclass is to bypass a bug in the standard CICS client C++ support. Although the CICS abend code is supposed to be available in the CclFlow object when a CICS transaction abend occurs, the abend code is actually reset before it can be retrieved from the CclFlow object. To make the abend code available to the application, a new ECI object is derived

## ECI Error handling in Microsoft Visual C++

If the client support encounters an error, it will throw an exception. There are two primary places where CICS clients will throw exceptions.

First, the flow method of a CICS connection object will attempt to execute a CICS program on the CICS server. If any errors are encountered, then an exception will be thrown. Many different errors are possible, such as the CICS server not running or a transaction abend (abnormal termination) in the CICS application program. After an exception, the error code and the associated text can be obtained from the ECI object using the `exCode` and `exCodeText` methods. The individual error codes can be found in the `Ccl` class.

Although the ECI object will issue an `ECI_Initialize` call in its constructor (when the object is created), the constructor does not check return codes and will not throw an error. This call will ensure that the CICS client is running. If the client is not running, then an attempt will be made to start the client. The call will also start a reply thread in the local process and attempt to connect to some shared storage. If the client does not start, or if the local process cannot find or connect to the shared storage (which can include Windows NT security reasons), then an exception will be thrown on later ECI calls, such as the flow method in the connection object or the `systemName` method in the ECI object.

---

If an exception is thrown when the program tries to get the server names, then no communication with CICS is possible, and in most cases the application should exit gracefully.

Other exceptions are possible, but these tend to be programming errors. For example, an attempt to delete a CclUOW object which has an active transaction running will raise an *activeUOW* exception.

When an error is thrown, the exception code and message are placed in the exception object. The exception object is only in scope for the duration of the catch clause. It contains the exception code and the associated text message. This information should be retrieved and stored using the `exCode` and `exCodeText` methods. The `diagnose` method returns a long text string with a fairly complete set of information about the error. This information could be written to an error log, for example.

The error information is also placed in the ECI object. This works in a single threaded environment. However, in a multi-threaded environment, the ECI information will be overwritten with the results of the next ECI call, so by the time one thread looks at the ECI, it may be seeing information about a different request.

## Bugs and Design flaw in Client Error Handling

As discussed above, there is also an error in the handling of the CICS abend code in the C++ support. Although the CclFlow object has a method which purports to return the CICS abend code, this method will never return a CICS abend code. By the time an error can be caught by the application, the abend code in the flow object will have been reset. To get the CICS abend code, the CclECI class must be subclassed and the `handleException` method overridden. See the example in the `inqycomm.h` or `inqybrws.h` files.

## Error code definitions

The ECI error codes themselves have a complete set of defined constants in the Ccl class. The enumeration for the error codes themselves can be found in the `CCLDEFS.HPP` file.

---

## Overview of the Microsoft Visual Studio

Most of the Microsoft compilers use a set of common components called the Visual Studio. They include a GUI screen painter, a source debugger, language editor, object viewer, and many other useful tools. The graphical interfaces used by the Microsoft Visual C++ applications in this SupportPac were developed using the Screen Painter, and debugged using the source debugger.

The Visual Studio includes an application wizard which should be used to generate the initial application. The class wizard should be used to create instance variables in dialogs and to add message handler routines to the modules in the project.

The sample application in this SupportPac was created using the AppWizard. To invoke the AppWizard, select the new option from the file menu. The MFC AppWizard (exe) option should then be selected, and a name given to the project. The sample application is dialog based (as opposed to single document or multiple document). As a result, there is no main window, and the user only uses one dialog at a time. Although the user is restricted to one dialog at a time, multiple instances of the application can be run concurrently.

---

## ECI versus "Screen Scraping" tradeoffs

There are two main techniques used to access existing production systems from a new client front end. In so-called *screen scraping*, the client application is programmed to appear as a 3270 terminal to the back end programs. With the second technique, the CICS programs are changed to allow them to be called directly by the client.

CICS clients offer a 3270 interface, known as the External Presentation Interface (EPI), which can use existing 3270 programs without requiring any change. This is attractive, and there are cases where this is the best approach (or possibly the only approach, such as cases where purchased packages are involved).

The advantage of using the EPI is that no changes are required to the back end (CICS) programs. However, this may not be the best solution in many cases. Programmers who use screen scraping techniques tend to encounter the following problems:

- Complex, inefficient and error prone programming environment
- Little or no reuse between different client environments
- Conflict with the stateless nature of the Web environment
- Cannot combine multiple host interactions into a single transaction

### Complex and Error Prone Programming Environment

The 3270 terminal was designed for human viewing. It was designed in a time when many users (typically 10-30) shared a single slow (1200-2400 BPS) telephone line to access a central computer. The 3270 data stream was designed to minimize the amount of data that was transmitted, and offered no programming capabilities on the terminal (client) end. Host CICS programmers were shielded from the complexities of the 3270 data stream by the CICS Basic Mapping Support (BMS), which was designed for application programming. Only the terminal hardware had to deal with the actual 3270 data stream.

When a PC program tries to simulate the 3270 terminal, it must deal with the complicated 3270 data stream arriving at the PC, and must return a similarly complex data stream to CICS. Interpreting 3270 data streams can be difficult, tedious and error prone. While it saves effort on the host side, the cost on the client side can be substantial and wipe out the apparent savings.

To further complicate matters, unsolicited or unexpected data streams can arrive instead of the usual application generated data stream. For example, CICS operator broadcasts or CICS generated error messages are expected to notify a human being sitting at a terminal of some situation. When arriving at a program, they can cause problems for the receiving program. This can cause screen scraping applications to be complex and unreliable.

Some examples of exceptional situations would include:

- Transaction abends
- Application errors (for example, File not available, Record changed by other user, etc)
- CICS operator broadcasts (for example, System shutting down in 10 minutes)

It is very easy for an EPI application to get confused in these sorts of situations, and to behave rather unpredictably.

## Reuse with Multiple client environments

A key advantage of converting CICS programs to be callable via the ECI is that these transactions can be used by multiple front ends. For example, it is common for customers to want to access back end CICS transactions from many different environments, such as:

1. Customer written GUI front end, often using multiple client environments, such as:
  - HTML and CGI program using C
  - Microsoft Visual Basic™
  - GUI application using C++
  - PowerBuilder™
  - Numerous other languages and/or tools
2. Web applications such as JAVA applets or servlets
3. MQS (IBM MQSeries) applications

If a screen scraping approach is used, the client handling of the CICS 3270 data streams must be redeveloped for each additional front end environment.

## Stateless nature of the Web

CICS 3270 applications assume that there is a persistent connection between a user's terminal and CICS. Many CICS transactions must be run in a particular sequence, and store status within CICS associated with the user's terminal. Users log on once at the beginning of their session, and their security context persists for the duration of their connection. CICS applications usually specify the next transaction to be run.

In the case of the web, CGI programs are transient. The programs are run in a separate process, which is started just for the particular invocation, and is stopped when the CGI program is finished. There is no easy method to maintain a terminal session with CICS, since terminal sessions are associated with a particular process. Thus, there is an inherent conflict using CICS 3270 applications within CGI applications.

In a typical 3270 usage scenario, a terminal is installed in the beginning of a session, and the user then signs on to this terminal, and continues to use the same terminal for the duration of the session. Transaction context is saved in CICS, and is associated with the terminal. However, in the case of the web, clients are stateless and cannot maintain persistent connections. When a CGI program is run, it is run in a transient process, which is shut down after each interaction with the user.

While solutions are technically possible, they usually involve elaborate structures involving daemon processes which persist on a server and maintain the connections, with shared memory and interprocess communications to communicate with the web applications. Timeouts are also necessary, since there is no indication if a client has terminated its interactions with a particular system and has no plans to return. Further discussion of such a design is beyond the scope of this SupportPac. Rather, the reader should assume that such a solution is quite complex, and should not be considered as normal application programming. This sort of programming is best left to vendors.

The ECI is by nature stateless, and fits much better in web environments.

## **Combine multiple transactions into a single transaction**

In some cases, it is desirable to integrate more than one existing CICS application into a single new transaction. When using 3270 based applications, the client has no control over CICS units of work, and cannot combine the different transactions into a single unit of work. The ECI supports making multiple calls as a single unit of work, using extended calls.

---

## **Conversion options**

Most existing CICS programs are written for 3270 terminals, and use BMS maps. These programs have many dependencies on 3270 terminals, some direct and obvious and others that are more subtle. The CICS applications used by the sample programs in this SupportPac were originally 3270 programs. The programs were changed to support ECI calls as well as a 3270 interface. For details about the CICS programs and the changes made, please see SupportPac CA03.

---

## Installation and Use

---

### Pre-requisites

This SupportPac assumes that the CICS clients have been installed and configured to access a CICS server. The CICS programs in the CA03 SupportPac are called by the sample applications and must have been installed on this server.

Executable versions of the programs are provided. The executable programs were built using Microsoft Visual C++ V5.

A Microsoft Visual C++ compiler (V4.2 or V5) is required to recompile or modify any of the source programs. No Microsoft Visual C++ runtime library is supplied with the programs. If the appropriate runtime is not available on the execution system, then the Microsoft Visual C++ V5 compiler™ must be installed. Users with earlier versions of the Microsoft Visual C++ compiler may wish to rebuild the application at their corresponding level.

---

### Installing the SupportPac

Create a new user directory, copy the provided zip file into this directory and unzip the file. For example, the following could be used:

```
MD \VSTUDIO
CD \VSTUDIO
COPY \TEMP\CA06.ZIP
UNZIP CA06
DIR
```

A directory named CREDGUIC will be built under the VSTUDIO directory, and the Visual C++ project will be unzipped into this directory.

The programs were tested and built using the CICS Clients V2.02, with the service level at SP4 plus nine. There is no known dependency on the level of the Clients

---

### Execution of the program

To execute the sample program, start CICS and (optionally) the CICS client. Then execute the CREDGUIC.EXE application (found in the Release subdirectory under the CREDGUIC directory). This will display a screen asking for a CICS user id, password, CICS system and client name. Enter a valid CICS user id and password (be sure to use the proper case), and an optional CICS system and 4 character name client name. If a client name is defined in the CICSCLI.INI file, then the name will be automatically filled in, but the user can override this value. The CICS system name should match the name of a CICS server as defined in the CICS client CICSCLI.INI configuration file. Press the Logon button to send the data to the the VERIFYPW program on CICS. If the user id and password are valid, the file inquiry page will be displayed.

To look up customer account information, enter the account number and press the button labelled "Inquire". The account information will be displayed.

At this point, more records can be displayed, new accounts can be added and data can be changed. The contents of the file can be displayed using the browse option. The button labelled "Clear Form" will clear all the fields on the main inquiry form, but does not invoke a CICS program.

Since the data file used by the IVP programs does not contain any data, some accounts should be added to the account file, so there is some data to display. This can be done using the 3270-based version of the programs that is supplied with the IBM Transaction Server for Windows NT™ Installation Verification Programs. As an alternative, new accounts can be added with the GUI interface. Fill in the rest of the fields and press the Add New Record button. The date must be in nn/nn/nn format, the phone number should be in the format nnn-nnnn and the amount in the format \$nnnn.nn, where n must be all numbers. The account number can be from 1 to 6 digits long, and must be numeric. To add a new account record, press the ADD button.

---

## Preparing the programs for recompilation

The Program Overview section of this document should be read first, to understand the functions of the various program modules.

### Preparing the Visual Studio

The Microsoft Visual C++ project file should be loaded, using the *Open Workspace* option of the file menu.

Next, select the *settings* option of the *project* menu. Select the Link tab. At the end of the Object/library modules field add an entry for C:\CICSCLI\LIB\CCLWIN32.LIB. Do this for both the debug and release entries. Finally, press the F7 key to build the application. The include directory for the clients must also be added to the project settings. Select the C/C++ tab and then the Preprocessor option. Add the c:\cicscli\include directory to the end of the include line. Please note that this paragraph assumes that the clients have been installed in the default directory on the c: drive. If this is not the case, then the drive and high level directory should be adjusted to match the actual location of the client installation.

The project should now be ready for compilation. To compile the program, select the *Rebuild all* option from the build menu.

---

# Microsoft Visual C++ Programming details

---

## Application overview

The CREDGUIC program is highly modular, and has numerous separate source files, in addition to the usual Microsoft Visual C++ project files. The project contains three dialogs, namely:

1. Logon menu
2. Account details
3. Browse details

When the application is started, the Logon menu dialog is shown first. This dialog will ask the user for a CICS user id and password. The user can also enter a client name and select a CICS system. The user id and password are validated by a CICS program called VERIFYPW, which is supplied as part of the companion CA03 SupportPac. If the user id and password are accepted, the main account details dialog is shown next.

The account details dialog allows a user to enter an account number and display the details. Once a record has been displayed, the user can then change the record. New records can be added to the file, and the contents of the file can be browsed.

If the browse option is selected, a separate browse dialog is used to display the results. The browse dialog shows summaries of four records at a time, and allows the user to move forward or backward.

## CICS COMMAREA considerations

The CICS COMMAREAs are encapsulated in separate classes, one for the Inquire/Change/Add dialog, and a second class for browse operations.

Both classes provide methods to perform their respective functions by calling the appropriate CICS program, and use instance variables to make the data available to the dialogs and to allow the dialogs to set the necessary input parameters. The internal structure of the COMMAREA is encapsulated within the COMMAREA classes and is not visible outside of these modules. The COMMAREA classes are defined in the following modules:

|              |   |
|--------------|---|
| INQYCOMM.H   | Definition of CICS COMMAREA class (inquiry)     |
| INQYCOMM.CPP | Implementation of CICS COMMAREA class (inquiry) |
| BRWCOMM.H    | Definition of CICS COMMAREA class (browse)      |
| BRWCOMM.CPP  | Implementation of CICS COMMAREA class (browse)  |

The COMMAREA layouts themselves are defined in the following header files:

|            |  |
|------------|--|
| CREDCOMM.H | COMMAREA definition for Inquire/Change/Add |
| BRWSCOMM.H | COMMAREA definition for Browse             |

## CICS ECI considerations

The CICS ECI calls as well as some common COMMAREA manipulation routines are contained in the main application program (CREDGUIC.CPP).

---

## Main Application Class (CREDGUIC.CPP)

### Instance variables in the main application

The main application class stores the CICS user id, password, system name and client name in instance variables. CICS error messages and abend codes are also made visible by use of instance variables. The instance variables in the main application are:

|                  |                             |
|------------------|-----------------------------|
| cics_user        | CICS user id                |
| cics_pw          | CICS password               |
| cics_system      | CICS region                 |
| cics_client_name | Client name (4 characters)  |
| cics_message     | ECI error message           |
| ics_abcode       | CICS transaction abend code |

### Common Subroutines in the main application

The following common subroutines will be discussed:

Most of the common subroutines in the main application are declared as static. The setStandardHeader and executeCICSTran routines use instance variables in the main application and therefore are not declared as static. There are some subtle differences between the two types of routines.

Static routines are part of the class and do not pertain to any particular instance of the class. Therefore, they cannot access instance variables. They are invoked by prefixing the method name with the class name and two colons. The bindings are resolved at compile time. As a result, if a static variable is passed in to a static method, the compiler will treat a reference as if it were defined as Const, and will attempt to resolve the binding at compile and link time. This may require some extra type casting on reference pointers, to avoid compiler errors.

For the non-static routines, a pointer to the main application object must first be obtained, using the AfxGetApp routine, and this pointer is then used to invoke the method.

- comm\_move
- move\_comm
- moveBMSfield
- int2char
- executeCICSTran
- setStandardHeader
- dumpStandardHeader

The move\_comm routine takes a CString as input, and moves the data from a C string format into a CICS COMMAREA in COBOL format. The CommField and maxLen parameters are the location and size of the field within the COMMAREA where the data is to be inserted. The isNumber field indicates the data type. It can have three possible values, namely ALPHA\_FIELD, NUMERIC\_FIELD or CICS\_FIELD. If the data type is ALPHA\_FIELD, then the data will be left justified and padded with blanks on the right, as necessary. If the data type is NUMERIC field, then the data will be right justified and padded with zeros on the left, as necessary. If the field type is CICS\_FIELD, then the data will be left justified and padded with binary zeros as necessary.

The `comm_move` field extracts data from the CICS COMMAREA. It strips any trailing blanks. The first parameter is a pointer to a character string where the data is to be placed. The third parameter contains the length of the variable. The second parameter is a pointer to the data within the CICS COMMAREA. The output is a standard null-terminated C style string.

The `moveBMSfield` routine is similar to and uses the `move_comm` routine. This routine also fills in the BMS length field with the length of the input data, and adds the length to a map data length field, to keep track of the total length of all BMS input data.

The `executeCICSTran` routine dynamically creates connection, flow and buffer objects, and then uses the `link` method of the connection object to execute the CICS transaction. Exceptions are caught and a message box is displayed in the event of an exception. The message box statement should be commented out or removed in a production application. If a CICS transaction abend occurs, the CICS abend code is retrieved and stored in the `cics_abcode` instance variable. ECI error messages are placed in the `cics_message` instance variable. These fields are reset before each ECI call.

The user id, password, system name and client name variables are set by the Login dialog.

The application's mouse pointer is changed to a wait cursor (usually an hour glass) before the ECI call, and then restored to a normal pointer before exiting this routine. If this behavior is not desired, then these statements should be removed.

The `setStandardHeader` method sets the fields in the standard COMMAREA header used by the sample CICS applications. This routine is useful if the conversion methodology described in SupportPac CA03 is used to create the CICS applications. This routine is contained in the main application class so that it can be shared by multiple COMMAREA objects in a single application.

The `dumpStandardHeader` method is mostly aimed at debugging. It is intended to support the MFC object dump support (`AfxDump`). It displays the contents of the standard header of a CICS COMMAREA.

---

## CICS Logon dialog Class (LOGIN.CPP)

The Login dialog class is designed to be largely reused in a customer application. This routine displays a modal dialog box with four user fields. When the user presses the *Logon to CICS* button, the user id, password and client name fields are sent to the selected CICS region for verification, invoking the `VERIFY PW` program on the CICS server. This routine checks for a CICS abend code of `AEY7`, which generally indicates an invalid user id or password, as well as a return code from the `EXEC CICS PASSWORD` executed on the CICS host. If the verification is successful, the dialog will exit. Otherwise, an error message is displayed and the user should correct the error and try again.

### Getting the CICS region names

A combo box is used to select the CICS region name. The names are defined in the Client initialization file (`CICSCLI.INI`), and are obtained by using the `serverName` method of the ECI object. This function is handled in the `loadSystemNames` routine in the Login dialog (`LOGIN.CPP`).

---

## Finding the CICS Client name

The Login dialog will attempt to find and read the CICS initialization file, and look for a client name defined in this file. If the file is found and a specific client name is specified in the CLIENT= parameter, then the first four characters of this name will be used as a client name. If no name is found, then either the user can enter one or the VERIFYPW program can generate one.

---

## Main Account Detail dialog Class (CREDGUICDLG.CPP)

The main dialog handles inquiry, change and add requests. A dialog is used with separate text boxes for each data item. A clear fields button is also provided, which will clear all the text boxes for the fields in the data records. This dialog uses the INQYCOMM object to build the COMMAREA in the proper format and access CICS. The INQYCOMM uses instance variables to pass data in to and out of itself. The main dialog must set the appropriate instance variables for the type of request it is making (inquiry, add or update). After a request is made, it must gather the results from the appropriate instance variables.

The main dialog will create and invoke a browse object if the user requests a browse. The main dialog will set the init\_acctno variable in the browse object, to pass the starting account number.

The change button is only enabled when a change request is valid. To change a user record, the data must be displayed first using an inquire request.

---

## Browse dialog Class (BROWSE.CPP)

The browse dialog handles browse requests. The user can move forward or backward through the file. Up to four records at a time are displayed.

A browse operation is started in the OnInitDialog message handler. Additional browse requests are handled in the message handlers for the Forward and Back buttons.

The browse dialog uses a separate class (BRWCOMM) to encapsulate the CICS COMMAREA and ECI calls. This class uses instance variables to pass the results of each browse request back to the Browse dialog.

---

## CICS COMMAREA classes

Two CICS COMMAREA classes are derived from a basic CObject class. The purpose of these classes is to encapsulate the format and structure of the CICS COMMAREA, and to insulate the dialogs from CICS ECI calls. Instance variables are used to pass data back and forth between the dialogs and the COMMAREA objects. Method calls are provided to invoke CICS transactions. The next two sections document the instance variables and method calls for each of the CICS COMMAREA classes.

## Inquiry, Change and Add COMMAREA object (INQYCOMM.CPP)

This class has the following public instance variables and methods.

## Instance variables

### Data Fields

|            |                |
|------------|----------------|
| co_acctno  | Account number |
| co_addr    |                |
| co_name    |                |
| co_phone   |                |
| co_date    |                |
| co_amount  |                |
| co_comment |                |

### Informational fields

|           |   |
|-----------|---|
| co_errmsg | Error messages from CICS or application |
|-----------|---|

### Private variables (for user COMMAREA)

|             |                        |
|-------------|------------------------|
| old_account | Current account number |
|-------------|------------------------|

## Method calls

|                  |  |
|------------------|--|
| executeInquiry() | Execute a CICS inquiry transaction         |
| executeChange()  | Execute a CICS change transaction          |
| executeAdd()     | Execute a CICS add new account transaction |

Internal subroutines are used to build the appropriate CICS COMMAREA and to get data out of a returned COMMAREA.

## Browse COMMAREA object (BRWCOMM.CPP)

This class has the following publically defined instance variables and methods.

## Instance variables

### Data Fields

|             |                                 |
|-------------|---------------------------------|
| br_account1 | Account number of first record  |
| br_account2 | Account number of second record |
| br_account3 | Account number of third record  |
| br_account4 | Account number of fourth record |
| br_name1    | Name field of first record      |
| br_name2    | Name field of second record     |
| br_name3    | Name field of third record      |
| br_name4    | Name field of fourth record     |
| br_amount1  | Amount field of first record    |
| br_amount2  | Amount field of second record   |
| br_amount3  | Amount field of third record    |
| br_amount4  | Amount field of fourth record   |

### Informational fields

|            |   |
|------------|---|
| co_errmsg  | Error messages from CICS or application |
| co_errmsg2 | Error messages from application         |
| co_errmsg3 | Error messages from application         |

## Method calls

executeStartBrowse()   Initiate a browse request  
executeBrowseNext()   Browse forward or backward

Internal subroutines are used to build the appropriate CICS COMMAREA (setMenuMapData and setDetailMapData) and to get data out of a returned COMMAREA into the appropriate instance variables (getDataFromComm).

---

## Server Program Overview

The sample programs in this SupportPac use three CICS programs, which are supplied as part of the CA03 SupportPac. The three programs are:

1. VERIFYPW.CCP
2. NEWCALL.CCP
3. NEWCBRW.CCP

The VERIFYPW program was written specifically for use by ECI applications, and does not have a 3270 interface. The other two programs were originally 3270 programs, and were converted to support an ECI call interface as well as continuing to support access from 3270 terminals.

---

## Server Programs Overview

### Password verification (VERIFYPW)

The password verification program performs three key functions, namely:

1. verifies that a CICS user id and password combination are valid.
2. logs all usage to the system console log.
3. Assigns a unique four character client name, if needed.

The input to the program is a user id and password combination, plus an optional four character client name. If a client name is passed in, then no client name is generated. If the client name field is blank or nulls, then a client name is generated and placed in the field. The algorithm used is very simplistic, and in most cases should be changed to one which better matches local conventions on terminal naming. In either case, the client name is written to the system log.

Many CICS applications require a unique four character terminal name. While CICS servers can assign a unique name to clients when they connect, there is no interface on either the host or client to determine the name. Since many different users will probably be using the web server, without having any names assigned to their browsers, there is no good method to determine a unique client name. In this case, a client name will be assigned by the VERIFYPW program, when the user first signs on to CICS. This name will be stored in all the HTML pages as a hidden variable.

### Cautions about using the VERIFYPW program

To generate unique names, the VERIFYPW program uses the first four bytes of the CICS Common Work Area. **IMPORTANT!! This use of the CWA may conflict with other applications. If it does conflict, then the location of the four byte area used by the VERIFYPW program must be changed. If no CWA has been defined to CICS, then a CWA of at least four bytes must be configured, and the CICS region restarted.**

The VERIFYPW program is written for CICS/ESA V4.1 or later. It uses the EXEC CICS VERIFY PASSWORD and EXEC CICS WRITE OPERATOR commands. Versions of CICS/ESA prior to V4.1, and all TXSeries products except CICS for OS/2, do not support the VERIFY PASSWORD command, and the TXSeries products, except for CICS for OS/2, do not support the WRITE OPERATOR command. If this program is used on a release of CICS which does not support these APIs, then the unsupported commands must be removed.

## **VERIFYPW COMMAREA fields**

The VERIFYPW program expects a 100 byte COMMAREA. It uses the first seventy-two bytes, with a 28 byte unused area at the end of the COMMAREA. All of the fields are character fields. The RESP field contains the result of the EXEC CICS VERIFY PASSWORD. It will normally contain either five zeros, for a normal return code, or 00069 or 00070 if the user id and/or password are invalid.

The user id and password fields are eight bytes long, and should be padded with blanks if necessary.

The client name field is four characters in length. If it contains all blanks or null characters, then the VERIFYPW program will attempt to generate a unique client name, and will return it in this field. The days left, last date and last time fields are filled in based on the results of the VERIFY PASSWORD request.

## **Inquire, Add and Update (NEWCALL)**

The NEWCALL program handles inquiry, update and add requests against the main file. It checks the transaction code to determine the requested function. It expects to be called from either a menu screen or, for update and add requests, from a detail screen. The program determines if it is in the first or second pass of an add or update from the length of the user COMMAREA. If the length is zero, then the program assumes that this is the first pass, and it writes a detail screen, to allow the user to change or enter the account data.

If the program is in the first pass, it expects data from the menu screen, consisting of only a transaction code and an account number. On the second pass, it assumes that the input comes from the detail screen. After the first pass of an add transaction, the program saves the status byte and account number in the user COMMAREA, for a total of seven bytes. For an update, the program saves a copy of the entire record in the user COMMAREA, for a total of 80 bytes.

In order to allow immediate update of a record, without having to read the record a second time, all inquiry transactions are submitted as if they were the first pass of an update transaction.

The program checks for a MAPFAIL condition, so the USRMPLN field must contain the total length of the map data.

For an inquiry, or the first pass of an add or update transaction, the account number must be placed in the account number field in the menu map. For the second pass on add or update transactions, the user COMMAREA fields and USRCALLEN field must be filled in, and the input data must be loaded into the detail map area.

## **Browse (NEWCBRW)**

The original browse program is conversational. The sample converted program was first manually converted to run as a pseudo-conversational transaction. This converted program (DFHCBRW.CCP) has been included in SupportPac CA03. To change the program to be pseudo-conversational, four fields were moved out of WORKING-STORAGE and placed in a user COMMAREA. The fields are the CURROP and LASTOP fields, and the RIDF and RIDB fields. The first two fields indicate the direction of the browse request, and the later two contain the first and last record keys currently displayed. In addition, logic was added to the program to issue an end browse and return to CICS, and to recognize when the program is restarted and to issue another start browse when the program is next executed.

When run from a 3270, the browse program is started from the main menu screen. The only input field which is required is the transaction code, with an optional account number field. The USRCALLEN should be set to zero on the first invocation. The initial browse operation is always in a forward direction. The browse operation will start at the current account number in the detail screen. If no account number is entered, then the browse will

start at the beginning of the file. If an account number of '999999' is entered and the browse key is pressed, then the browse operation will start in a backwards direction from the end of the file.

The browse results are displayed using the browse screen. There are fields for account, name and amount from four account records, and a one character input field to allow the user to select forward or backward browsing by entering an "F" or "B". The user can also page forward with the F1 key and backward with the F2 key. There are also fields for messages.

To continue a browse operation, the desired direction must be filled in in the DIRI field in the browse map, namely an "F" for forward or a "B" for backward browse operations. The total map length field must be set, otherwise a mapfail condition will be generated, and the transaction will end. As an alternative, the USRAID field can be set to PF1 for forward operations, and PF2 for backward operations, in which case the DIRI field will be ignored.

---

## Debugging techniques

---

### Debugging overview

A CGI application accessing a CICS program consists of two distinct parts, namely:

1. C++ Client program
2. CICS program

---

### C++ GUI program debugging

The Microsoft Visual C++ environment includes a powerful source level debugger, which can be used to debug the client applications. The MFC libraries also include additional functions to aid the debugging process. This includes the TRACE, ASSERT and afxDump() statements.

The TRACE statement will display information in the debugger message window. Many of the modules contain trace statements indicating when a routine is entered. The trace statements are only active for a debug build.

The ASSERT statement is used to validate that some condition is true, and can be used to check for invalid data, such as null pointers or zero lengths. Sample ASSERT statements are included in the executeCICSTran routine in the main application module (CREDGUIC.CPP) and in the executeBrowseNext routine in the browse COMMAREA object (BRWCOMM.CPP).

The afxdump statement can be used to write the contents of an object to the debugger message window. To accomplish this, the dump method in CObject must be overridden. Two samples of this technique are provided, in the two COMMAREA objects (BRWCOMM.CPP and INQYCOMM.CPP). To actually display the contents of the object, an afxDump statement must be placed at an appropriate point in the program and then must be executed.

### CICS Client listState method

The CICS clients are not derived from the standard windows CObject class, and therefore do not include a standard dump method. Most of the CICS client objects do include a listState method, however, which can be used to dump out the status of a particular CICS client object. Should the need arise, the user should be aware of this capability.

---

### Debugging the CICS back end program

Once the C++ program appears to be sending the proper COMMAREA values to CICS, it is now time to debug the CICS application. While the program can be driven directly by the GUI program, it is often convenient to be able to execute the CICS program directly from a 3270 terminal session. This can be accomplished by using standard CICS facilities.

The CECI transaction is a standard part of all IBM CICS products. It allows a user at a 3270 terminal to execute CICS commands, including calling CICS programs as subroutines by using the LINK statement. A user variable can be defined for the CICS COMMAREA, data entered into the COMMAREA, and the CICS program can then be called, with the COMMAREA being passed as input, just as if the program had been called by the C++ program.

If the contents of the input COMMAREA are fairly simple, then the data can be entered by hand directly by using the CECI transaction variable screen. However, this is not always the case. Therefore, a method is provided to capture an actual input COMMAREA is desirable.

---

## Capturing an input COMMAREA in CICS

A simple CICS COBOL program (CAPTURE.CCP ) is provided as part of this SupportPac. The program should be compiled and added to the target CICS region. The C++ program should then be modified to execute the CAPTURE program rather than the CICS program it is intended to call. The C++ program should then be executed. The CAPTURE program will write the contents of the input COMMAREA into a temporary storage queue named COMMAREA.

The contents of the captured COMMAREA can be examined with the CICS CEBR transaction.

To use the captured input COMMAREA, invoke the CECI transaction from a 3270 terminal session. Three steps are then necessary to execute the target CICS program with the input COMMAREA, namely:

1. Create user variables for the COMMAREA length and data.
2. Issue a temporary storage read to get the COMMAREA data into the user variable.
3. Issue a CICS link to execute the target CICS program, passing the user variable with the COMMAREA as input

This can be accomplished as follows. First, start the CECI transaction, by typing CECI and pressing enter (usually the lower right ctrl key on a PC keyboard). Next, press F5 to go to the variable screen. Tab down to the line below the last variable and type &LEN for the variable name, and 4 for the variable length. Press enter to create the &LEN variable. Again, tab to the line below the last variable, and type &CA in the variable name, tab to the length field and type in the length of the user COMMAREA. Press enter to create the &CA variable. Press F5 to return to the main execution screen.

Please note that it usually does not matter if the COMMAREA variable is longer than needed, but program exceptions can result if the area is smaller than required.

Read the COMMAREA data into the user variable by using typing the following command into the top line.

```
READQ TS Q(COMMAREA) INTO(&CA) LEN(&LEN)
```

Press the enter key to check the command syntax. The message "about to execute command" should be displayed. Press enter again to execute the read. Check the return code, making sure that it was normal (0).

To execute the actual CICS program, return to the top line and type the following:

```
LINK PRO(pgmname) COMM(&CA) LENG(&LEN)
```

Press enter to check the command syntax. The message "about to execute command" should be displayed. Press enter again to execute the program.

---

## Debugging the CICS program

If a CICS debugger is to be used, it should usually be started or configured before the CECI transaction is started.

There are several options for debugging the CICS program. The three that will be covered are:

1. CEDF
2. Source level debuggers
3. Purchased CICS debugging packages

All IBM versions of CICS include the standard CICS Execution Diagnostic Facility (CEDF) debugger. Unfortunately, CEDF is quite limited in its debugging capabilities. It is most useful for examining CICS calls made by the application.

In some CICS environments, such as Windows NT, the compilers provide source level debugging capabilities for CICS programs. These allow CICS programs to be single stepped, variables to be examined and values changed, breakpoints set, as well as many other useful capabilities. These debuggers can be extremely valuable in finding bugs in CICS programs.

The last category is CICS debugging packages that can be purchased from a variety of non-IBM vendors. Many of these packages offer high quality tools for debugging CICS programs, and can be used directly in this environment.

Details on the use of CICS debugging tools, however, is beyond the scope of this SupportPac.

---

## CICS Client Trace

The CICS clients offer a very useful and powerful trace facility. Prior to V2.03 of the clients, this trace showed all communications traffic between the client and server. It would not show errors from invalid API calls, however, since no communications took place. As of V2.03, an API trace option is also available, which would show API failures as well as communications traffic (to be accurate, the API trace became available in a service pack offering to V2.02).

The CICS clients support two types of traces, in memory and direct to disk. The traces are written as a plain ASCII text file and can be viewed with any editor, including the Windows notepad facility. By default, the client trace file is written to the \CICSCLI\BIN directory with a name of CICSCLI.TRC.

## Starting the Client Trace

The client trace is started by using the CICSCLI command. To execute this command, start a DOS command window. If the client is not already running, it should be started. To check if the client is already running, execute the CICSCLI command with the /L parameter. This will display a message indicating if the client is running, and if it is running, will show any server connections which have been started. For example, the following command could be used:

```
CICSCLI /L
```

If the client is not running, it can be started with the following command:

```
CICSCLI /S
```

To start a connection to a particular server, change the /S parameter to include the CICS server name as defined in the CICSCLI.INI file. For example,

```
CICSCLI /S=CICSTCP
```

To start a client trace and write the results directly to disk, the following command could be used:

```
CICSCLI /D=512
```

The number after the /D statement is the maximum number of bytes in a user COMMAREA to write for a single trace entry. The purpose of this parameter is to limit the size of the trace files, which can grow large rather rapidly. To view a trace file using the notepad facility, the following command could be used:

```
NOTEPAD \CICSCLI\BIN\CICSCLI.TRC
```

The trace file can be erased or renamed at any time. Subsequent data will be written to a new CICSCLI.TRC file. To save a trace file, it should be renamed. Large areas of uninteresting trace entries can be removed with a text editor. When a trace file is no longer needed, the trace facility should be stopped and the trace file deleted. For example,

```
CICSCLI /0  
ERASE \CICSCLI\BIN\CICSCLI.TRC
```

The first command stops the trace and the second command erases the trace file.

To use the memory trace, the memory trace option must first be started. For example, to start the in memory trace, use the following command:

```
CICSCLI /T=512
```

The trace is kept in an in memory buffer. When the buffer is filled, the trace will wrap around to the front and start writing over the older entries. To view the in memory trace, two steps are required. First, the trace must be written to disk and second the file must be viewed or printed. For example, to dump the trace buffer to disk and then view it using the notepad facility, the following commands could be used:

```
CICSCLI /Z  
NOTEPAD \CICSCLI\BIN\CICSCLI.DMP
```

Please notice that the file extension is different than the normal trace file extension.

## Using the CICS client trace

The client trace shows both the ECI parameters and return codes, plus the inbound and outbound user COMMAREA. This usually makes it very easy to spot errors in parameters and user data. For example, the CICS program name, system name, user id and password are all clearly visible in the trace. It is generally easy to find user data fields as well, and to check if they have the proper values.

---

## Advanced Topics

---

### State Management

---

With 3270 CICS applications, the CICS application is in charge of the application flow. Typical CICS applications consist of a sequence of transactions which use certain common techniques to control the flow from one transaction to the next. In many cases, an individual transaction must leave information for the next transaction in the sequence. This information left for the next transaction is often referred to as context.

There are several CICS facilities commonly used to store transaction context, such as COMMAREAs, temporary storage, Terminal Control Table User Areas (TCTUAs), etc. Transaction flow is controlled by using the TRANSID parameter on the EXEC CICS RETURN command.

In a CICS 3270 environment, the user display (terminal) is connected to and controlled by CICS. CICS controls what is displayed on the screen and what applications the user can execute. While the user can clear the screen or log off from CICS, either the user application or CICS itself is notified of the user action. With a GUI application, the client can move to completely different application, with no indication to the CICS application or region the user last interacted with.

The sample programs use CICS COMMAREAs to save transaction state. This works well for relatively small amounts of data and numbers of variables, as is true for the sample programs. If the transaction state that must be maintained is very large, then

the context should be stored within CICS. This requires some additional application programming. A logical place to store this context is in temporary storage, passing back only the key of the temporary storage. This key would then be saved in the CICS COMMAREA, and passed back to the application. The CICS application could then use the key to find the transaction context.

---

### Client names

Each 3270 terminal which accesses CICS is given a unique four character terminal name. These names are often used within applications to create unique names for temporary storage queues, to log user access to resources and for other similar purposes. In addition, COMMAREAs which are passed from one transaction to the next automatically are associated with a particular terminal.

The sample program provided with this SupportPac will look for the CICS client definition file (CICSCLI.INI) when it first starts up, and if it finds the file and there is a client name specified, as opposed to an asterisk (which implies host generation of a name through an auto install type of process), the client name field will be filled in automatically using this name. The program also allows a user to enter a four character client name, and if one is not entered, will generate one in the VERIFYPW program (shipped as part of the companion CA03 SupportPac).

At least two other techniques to generate client names are worth mentioning.

In some cases, client systems are given unique names by some other application or technique. If the client has such a name in another application, and that name can be accessed by the C++ program, then this name should be used for a client name.

The client TCP/IP address or host name also can make a good candidate for a client name. For example, if a host name is present, then the first or last (or some other combination) four characters can be used. In other cases, the TCP/IP address of the client can be used.

---

## Multi-threading considerations

The sample programs are single threaded and use simple synchronous calls. The user is advised to read the section on error handling problems in the client support, since the technique used to retrieve the CICS abend code does not support multi-threaded applications. This SupportPac is aimed at applications where a user navigates between numerous application dialog windows, but only deals with one application at a time.

Some multi-threading can be accomplished by running more than one instance of the application at the same time. However, the routine which uses the CICSCLI.INI file to find a client name must be changed in this case. If this routine is merely removed, then each client instance will have a unique name assigned by the VERIFYPW program.

---

## Extended units of work

The subroutines provided with this SupportPac do not allow for extended units of work. To support extended units of work, the buffer, connection and flow objects must be moved out of the executeCICSTran routine, so that they will remain in scope for multiple successive calls. A method of indicating an extended call, as well as support for a COMMIT call, would have to be added.

---

## Support for multiple CICS systems

The sample programs in this SupportPac assume that all transaction requests are sent to a single CICS system. The user can select the option to rerun the logon dialog and select a different system. However, no provision is made to support the use of more than one CICS system for a single Microsoft Visual C++ application. To add this support, a CICS system name could be passed in as a parameter to the executeCICSTran routine, rather than storing this value as a global variable in the application object. However, unless the same user id and password is valid in all possible regions, some method of executing the logon dialog multiple times must be provided, as well as some facility for storing the multiple user ids and passwords, and recognizing when a particular user id and password have already been validated for a particular system.

This could be accomplished by using a table for the CICS user ids, passwords and system names, and prompting the user with the logon dialog on the first reference to each individual system. As an alternative, the first user id and password could be tried on each new system first, before prompting the user. This would allow for some user ids and passwords being valid on more than one system.

---

## SupportPac Contents

This SupportPac is delivered as a single zip file (CA06.ZIP). The zip file contains the following files:

LICENSE2.TXT License Agreement for SupportPac  
CA06.PS Documentation (Postscript)  
CREDGUIC.ZIP Sample program file

Both source and executable versions of all the programs are provided. The GUI programs are written in Microsoft Visual C++ and compiled with the Microsoft Visual C++ V5™ compiler. They are 32-bit windows programs, and should run under either Windows NT™ or Windows 95™. The GUI programs use CICS Client ECI calls, and therefore require include and library files from the CICS clients for Windows NT™.

The CREDGUIC.ZIP file will create a directory called CREDGUIC, and two subdirectories, res and release. The CREDGUIC directory will contain the following files:

|                 |   |
|-----------------|---|
| BROWSE.CPP      | Browse dialog source program              |
| BROWSE.H        | Browse dialog definitions                 |
| BRWCOMM.CPP     | CICS COMMAREA object source               |
| BRWCOMM.H       | CICS COMMAREA object definitions          |
| BRWSCOMM.H      | CICS COMMAREA layout (browse)             |
| CLNTERR.H       | ECI object subclass definition            |
| CREDCOMM.H      | CICS COMMAREA layout (inquiry/update/add) |
| CREDGUIC.APS    | Project resources file                    |
| CREDGUIC.CLW    | Project Class Wizard                      |
| CREDGUIC.CPP    | Main application Source                   |
| CREDGUIC.DSP    | Project file                              |
| CREDGUIC.DSW    | Workspace file                            |
| CREDGUIC.H      | Main application definitions              |
| CREDGUIC.OPT    | Project option settings                   |
| CREDGUIC.RC     | Project resources file                    |
| CREDGUICDLG.CPP | Main dialog source file                   |
| CREDGUICDLG.H   | Definitions for main dialog               |
| DFHCGA          | Menu map area layout                      |
| DFHCGB          | Account detail map area layout            |
| DFHCGC          | Browse map area layout                    |
| INQYCOMM.CPP    | CICS COMMAREA object source               |
| INQYCOMM.H      | CICS COMMAREA definitions                 |
| LOGIN.CPP       | Login dialog source program               |
| LOGIN.H         | Definitions for login dialog              |
| README.TXT      | Readme from AppWizard                     |
| RESOURCE.H      | Resource header file                      |
| STDAFX.CPP      | Precompiled headers source                |
| STDAFX.H        | Precompiled headers definitions           |
| STDHEAD         | COMMAREA standard header layout           |
| VERIFYPW.H      | Password verification COMMAREA layout     |

The res subdirectory will contain the following:

CREDGUIC.ICO Project icon file  
CREDGUIC.RC2 Non-editable resources for project

The release subdirectory will contain the following:

CREDGUIC.EXE      Program executable

## **Glossary**

|                 |   |
|-----------------|---|
| <b>Abend</b>    | - Abnormal End (Program failed)                               |
| <b>AOR</b>      | - Application Owning Region                                   |
| <b>API</b>      | - Application Program Interface                               |
| <b>ATL</b>      | - ActiveX Template Library                                    |
| <b>COM</b>      | - Component Object Model                                      |
| <b>COMMAREA</b> | - Communications Area - user data area used in CICS ECI calls |
| <b>DCOM</b>     | - Distributed Component Object Model                          |
| <b>ECI</b>      | - External Call Interface                                     |
| <b>EPI</b>      | - External Presentation Interface                             |
| <b>GUI</b>      | - Graphical User Interface                                    |
| <b>GUID</b>     | - Global Unique Interface Definition                          |
| <b>LAN</b>      | - Local Area Network  |
| <b>MFC</b>      | - Microsoft Foundation Classes                                |
| <b>OCX</b>      | - OLE Custom Control  |
| <b>OLE</b>      | - Object Linking and Embedding                                |
| <b>RPC</b>      | - Remote Procedure Call                                       |
| <b>TOR</b>      | - Terminal Owning Region                                      |



---

## **Sending your comments to IBM**

**Microsoft Visual C++ V5™**  
**Front End to**  
**CICS applications**

### **CA06**

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form.
- By fax:
  - From outside the U.K., after your international access code use 44 1962 841409
  - From within the U.K., use 01962 841409
- Electronically, use the appropriate network ID:
  - IBMLink: IBMGB(TSCC)
  - Internet: [tsc@hursley.ibm.com](mailto:tsc@hursley.ibm.com)

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



# Readers' Comments

**Microsoft Visual C++ V5™**  
**Front End to**  
**CICS applications**

## CA06

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

---

Name

---

Address

---

Company or Organization

---

Telephone

---

Email

**C++ GUI to CICS using ECI  
CA06**

**You can send your comments POST FREE on this form from any one of these countries:**

|           |           |            |                     |                      |               |
|-----------|-----------|------------|---------------------|----------------------|---------------|
| Australia | Finland   | Iceland    | Netherlands         | Singapore            | United States |
| Belgium   | France    | Israel     | New Zealand         | Spain                | of America    |
| Bermuda   | Germany   | Italy      | Norway              | Sweden               |               |
| Cyprus    | Greece    | Luxembourg | Portugal            | Switzerland          |               |
| Denmark   | Hong Kong | Monaco     | Republic of Ireland | United Arab Emirates |               |

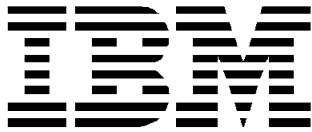
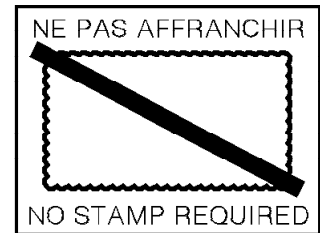
**1** Cut along this line

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

**2** Fold along this line

**By air mail  
Par avion**

IBRS/CCRI NUMBER: PHQ - D/1348/SO



REPONSE PAYEE  
GRANDE-BRETAGNE

IBM United Kingdom Laboratories Limited  
Information Development Department (MP 095)  
Hursley Park  
WINCHESTER, Hants  
SO21 2ZZ  
United Kingdom

**3** Fold along this line

From: Name \_\_\_\_\_  
Company or Organization \_\_\_\_\_  
Address \_\_\_\_\_  
\_\_\_\_\_   
EMAIL \_\_\_\_\_  
Telephone \_\_\_\_\_

**1** Cut along this line

**4** Fasten here with adhesive tape 