**13. CICS**

**Table of Contents**

1. [Introduction 3](#_TOC_250024)
2. [Customer Information Control System 4](#_TOC_250023)
3. [CICS – Transaction flow 5](#_TOC_250022)
4. [CICS Services 6](#_TOC_250021)
   1. [Data Communication Services 6](#_TOC_250020)
   2. [Data Management Services 6](#_TOC_250019)
5. [Programming in CICS 7](#_TOC_250018)
   1. [CICS commands 7](#_TOC_250017)
6. [CICS Program – Execution steps 8](#_TOC_250016)
   1. [Sample CICS-COBOL program 8](#_TOC_250015)
   2. [Sample CICS-COBOL translator/compile JCL 10](#_TOC_250014)
7. [Introduction to BMS 12](#_TOC_250013)
   1. [Characteristics of 3270 display 12](#_TOC_250012)
   2. [The attribute byte 12](#_TOC_250011)
   3. [Attribute byte – format 13](#_TOC_250010)
   4. [Using BMS macros 13](#_TOC_250009)
   5. [Coding a mapset 13](#_TOC_250008)
   6. [DFHMSD 15](#_TOC_250007)
   7. [DFHMDI 15](#_TOC_250006)
   8. [DFHMDF 15](#_TOC_250005)
   9. [CICS Command 16](#_TOC_250004)
8. [CICS and File operation 18](#_TOC_250003)
   1. [READ operation 18](#_TOC_250002)
   2. [WRITE operation 19](#_TOC_250001)
9. [Appendix –A 21](#_TOC_250000)

# Introduction

Transaction management system is a sub-set of IBM Mainframe operating systems for performing and maintaining online transactions. In general, transaction systems must be able to support high number of concurrent user access to the system and allow users to invoke different types of transactions. Online transaction processing (OLTP) is a technique in mainframe that allows users to perform multiple transactions interactively. Some of the characteristics of an OLTP system include:

* 1. Performing user authorization checks for online system access
  2. Maintaining parallel tasks and dispatching the tasks efficiently
  3. Memory management across user’s task
  4. Synchronizing the simultaneous user access to the data files

Some of the real time examples, where the mainframe online transaction systems are used are Bank ATM transactions, Supermarket purchasing with credit and debit cards, Internet online shopping, etc. There are two transaction management systems available in IBM mainframe as follows:

1. Customer Information Control Systems (CICS)
2. IMS Transaction Manager

CICS is one of the widely used transaction processing system that provides various services for running an application online in mainframe. This paper discusses in detail about the CICS applications, transaction flows, CICS commands and programming in CICS.

# Customer Information Control System

CICS is an online transaction processing system in mainframe, which can handle hundreds of users simultaneously and can run several application programs behind the scene supporting the requests. CICS system acts as an interface between the application program and the operating system to communicate with the terminal and other computer hardware. For instance, an application program that needs access to a computer terminal should just need to call the appropriate CICS interface function. The CICS interface function will in turn communicate with the terminal or the device.

In general, the operating system has multiple CICS systems up and running to serve user request. These CICS systems runs in its own address space, commonly called as CICS regions. There is a feature called multi-region operation (MRO) available in CICS which enables it to maintain several regions working on specific functions. For example, there could be set of CICS regions to do the terminal control which are known as terminal owning region (TOR), a set of regions to control application programs known as application owning region (AOR) and file owning regions (FOR) for file controls.

The table below shows different hardware and software in which CICS can run:

|  |  |
| --- | --- |
| Hardware | Operating System |
| IBM zSeries 900 | z/OS |
| IBM S/390 | OS/390, MVS, VSE |
| IBM AS/400 | OS/400 |
| IBM RS/6000 | AIX(UNIX) |
| PC Server | Windows NT, OS/2 |

The most popular of the hardware and software in which CICS is said to be run is IBM S/90 with the OS/390 operating system.

The below table lists the programming languages in which CICS commands can co-exist to make a CICS online application program.

|  |  |
| --- | --- |
| Languages | Description |
| COBOL | The widely used programming language for batch and online  applications in a mainframe environment. Today, an overwhelming majority of CICS programs are in COBOL. |
| Assembler Language | Assembler language was fairly popular in the 70s and early 80s before COBOL became the dominant business language. Today, assembler language is still being used for special−purpose devices  like ATM machines. |
| PL/I | In the 70s and 80s, PL/I was an alternative to COBOL that was used by a small percentage of CICS shops. However, nowadays there are no new programs developed in PL/I. |
| C and C++ | Traditionally used in engineering and math environments, C and C++ can also be used to access CICS services. With C++, an object−oriented language, we can develop object−oriented classes  that access CICS services. |
| Java | The newest language for writing CICS applications, Java can also be used for developing object−oriented classes that access CICS  services. |

CICS uses techniques such as multitasking and multithreading to manage multiple user operations simultaneously.

# CICS – Transaction flow

In CICS environment, application programs can be executed by specifying respective transactions in the terminal. A CICS transaction is a four character unique name, also known as “Trans-ID”, that relates to an application program which needs to be executed. When a Trans-ID is specified, CICS loads the application program to the storage (if it is not already loaded) and initiates a task for the user. For the successful execution of the transaction, the transaction ID and the associated program should be registered in CICS.

CICS maintains several control tables for keeping track of the transactions, related programs and the resources that will be used by the program. These control tables are listed below:

* 1. Program Control Table (PCT)
  2. Processing Program Table (PPT)
  3. File Control Table (FCT)
  4. Terminal Control Table (TCT)

PCT maintains the relationship between Trans-IDs and programs. When a Trans-ID is entered in the terminal, CICS locates the Trans-ID in PCT and reads the corresponding program name to be executed. Upon determining the program, CICS locates the program name in PPT to know if the program has already been loaded. If already loaded, CICS proceeds for execution. Else, the load module is loaded into main memory for execution and an update is made to PPT.

The Trans-ID and the program name can be registered to PCT and PPT using the CICS registered transaction called CEDA. CEDA transaction invokes Resource Definition Online (RDO) process of CICS in the backend for the registration. FCT and TCT are discussed in detail in the below sections.

# CICS Services

CICS provides various services for application programs to access terminal, data files, inherit operating system’s features, etc. These CICS services are also known as application program interfaces (APIs). Below are three categories of CICS services that are available for application program’s usage.

* 1. Data Communication Services
  2. Data Management Services
  3. CICS Management Services

## Data Communication Services

CICS Data Communication Services allows application programs to communicate with the terminal device. Basically, it offers a set of APIs to send and receive data from the terminal. CICS uses Terminal Control interface to interact with the operating system’s telecommunication access method. An application program that needs access to terminal devices can call terminal control interfaces directly. But, a cumbersome parameters preparation needs to be done before calling the terminal control interface and a special string processing needs to be added after the call. To make CICS application programming easier, a technique called, Basic Mapping Support (BMS) was developed for application programs to easily design the terminal outlook and to send and receive terminal field values.

BMS lets user to create a map that specifies the format in which the data to be displayed in the terminal. Conversely to receive the terminal inputs, application programs issues BMS request. After BMS receives the map, it creates a terminal control request which will in turn process the map and sends the information to the terminal.

CICS uses TCT to maintain the terminal details. Each terminal is given a unique four character terminal identifier, or term-id. The TCT definition helps CICS to identify the intended terminals to which the program output should be directed to.

## Data Management Services

CICS provides Data management services for application programs to access VSAM files, DB2 and IMS databases. The service consists of file control, SQL and DL/I interface to access VSAM, DB2 and IMS respectively.

The file control service allows application program to perform file operations. Also, file controls helps CICS system to synchronize the file access across multiple users. CICS uses FCT for maintaining the list of available files per application program. In addition, FCT also contains file access information such as, whether the file will be read sequentially, randomly, indexed, etc.

As in general, SQL interface calls passes the application program request to DB2 and DL/I interface calls passes the request to IMS database.

Note: CICS Management Services will not be discussed in this paper.

# Programming in CICS

There are two different types of CICS programming.

* 1. Conversational programming
  2. Pseudo-conversational programming

In conversational programming, the application program interacts with the user until user enters the end equivalent command in the terminal. In otherwords, the task related to the application program will be actively running waiting for user’s response/input. Normally, this style of programming is not recommended as it locks a huge amount of resources loaded in the storage. When a CICS program is run, there are many components such as the load module and all the related control tables that need to be loaded into the main storage.

In pseudo-conversational programming, a group of individual application programs work together to present an interactive-like application. The tasks related to every application program end when the application program completes sending or receiving the data from the terminal. Every action that user performs at the terminal will trigger a new task to process the request. This style of programming is very efficient as the resources associated with the program will be released as and when the task gets completed.

## 5.1 CICS commands

Application programs can invoke CICS services using EXEC CICS command. The syntax of this command is:

EXEC CICS

command option(value) END-EXEC.

Below is the list of commonly used CICS commands.

|  |  |  |
| --- | --- | --- |
| Sl.No | Commands | Description |
| 1 | RECEIVE MAP | Receive input data from the terminal |
| 2 | SEND MAP | Send output data to the terminal |
| 3 | ABEND | In the event of abend, transfer control to the abend routine |
| 4 | LINK | Invoke another program |
| 5 | RETURN | Return control to CICS or the invoking program |
| 6 | XCTL | Transfer control to another program |
| 7 | DELETE | Delete a record from the file |
| 8 | READ | Read a record from the file |
| 9 | REWRITE | Update a record in the file |
| 10 | SYNCPOINT | Changes made to one or more files are committed or rolled back |
| 11 | WRITE | Write a record to a file |

# CICS Program – Execution steps

There are a series of steps that needs to be followed before executing a CICS program. For description purpose, a simple CICS-COBOL program has been chosen.

* 1. Design and develop a CICS-COBOL application program.
  2. Supply the program through a CICS translator. This will replace all the EXEC CICS commands to the equivalent native program calls. For instance, in case of CICS-COBOL program, the translator will comment the EXEC CICS….END-EXEC statement and replaces these statements with the equivalent COBOL calls.
  3. Compile the output version of translator. In case of CICS-COBOL program, subject the translator output to a IGYCRCTL step.
  4. Linkage edit the COBOL program to generate a load module.
  5. Once the load module has been prepared, copy the load module to the CICS load library. In Marist, CICSS.COMMON.CICSLOAD is the CICS load library.
  6. Login to the CICS region using your TSO user ID password.
  7. Register the Trans-ID in PCT with the CEDA transaction. A sample CEDA syntax to define transaction is:

CEDA DEF TRANS(<Trans-ID>) PROGRAM(<Program-Name>) GROUP(<Group-

Name>)

* 1. Register the program name in PPT with the CEDA transaction. A sample CEDA syntax to define the program is:

CEDA DEF PROGRAM(<Program-Name>) GROUP(<Group-Name>)

* 1. Install the transaction in CICS using CEDA transaction. A sample CEDA syntax to install the transaction is:

CEDA INSTALL TRANSACTION(<Trans-ID>) GROUP(<Group-Name>)

* 1. To invoke the program, enter the Trans-ID in the CICS terminal.

## Sample CICS-COBOL program

A sample CICS-COBOL program that takes two numbers from the CICS terminal and prints out the sum to the terminal is as shown below:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |  |  |
| --- | --- | --- |
| \* PROGRAM NAME: | CICSADD | \* |
| \* AUTHOR: | R. Subashree | \* |
| \* |  | \* |
| \* FUNCTION: | This program is a sample CICS program to add | \* |
| \* | two numbers. | \* |
| \* |  | \* |
| \* | When user inputs two numbers in the input | \* |
| \* | terminal, this program adds the two numbers | \* |
| \* | and displays the sum. | \* |
| \* |  | \* |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IDENTIFICATION DIVISION.

PROGRAM-ID. CICSADD.

AUTHOR. SUBASHREE. DATE-WRITTEN. 11/30/2011. DATE-COMPILED.

DATA DIVISION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* VARIABLE DICTIONARY \*

\* \*

* TER-INPUT Input area declaration containing number \*
* fields. \*

\* \*

* TER-OUTPUT Output area declaration to print the sum \*

\* \*

* TER-MSG-LEN Indicates the read / write data length \*
* to the terminal \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WORKING-STORAGE SECTION.

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | TER-INPUT. 05 TER-NUM1 | PIC | 9(3). |
|  | 05 TER-NUM2 | PIC | 9(3). |
| 01 | TER-OUTPUT. 05 TER-NUM3 | PIC | 9(4). |
| 01 | TER-MSG-LEN | PIC | S9(2) COMP. |

PROCEDURE DIVISION. BEGIN.

MOVE 6 TO TER-MSG-LEN.

* Get input from the terminal EXEC CICS RECEIVE

INTO(TER-INPUT)

LENGTH(TER-MSG-LEN) END-EXEC.

* Calculate the sum

COMPUTE TER-NUM3 = TER-NUM1 +

TER-NUM2.

MOVE 4 TO TER-MSG-LEN.

* Print the total in the terminal EXEC CICS SEND

FROM(TER-OUTPUT)

LENGTH(TER-MSG-LEN)

ERASE END-EXEC.

GOBACK. BEGIN-EXIT. EXIT.

## Sample CICS-COBOL translator/compile JCL

//KC03IF0A JOB ,'RAMACHANDRAN',MSGCLASS=H,NOTIFY=KC03IF0

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* The purpose of this job is to compile and linkage-edit a

//\* CICS-COBOL program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* TRANSL- Translates EXEC CICS commands to equivalent COBOL

//\* calls.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//TRANSL EXEC PGM=DFHECP1$,PARM='COBOL2'

//STEPLIB DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR

//SYSPRINT DD SYSOUT=\*

//SYSIN DD DSN=KC03IF0.CSCI565.CICSSRC(CICSADD),DISP=SHR

//SYSPUNCH DD DSN=&&SYSCIN,

// DISP=(,PASS),UNIT=SYSDA,

// DCB=BLKSIZE=400,

// SPACE=(400,(400,100))

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* COMPL - Compiles CICS-COBOL program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//COMPL EXEC PGM=IGYCRCTL,PARM='FLAG(I,I),APOST',REGION=2048K

//STEPLIB DD DSN=IGY340.SIGYCOMP,DISP=SHR

//SYSLIN DD DSN=&&OBJMOD,UNIT=PUB,SPACE=(TRK,(3,3)), X

// DISP=(NEW,PASS,DELETE)

//SYSPRINT DD SYSOUT=\*

//SYSUT1 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT2 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT3 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT4 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT5 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT6 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSUT7 DD UNIT=PUB,SPACE=(CYL,(1,1))

//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* LKED - Linkage edits the CICS-COBOL program and generates

//\* the load module.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LKED EXEC PGM=IEWL,REGION=1024K

//SYSLIB DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR

// DD DSN=CEE.SCEELKED,DISP=SHR

//SYSLMOD DD DSN=KC03IF0.CSCI565.LOADLIB(CICSADD),DISP=MOD

//SYSUT1 DD UNIT=PUB,SPACE=(TRK,(10,10))

//SYSPRINT DD SYSOUT=\*

//SYSLIN DD DSN=&&OBJMOD,DISP=(OLD,DELETE,DELETE)

//

# Introduction to BMS

The online interactive screens used in the CICS application can be designed using Basic Mapping Support (BMS). BMS provides a set of macros using which the interactive screens can be designed by organizing the input and output fields, making the screens more user friendly. These macros also provide several options to control the display of contents in the screen making it more appealing and user friendly. In order to understand the BMS features and the macro options, it is essential to know some of the basic properties of 3270 display and the terminologies associated with the 3270 display. This chapter gives a brief about the characteristics of 3270 display, later discusses about the BMS macro usage and integrating BMS mapset with CICS application.

## Characteristics of 3270 display

The 3270 display is a field-oriented display in which the screen is divided into a number of logical user defined fields. The beginning and ending of a field is identified by an “attribute byte”. The attribute byte is a one byte field that marks the beginning or end of the field and also controls the display of the field. The attribute byte is displayed as a space in the 3270 display screen. A field begins from the next byte following the attribute byte and continues until the next attribute byte is encountered. If there is no ending attribute byte in a line, the end of the line is marked as the end of the field.

## The attribute byte

In addition to marking the field’s begin and end, attribute byte also defines the property of field. There are three properties (aka attributes) associated with a field that can be set using the attribute byte. They are:

* + 1. Protection
    2. Intensity
    3. Shift

### Protection attribute

This attribute defines if a field is *protected* or *unprotected*. As the name suggests, protected indicates that data entry is not allowed in the field and unprotected indicates that data can be entered in the field. When the field is set as protected, there is another option that can be specified called *auto-skip* that governs the behavior of cursor as it is advanced to this field. There are two options associated with auto-skip – *skip* and *stop.* When the *skip* option is set, the cursor tabbing automatically skips this field and stops in the next unprotected field. When the stop option is set, the cursor tabbing will place the cursor in this protected field even though user cannot enter any data in this field.

### Intensity attribute

This attribute defines the style of field display. There are three options with in this attribute – *normal*, *bright* and *no-display*. *Normal* indicates that the data will be displayed with normal intensity. *Bright* indicates that the data will be displayed brighter than the normal intensity. *No-display* makes the data in the field invisible.

### Shift attribute

This attribute indicates whether *alpha-numeric* or *numeric* characters are expected in the field.

## Attribute byte – format

The attribute options listed above correspond to bit settings in the attribute byte. Below table shows the bit position and corresponding values for the respective options.

|  |  |  |
| --- | --- | --- |
| **Bit positions** | **Attribute** | **Bit values** |
| 0 – 1 | Unused | Unused |
| 2 – 3 | Protection and shift | 00 – Unprotected alphanumeric 01 – Unprotected numeric   1. – Protected stop 2. – Protected skip |
| 4 – 5 | Intensity | 1. – Normal 2. – Normal 3. – Bright 4. – No-display |
| 6 | Unused | 0 always |
| 7 | MDT | 1. – Field has not been modified 2. – Field has been modified |

MDT (Modified Data Tag), is used to indicate whether or not user has modified the data in the field. The 3270 terminal turns on this bit if the user has modified the following field. The terminal will transmit the data from this field only if this bit is on. This helps in saving data transmission time.

## Using BMS macros

The online screen can be designed using the BMS macros. A set of BMS macro instructions what is known as map, formats the screen. A set of map is called a BMS mapset. A mapset can contain more than one map although it is always restricted to one for better maintenance and efficiency.

BMS macros are assembler macros that need to be coded by providing display attribute options in the macro’s keyword parameter. The macros would then have to be assembled to produce two components namely *physical map* and *symbolic map*. The *physical map* is a load module that contains all the screen attributes as defined by the BMS macro options which will be displayed in the 3270 panel. The *symbolic map* represents the screen in the form of a data structure of the specified language. For example, if the language used in the application is assembler, then symbolic map will be a DSECT structure. If COBOL is the language, then the symbolic map is a COPYBOOK that will be used to reference in the COBOL program.

## Coding a mapset

A mapset can be constructed with the help of three BMS macros and with varying options. The macros are:

* + 1. DFHMSD
    2. DFHMDI
    3. DFHMDF

The below table shows a sample BMS mapset and the corresponding screen layout.

PRINT NOGEN

DFHMSD TYPE=&SYSPARM,

LANG=COBOL, MODE=INOUT, TERM=3270-2, CTRL=FREEKB,

INQSET1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DFHMSD TYPE=FINAL

END

DFHMDF POS=(5,26),

LENGTH=6, ATTRB=(NORM,UNPROT,IC), COLOR=TURQUOISE, INITIAL=’ ’

DFHMDF POS=(5,33),

LENGTH=1, ATTRB=ASKIP

CUSTNO

STORAGE=AUTO, DSATTS=(COLOR,HIGHLIGHT), MAPATTS=(COLOR,HIGHLIGHT), TIOAPFX=YES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INQMAP1 DFHMDI SIZE=(24,80),

LINE=1, COLUMN=1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DFHMDF POS=(1,1),

LENGTH=8, ATTRB=(NORM,PROT), COLOR=BLUE, INITIAL=’INQMAP1’

DFHMDF POS=(1,20),

LENGTH=16, ATTRB=(NORM,PROT), COLOR=BLUE,

INITIAL=’Customer Inquiry’

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DFHMDF POS=(3,1),

LENGTH=42, ATTRB=(NORM,PROT), COLOR=GREEN,

INITIAL=’Type a customer number’

DFHMDF POS=(5,1),

LENGTH=24, ATTRB=(NORM,PROT), COLOR=GREEN,

INITIAL=’Customer name’

BMS mapset definition

Type a customer number

Customer number

Customer Inquiry

INQMAP1

Sample screen layout

## DFHMSD

This macro defines a group of related maps known as mapset. This macro has two formats in which it can be defined. One format identifies the beginning of a map and the other (TYPE=FINAL) marks the end of the map. Below shows the DFHMSD prototype:

Format1

TYPE= [SYSPARM|DSECT|MAP], LANG= [COBOL|ASM|PLI], MODE= [IN|OUT|INOUT],

TERM= terminal-type, CTRL= (option,option. . .) STORAGE=AUTO, MAPATTS=(COLOR,HILIGHT), DSATTS=(COLOR,HILIGHT), EXTATT= [YES|NO|MAPONLY], TIOAPFX= [YES|NO]

DFHMSD

name

Format 2

name DFHMSD TYPE=FINAL

## DFHMDI

This macro is used to define a map within the mapset. This map name should be a unique name within the mapset. The prototype of this macro is as below:

## DFHMDF

This macro is used to define the attribute of a field. For a protected field, one DFHMDF macro needs to be coded. For an unprotected field, two DFHMDF macros need to be coded in which the first one indicates the beginning of the field and the next DFHMDF macro indicates the end of the field. Below shows the prototype of DFHMDF macro.

POS=(line,column), LENGTH=field-length,

ATTRB=(BRT|NORM|DRK,PROT|ASKIP|UNPROT,NUM, FSET),

INITIAL=’literal’ COLOR=color, HILIGHT=highlight, PICIN=’picture-string’, PICOUT=’picture-string’

DFHMDF

name

## CICS Command

There are two CICS commands that are used for displaying the screen to the online panel and receiving the screen along with the user input from online.

SEND MAP

The “SEND MAP” command sends the mapped output data to a terminal. Control options such as ALARM, FREEKB, FRSET or PRINT can be specified in this command in addition to specifying in the DFHMSD and DFHMDI macro. The control options specified in this command will override the control options specified in the BMS macros. Below shows the syntax and an example of SEND MAP command:

MAPSET – Specified the mapset name from where the map should be used

FROM - Specifies the data area that contains the data that is to processed and sent to the terminal

LENGTH – Specified the length of the data that is to formatted DATAONLY – Specifies that only application data is to written

- Specifies the name of the map that is to be used

MAP

MAPSET (mapset name) FROM (data-area) LENGTH (data-value) DATAONLY

Description:

(map name)

SEND MAP

Example:

EXEC CICS

SEND MAP(‘INQMAP1’) MAPSET(‘INQSET1’) FROM(INQMAP10) DATAONLY

END-EXEC

RECEIVE MAP

The “RECEIVE MAP” maps input data from a terminal into a data area in an application program. Following a RECEIVE MAP command, the inbound cursor position is placed in EIBCPOSN, and the terminal attention identifier (AID) is placed in EIBAID.

MAPSET – Specifies the mapset from where the map should be

- Specifies the name of the map that is to be used

MAP

Description:

MAP(name) MAPSET(name) INTO(data-area)

RECEIVE

displayed

INTO - Specifies the data area into which the mapped data is to be written

Example

EXEC CICS

RECEIVE MAP(‘INQMAP1’)

MAPSET(‘INQSET1’) INTO(INQMAP1I)

END-EXEC.

# CICS and File operation

CICS provides various commands for working with files. Both random and sequential files can be read, write, updated or deleted using CICS commands. While programming in COBOL, the file handling statements in COBOL such as SELECT, READ, WRITE do not work in the CICS environment. Like any other resources such as transaction, program or load module, files used in CICS application program should also be registered in the control table known as File Control Table. CICS refers to the FCT table for getting the physical file name associated with the file name that is used in the program. The below section discusses about the various file handling commands that are used in the CICS application.

## READ operation

The READ command can be used to read a VSAM or sequential file. Below is the syntax of the READ command.

EXEC CICS

READ FILE (filename) INTO (data-name) RIDFLD (data-name) RRN|RBA

UPDATE

RESP(response)

END-EXEC

Field description:

FILE - This specifies the name of the file that needs to be read. The name should have an entry in the FCT.

INTO - This specifies the data area where the record that is read has to be placed.

RIDFLD - For a key-sequenced file, this specifies the data area which contains the record key that needs to be fetched. In the case of RRN or RBA options are specified, this field represents the relative record number or the relative byte address.

RRN - This indicates that the file is a relative record file that needs to be addressed with record number.

RBA - This indicates that the record in the file should be addressed with relative byte addressing.

UPDATE – This is an optional parameter that indicates that the program may update the record that is being read using the current READ statement.

RESP - This field indicates the data area where the response code of the READ operation will be placed.

Sample READ command

EXEC CICS

READ FILE(‘CUSTMAS’)

INTO(CUSTOMER-RECORD’) RIDFLD(CM-CUST-NUMBER) UPDATE

RESP(RESPONSE-CODE)

END-EXEC.

* The above read command reads the file represented by CUSTMAS in the FCT table
* The record is placed into the working storage variable CUSTOMER-RECORD
* CM-CUST-NUMBER is a working storage variable that has the key value corresponding to the record that is to be read.
* UPDATE in this READ command indicates that the record that is read will be subjected to update later in the program.
* RESPONSE-CODE field will contain the return code of the READ command.

## WRITE operation

The WRITE command can be used to add a record to the random or sequential file. Below is the syntax of the WRITE command.

EXEC CICS

WRITE FILE(filename)

FROM(data-name) RIDFLD(data-name) RRN|RBA

END-EXEC

Field description:

FILE - This specifies the name of the file that needs to be written to. As mentioned in the READ command, the name should have an entry in the FCT.

FROM - This specifies the data area which specifies the data that needs to be written to the file.

RIDFLD - For a key-sequenced file, this specifies the data area which contains the record key that needs to be fetched. In the case where RRN or RBA options are specified, this field represents the relative record number or the relative byte address.

RRN - This indicates that the file is a relative record file that needs to be addressed with record number.

RBA - This indicates that the record in the file should be addressed with relative byte addressing. Example:

EXEC CICS

WRITE FILE(‘CUSTMAS’)

FROM(CUSTOMER-MASTER-RECORD) RIDFLD(CM-CUSTOMER-NUMBER) RESP(RESPONSE-CODE)

END-EXEC.

* The above command writes a record to the file mentioned in FCT corresponding to the logical name CUSTMAS.
* The data stored in the working storage variable CUSTOMER-MASTER-RECORD will be written to the file CUSTMAS
* RIDFLD represents the key value for the record that is being written
* RESPONSE-CODE field contains the return code of the WRITE command operation

In addition to the READ and WRITE command, CICS also provides other file handling commands as below:

REWRITE – Used to update a record that was read using READ command with UPDATE option DELETE – Used to delete a record from a file

UNLOCK – This command is used to unlock a record that was held by a previous READ command

# Appendix –A

Screen shots of CEDA registration, display and install panels in Marist, are given below:

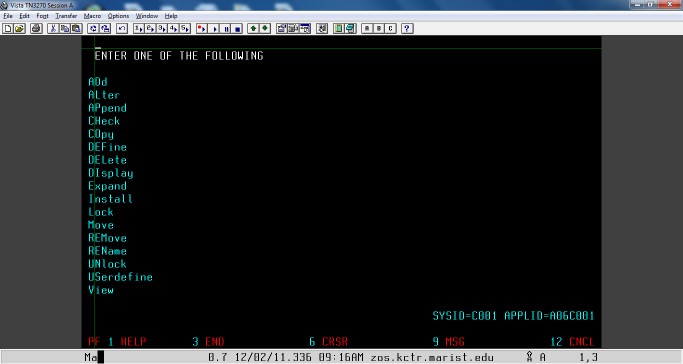


Fig 1: CEDA transaction – Primary Menu

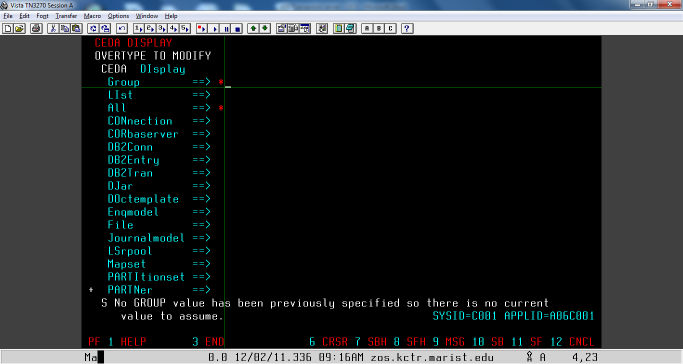


Fig 2: CEDA DISPLAY – Selection terminal

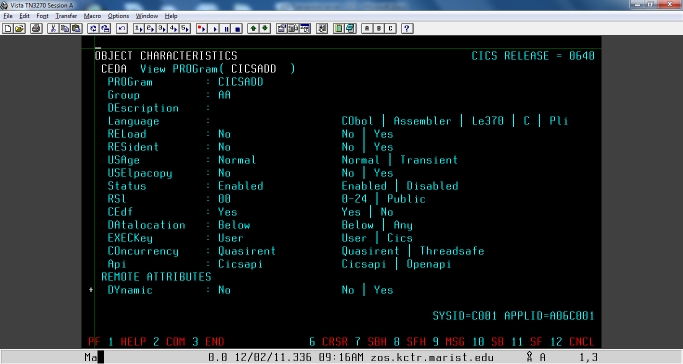


Fig 3: CEDA PPT display for CICSADD program

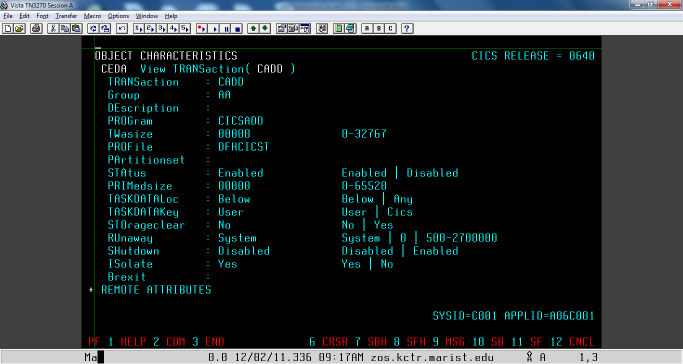


Fig 4: CEDA PCT display for Trans-ID CADD for CICSADD program