Customer Information Control System

CICS

CICS (**C**ustomer **I**nformation **C**ontrol **S**ystem)

History

IBM launched the initial version of CICS in 1968.
It is a Database/Data Communication Control System where an application program can concentrate on the application processing without worrying about OS, hardware and others. Initially CICS was on macro level and later upgraded to command level.

This book is written based on CICS/MVS V2R3

Task and Transaction

Task is a unit of work and transaction is an entity that initiates the execution of task. The transaction identifier identifies the transaction in CICS.

Conversation and Pseudo conversation

Program can communicate with user by a pair of SEND and RECEIVE commands. But in between the SEND and RECEIVE, that is during the human response time, all the resources are held by the program.

Once the user entered the information in the screen, the program proceeds further. This mode of communication is called Conversational mode. It is un-famous for the resource wastage.

In Pseudo conversation mode, whenever there is a need for conversation with user, the program logically terminates there, releases the resources held by it and pass control information to next transaction and the next transaction is automatically started once user entered the information in the screen.

It is actually multi task operation but looks like conversation from user point of view.

For easy understanding we would say the communication in telephone line is conversational mode (Telephone line is in usage through out the communication.) IRC as Pseudo conversation (Once the message sent, the line is freed and it again gets the resources when the other side replied.)

Multitasking and Multithreading

Operating system allows execution of more than one task concurrently and this is called multitasking. If the one or more concurrent tasks use the same copy of the program, then this is called multithreading. So it is obvious that multitasking is a subset of multithreading.

Reentrant and Quasi-reentrant

If the same copy of the program is used by multiple tasks, then the system should take care of the proper reentrance after the SVC/CICS interruption.

Reentrant program is defined as a program that does not modify itself so that it can reenter to itself and continue processing after an interruption by the SVC call of OS. Batch programs are non-reentrant. Reentrancy under CICS environment is called quasi entrant as the interruption in CICS may involve more than one SVC calls or no SVC at all.

COBOL programs should be compiled with RENT option for reentrancy in multithreading environment.

CICS Control Program and Control Tables

CICS Nucleus consists of IBM-Supplied CICS control programs and corresponding user-specified CICS control tables. The important tables with respect to developer point of view are given below:

Control Table	Function
PCT	The transactions and the main program associates with the transaction should be registered in Program table. Task control Program (KCP) refers PCT.
PPT	All the CICS programs and Maps have to be registered in Processing Program Table. Program Control Program (PCP) refers PPT.
FCT	All the VSAM files used in the CICS programs has to be registered in File Control Table. File Control Program (FCP) refers FCT.
DCT	Transient data queues should be predefined in Destination Control Table. Transient Data Program refers DCT.
TST	If you want to recover Temporary storage queues during system crash, then they should be registered in Temporary Storage Table.
RCT	If any DB2 commands are used in the program, then the PLAN should be registered here.
SNT	User ID and Password should be registered in Sign-On-Table.
TCT	All the terminals should be registered in Terminal Control Table.
PLT	All the programs that need to be automatically started during CICS start up and Shut down should be listed in Program List Table.
JCT	Control information of system logs and journal files is stored in Journal Control Table. Journal Control Program refers to JCT.

CICS-DB2-COBOL Program –Compilation

COBOL programs cannot recognize CICS commands. So all the CICS commands are coded within EXEC CICS and END-EXEC scope. The program is first passed to CICS Translator and the translator will convert all the CICS commands to COBOL call statements and this modified source is passed to COBOL compiler.

If your Program has DB2 commands then the sequence of preparing load module would be DB2 Pre-compiler, CICS Translator, COBOL Compiler and Link editor. Logically speaking the order of DB2 precompiler and CICS translator can be reversed also. But as a convention we are first doing precompilation.

If translation is done first, CICS translator tries to recognize to DB2 statements and would issue diagnostic messages. The other reason is, most of the programs do lot of DB2 operations than CICS operations. So if you do pre-compilation first, all the DB2 statements are converted into COBOL call statements in first phase itself and the translation time would be less.

COBOL-CICS programs should be compiled with RENT RESIDENT NODYNAM and LIB options.

General Syntax of CICS statement

```
EXEC CICS function
        [(option (argument value)]
        [(option (argument value)]
END-EXEC.
```

Restricted COBOL commands in CICS environment

OS SVC Triggering statements	ACCEPT CURRENT-DATE DATE DAY DISPLAY
I/O statements	OPEN CLOSE READ WRITE REWRITE DELETE START
SORT statement	RETURN RELEASE

VS COBOL 2 allows STOP RUN and that returns control back to CICS.

MAP DESIGN

Before get into program design, let us see how maps (screens) are designed in CICS. Most of the installations use tools like SDF for screen designing. The tools generate BMS macros for the designed screen. We brief the BMS macros involved in the map design. BMS is acronym for Basic Mapping Support.

MAP and MAPSET

A screen designed thru BMS is called MAP. One or a group of maps makes up a MAPSET.

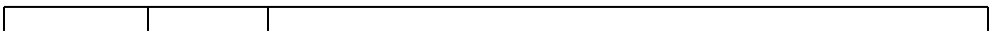
PHYSICAL MAP AND SYMBOLIC MAP

Physical maps control the screen alignment, sending and receiving of CONSTANTS and data to and from a terminal. They are coded using BMS macros, assembled and link edited into CICS LOAD LIBRARY. They ensure the device independence in application programs.

Symbolic maps define the map fields used to store the VARIABLE data referenced in COBOL program. They are also coded used BMS Macros. But after assembling, they are placed in a COPY library and then copied into CICS programs using COPY statement. They ensure device and format independence to the application programs.

BMS Macro Coding Sheet

Since BMS map definitions are purely assembler macros, the following coding convention must me maintained.



1 10 16 72
(1-8) Label (10-15) Macro name (16-71) Operands 72-Continuation.

Macro name. There are three macros used in BMS coding.
1. DFHMSD. Defining Mapset
2. DHMDI. Defining Maps
3. DFHMDF. Defining Fields.

All the BMS coding starts with DFHMSD followed by one or more DFHMDI. One or more DFHMDF follow DFHMDI.

Label: Label is name of the operation. If the macro is DFHMSD, it specifies map-set name. If the macro is DFHMDF then it specifies field name.

Operands: Define the parameters for the macro being invoked.

Continuation: Continuation of macro is achieved by 'X' in 72nd column of the current line and continues from the 16th column of next line.

Comments: Comment lines are indicated by '*' in column 1. A comment line cannot be placed between continuation lines and comment lines cannot be continued.

Blank lines: You cannot have blank lines in the assembler program.

DFHMSD

It is used to define a mapset with its characteristic and to end a mapset. So you will find two DFHMSD in any BMS coding. The important operands are below:

TYPE.

It should be DSECT for Symbolic map generation, MAP for physical map generation and FINAL to indicate the end of mapset. Alternatively symbolic parameter &SYSPARM can be coded in the TYPE of defining DFHMSD and the value can be overridden in the PARM parameter of assembly procedure. This avoids the change in the BMS coding.

MODE.

IN for input maps like order entry screens and OUT for output maps like display screens and INOUT for input-output maps like update screens.

LANG.

It specifies the language in which the symbolic map is to be generated. It can be COBOL, PLI, ASM or RPG.

STORAGE.

AUTO is used to acquire separate symbolic map area for each mapset. BASE=MAP-IOAREA allows multiple maps from more than one mapset to share same storage area. MAP-IOAREA will be redefined multiple times to achieve it.

TIOAPFX.

It should be 'YES' to reserve the prefix space of 12 bytes for BMS commands to access TIOA properly. This is required for command level CICS.

CTRL.

Device control requests are placed here. Multiple parameters are separated by comma. FREEKB is used to unlock the keyboard. FRSET is used to reset the MDT of all the fields in all the maps to zero. ALARM is used to set an alarm at screen display time. PRINT is used to send the mapset to printer.

TERM.

If anything other than 3270 terminal is used for display of screens, then it should be coded here. This ensures device independence by means of providing the suffix. SUFFIX is used to specify suffix for the terminal and it should correspond to TCT entry of the terminal.

DFHMDI

It is used to define a map with its characteristic in a mapset. There can be any number of DFHMDI. Some of the important operands of DFHMDI are below:

SIZE. It has two arguments namely length and breadth and as a whole the size of the map is specified here.

LINE. The map starting line is mentioned here.

COLUMN. The map-starting column within the LINE is mentioned here.
JUST. RIGHT or LEFT is coded here to inform the justification of the map within mapset.

The above four parameters decides the size and location of map within map set. CTRL and TIOAPFX can be also coded in DFHMDI. Value of DFHMDI overrides the value of DFHMSD.

DFHMDF

It is used to define a field with its characteristic in a map. There can be any number of DFHMDF within DFHMDI. Some of the important operands of DFHMDF are below:

POS.

It has two arguments that decided the position of the field. The two arguments are line and column. It is the position where the attribute byte of the field starts.

LENGTH.

The length of the field is coded here. It excludes the attribute character.

ATTRIB.

All the input and output fields are prefixed by one byte attribute field that defines the attributes of the field. Some of the attributes are:

1. ASKIP/PROT/UNPROT Mutually exclusive parameters that define the type of the field. UNPROT is coded for input and input-output fields. PROT is coded for output and stopper fields. ASKIP is coded for screen literals and skipper fields. The cursor automatically skipped to next field and so you cannot enter data into skipper field.
2. NUM. 0-9,Period and – are the only allowed characters.
3. BRT/NORM/DRK Mutually exclusive parameters that define the intensity of the field.
4. IC Insert Cursor. Cursor will be positioned on display of map. If IC is specified in more than one field of a map, the cursor will be placed in the last field.
5. FSET Independent of whether the field is modified or not, it will be passed to the program. MDT is set for the field.

JUSTIFY.

RIGHT is the default value. Code LEFT for numeric fields.

PICIN and PICOUT.

It defines the Picture clause of the symbolic map in COBOL and useful for numeric field editing.

INITIAL.

The default value of the field is coded here. When the MAP is sent, this value will appear in the field. The constant information like TITLE is coded using INITIAL keyword of field definition. To avoid data traffic, these constant information fields should not be coded without LABEL parameter. If there is no LABEL parameter, then symbolic map will not generated for those fields as they are unnamed fields.

How a field looks like in a Symbolic map

Let the label of one DFHMDF is EMPNAME in the map EMPDET and the length of it is 20. The respective symbolic map would look like follows.

```
01 EMPDETI.
    02 FILLER PIC X(12)                TIOAPFX = YES creates this 12 byte filler.
    02 EMPNAMEL PIC S9(04) COMP.       Length Field
    02 EMPNAMEF PIC X.                 Flag byte
    02 FILLER REDEFINES EMPNAMEF.
       03 EMPNAMEA PIC X.              Attribute byte
    02 EMPNAMEI  PIC X(20).            Actual field (Input)
Other fields….
01 EMPDETOO REDEFINES EMPDETI.
   02 FILLER PIC X(12)                TIOAPFX = YES creates this 12 byte filler.
   02 FILLER PIC X(03)
   02 EMPNAMEO  PIC X(20).            Actual field (Output)
Other fields…
```

So four fields are generated for every named field in the BMS.

Fieldname+L

It has length of the field entered by the user during the input operation.

Fieldname+I

It is the actual input field that carries the entered information. The value of this field is X'00' if no data is entered. The space corresponds to X'40'.

Fieldname+A

It is attribute byte. It defines the attributes of the field.

Fieldname+F

It is a flag byte. It has X'00' by default. It will be set to X'80' if the user modifies the field but no data is sent. That is when the user pressed clear key over the field.

Fieldname+O

This field should be populated in the program before sending the screen to display.

Please note that the words INPUT (RECEIVE) and OUTPUT (SEND) are with respect to program.

More on Attribute field

We have just seen that attribute field is of 1 byte that specifies the attributes of the field. One byte is of eight bits and the values in each bit has its own meaning. For example,

Bit 2 - '0' indicates the field is unprotected and '1' indicates the field is protected.
Bit 3 – '0' indicates the field is alphanumeric and '1' indicates the field is numeric.
Bit 2 and 3 – '11' indicates that the field is Auto-skip.
Bit 4 and 5 – '00' indicates Normal intensity and non-detectable.
'01' indicates Normal intensity and detectable.
'10' indicates High intensity and detectable.

'11' indicates Dark and Non-detectable.
Bit 7 – '0' indicates MDT is OFF '1' indicates MDT is ON.

Modified Data Tag(MDT)

MDT is one bit field of the attribute byte. The program can receive only the fields with MDT '1' on RECEIVE. Effective use of MDT can reduce the data traffic drastically in the communication line.

MDT can be SET/RESET in the following ways.

1. When the user modifies the field, the MDT of the field is automatically set to ON.
2. CTRL=FRSET of DFHMSD or DFHMDI will RESET the MDT to 'OFF' for all the fields in the mapset or map. FSET keyword of the attribute operand definition of DFHMDF will set the MDT to 'ON '. It overrides the FRSET definition for the specific field.
3. Before sending the screen, by overriding the MDT bit of attribute byte of field the MDT can be set to 'ON'.

If you are specific on the values of some fields independent of whether the user has modified or not, code them with FSET. One good example is default values for the input fields (like Order received date). If the user finds default value (current date) in the screen and it is fine with his requirement, then he won't touch the field and MDT will not be set. The program cannot receive the field as MDT is OFF.
But actually the program needs this field. So this field should have been defined with FSET.

CURSOR Positioning

Positioning of cursor is an important area in screen design. By default, the cursor will be placed in the first unprotected field.

Static Positioning.

If IC is coded in the attribute operand of DFHMDF macro, then the cursor will be placed in that field. If IC is coded in more than one field, then last field with IC will get the cursor.

Symbolic Positioning.

Move –1 to length of the field where you want place the cursor and send the map with CURSOR option. This is device independent method and recommended to control the cursor position dynamically.

Relative Positioning.

Send the map with CURSOR(value) option. This will set the cursor at the position coded by 'value', relative to the first column of the screen. This is device dependant method of dynamic cursor positioning and not recommended. When there is change in screen layout, the program needs to be modified.

CURSOR(30) will place the cursor in the 30th column. (First line first column is ZERO).

Cursor Position.

EIBPOSN of DFHEIBLK contains the offset of cursor position in the screen when the data was transferred to program from the terminal.(relative to zero). It is half word binary field.

Dynamic attribute setting

Any attribute of the field can be modified in the program by setting the bits of attribute field properly. Changing the attribute in the program in a dynamic method. CICS provides the standard attribute character list in the form of copybook. (DFHBMSCA). This can be copied into our application program using COPY statement and we can use the variables in the copybook to change the attribute of a field in the program.

For example, if the information entered by the user in a particular field is failed in validation, then as a usual practice, in addition to throw an error message in the bottom of the screen, you may wish to hi-light the field that is in error.
MOVE DFHUNIMD to FIELDA. == > Will achieve this.

SKIPPER and STOPPER field

Skipper is unlabelled one byte field with the auto-skip attribute and stopper is unlabelled one byte field with protected attribute.

Skipper field skips the cursor to next field when the current field is filled up to its full length, whereas stopper field stops entering any more character after the length of the field is reached. The user has to press RESET key to proceed further.

Skipper field is used to speed up the user entry and recommended to code at the end of each unprotected field. Stopper field is coded based on requirement and importance of the field. Stopper field usually follows the last unprotected field in the screen.

BMS INPUT-OUTPUT OPERATIONS

Basic BMS input-output operations are carried out using the following commands. RECEIVE MAP, SEND MAP, SEND CONTROL, SEND TEXT, SEND PAGE.

RECEIVE MAP.

It is used to receive the information entered by the user into program. If INTO is not coded, then CICS automatically finds the symbolic map area (mapname+I) and places the mapped data. The values of the field can be read in the program by referring to fieldname+I.

EXEC CICS RECEIVE MAP(map-name) MAPSET(mapset-name) END-EXEC.

The option TERMINAL ASIS overrides the upper-case translation specified in the TCT. As we already discussed, if the user didn't modify any of the fields then MDT will be 'OFF' for all the fields. Zero bytes transmission results MAPFAIL exception error.

SEND MAP.

This is used to send a map to a terminal. Before issuing this command, the application program should prepare the data in the symbolic map area. If FROM is not coded, then CICS automatically finds the symbolic map area (mapname+O) and takes the data to the terminal.

EXEC CICS SEND MAP(map-name) MAPSET(mapset-name) END-EXEC.
Other Options in SEND MAP are:

ERASE. Erases the screen before displaying the MAP. When you first time throw the screen, it should be specified with ERASE. It will not be coded when you just want to display the error messages over the current screen. ERASEUP erases only the unprotected fields of the screen before displaying the MAP.

DATAONLY. Only symbolic map data is sent to the terminal.

MAPONLY. Only physical map data is sent to the terminal. FROM cannot be coded.

FREEKB, ALRAM and FRSET can be also coded and the meaning is already discussed.

TEXT Building- SEND-TEXT, ACCUM and PAGING

Text streams can be sent to the terminal without any pre-defined BMS maps. This is called text building. Text streams can optionally have header and trailer.

Syntax the SEND TEXT command is:

```
EXEC CICS SEND TEXT
          FROM(data-value)
          LENGTH(data-value)
          [HEADER(data-value)]
          [TRAILER(data-value)]
          [ERASE] [ACCUM] [PAGING]
END-EXEC.
```

When to code ACCUM?

The text stream can consist of multiple paragraphs and the building of message can happen in multiple stages. If you want to send the text stream only after the building of all the paragraphs, use ACCUM option for the SEND-TEXT of every individual paragraph being built. They are accumulated and will not be sent immediately.

Once the page is built with complete message of all the paragraphs, issue SEND PAGE command that will send the complete text-stream to the terminal. HEADER is placed in the first SEND-TEXT and TRAILER is placed in the SEND PAGE.

When to code PAGING?

ACCUM alone is enough if the text-stream can fit within a single page. But if the text stream is expected to exceed one page, then code PAGING in all the SEND-TEXT commands in addition to ACCUM option. The last SEND-PAGE command sends the first page of message to the terminal.

The user can enter paging commands or keys to walk through the pages. To use keys instead paging commands, the keys should be mapped with paging commands in the System Initialization Table (SIT). Usually there will be a mapping for PF7 and PF8 with page up and page down commands respectively. PURGE MESSAGE is used to purge the message that is accumulated but not yet send.

Format of HEADER and TRAILER

If TEXT-HEADER is the variable used in HEADER command and the header is 'ORDER INVENTORY –Page nn- ', then the variable TEXT-HEADER should be defined as follows in working-storage section.

```
01 TEXT-HEADER.
    05 FILLER PIC S9(4) COMP VALUE 27.  => Length of the text.
    05 FILLER PIC X  VALUE '&'.         => Character identifying automatic
                                          page number in the text.

    05 FILLER PIC X.                    => One byte control field used by BMS.
```

```
05 FILLER PIC X(27) VALUE 'ORDER INVENTORY – Page && - '.
                                    => Actual text in the header.
```

TRAILER also can be coded in the same way. Automatic- page-number-identifying character will be spaces for TRAILER working storage variable.

ACCUM and PAGING can be also used with SEND-MAP and the meaning is same.

SEND-CONTROL

When the MAP is sent, we can specify device control options like FREEKB ALARM ERASE, along with SEND-MAP. If one wants to issue the device control options before sending the map, SEND CONTROL will be useful. It is used to establish the device control options dynamically.

Syntax:

```
EXEC CICS SEND CONTROL
          [CURSOR(data-value)] [ERASE | ERASEUP] [FREEKB] [ALARM] [FRSET]
END-EXEC
```

PRINT Through BMS

PRINT option of SEND MAP command allows the reports to be printed at the local printer connected to the terminal. In this case, MAP is sent to the printer and not to the terminal.

Syntax:

```
EXEC CICS SEND MAP('MAP-NAME')
                MAPSET('MAPSET-NAME')
                PRINT
                NLEOM
END-EXEC.
```

NLEOM option is recommended with PRINT option. When this option is used,

1. BMS builds the data stream using new line characters and blank characters to position the fields on the printer page.
2. The data stream is terminated by EOM (end of message) that stops printing. So printer buffer is efficiently used, allows larger pages to be printed from the same buffer that makes the printing faster.

Message Routing

A message can be routed to one or more terminals other than the direct terminal with which the program has been communicating. The message eligible for message routing is, a message constructed by the SEND MAP command with the ACCUM option.

ROUTE command establishes the message routing environment and the SEND PAGE command issued after ROUTE command sends the message to the destination.

Syntax:

```
EXEC CICS ROUTE [LIST(data-area)], [OPCLASS(data-area)],
                        [INTERVAL(hhmmss)|TIME(hhmmss)],
                        [TITLE(data-area)], [ERRTERM(name)]
END-EXEC.
```

LIST and OPCLASS name the route list and operator class codes respectively.
INTERVAL / TIME determines the actual timing of message delivery in the time interval or the time respectively.
TITLE names the title field defined in the working storage section and
ERRTERM specify the terminal ID where the error message (if any) to be sent.

Route list is prepared in working storage using the following convention.

TTTTrrOOOsrrrrrr – 8 bytes named 'r' are declared as spaces. TTTT names the terminal identifier as in TCT and OOO specify the operator id as in SNT. s is status flag. Code as many 16 bytes fields as the destinations and indicate end of route list is with the declaration of half word binary field with –1 value.

The message can be routed to every terminal at which users of the specified operator class is signed on. This is done using OPCLASS.

Program Design

Pseudo logic in CICS programs is achieved in three ways.

1.Multiple Programs and Multiple Transaction Ids.

The conversation program is logically and physically divided into multiple programs and each program is registered with one transaction ID. Each program issues RETURN command with next transaction ID. So after the screen is sent, the first program will be terminated. Once the user filled up the information in screen and press ENTER, the next program of the next transaction ID (returned by the first program) gets control and the same process is followed for n programs.
Advantage: Easy development and maintenance.
Disadvantage: Possible Code redundancy and number of PCT and PPT entries.

2.Multiple Transactions and single program.

First paragraph of the procedure division controls the flow the different sections of the program based on the transaction identifier stored in EIBTRNID. Every section ends with RETURN command with next transaction to be invoked on completion of the screen information by the user.
Thus the conversational program is logically divided than physical division.
Advantage: PPT entries are less compared to case 1.
Disadvantage: Number of PCT entries are still the same as case 1.

3.One transaction and One program. (Preferred way)

RETURN command has an optional parameter of COMMAREA and LENGTH. Using this parameter we can pass information between the transactions.

So first paragraph of your program will check whether this is the first entry or nth entry. If EIBCALEN is zero, this is first entry. So divert the program to the section that will do all initial processing and throw the first screen for the user.

In addition to this it updates COMMAREA with some control information like internal transaction identifier. This COMMAREA is passed along with RETURN TRANSID. So EIBCALEN is updated with LENGTH of COMMAREA. The second and nth entrances don't have EIBCALEN as zero and so based on the information in the COMMAREA, transfer the control to different sections of the program.

Task Initiation Process

1.User entered transaction identifier.

2.Terminal Control Program along with Terminal Control Table recognizes the incoming data from the terminal.

3.Storage Control Program acquires the storage for the terminal input output area (TIOA)

4.Terminal Control Program places the data from the terminal into TIOA and sets the pointer into TCT entry of the terminal.

5.If there is no task is associated with the terminal, then TCP gives the control to task Control Program, which realizes the transaction identifier in the data in TIOA. (KCP)

6.SCP acquires the storage area for task control area (TCA) in which KCP prepares the control data for this task.

7.KCP refer PCT to identify the program associated with the transaction.

8.Program Control Program (PCP) loads the program into main memory from the load library and gives the control back to KCP.

9.KCP gives the control to application program loaded into main memory.

10.Program started processing – Task is initiated.

Ways of Data Passing between transactions.

1. By COMMAREA.

In the example below, Working Storage item WS-ITEM of length WS-LENGTH is passed to the transaction EMPC. The program for the transaction 'EMPC' should have DFHCOMMAREA of size WS-LENGTH in the linkage section to receive the information passed by RETURN of this program.

```
EXEC CICS
RETURN TRANSID('EMPC') COMMAREA(WS-ITEM) LENGTH(WS-LENGTH)
END-EXEC
```

WS-LENGTH is full word binary and the maximum length that can be specified is 64K. So we can pass data of maximum size 64K. But it is recommended not to pass more than 24K.

LINK and XCTL can also have use COMMAREA to pass the information to the program being called and the concept is same.

2. Queues.

There are two types of queues TSQ and TDQ. TSQ can be used as scratch pad in main memory. You can write as much as you want in TSQ and they will be available to all the transactions that are aware of the queue name. TSQ is primarily used to share huge amount of information across the transactions. Refer Queues section for more details.

3. TCTCA, TWA, CWA.

TWA – Transaction work area – One per task - Work area associated with the task.

TCTUA – Terminal Control Table User Area – Work area associated with a terminal and defined as one per terminal in TCT.

CWA - Common work area – System work area defined by system programmer in SIT. (System Initialization Table)

We can use TWA, TCTUA and CWA for sharing the information across the transactions.

External address ability using BLL CELLS of OS/VS COBOL

Base Locator for Linkage is used to access the memory outside you working storage. If it is used in input commands like READ, RECEIVE, the performance will be good as we would be directly accessing the input buffer than working storage item. If you want to access system areas like CWA, TCTUA or CSA that exist outside your program, then you should use BLL.

Code Linkage section as below:

```
01 PARM-LIST.
       05 FILLER          PIC S9(08) COMP.
       05 TCTUA-PTR       PIC S9(08) COMP.
       05 TWA-PTR         PIC S9(08) COMP.

01 TCTUA-AREA.
       05 TCTUA-INFORMATION PIC X(m).
01 TWA-AREA.
       05 TWA-INFORMATION PIC X(n).
```

The mapping between the pointers and data area is done in linkage section as follows.
First filler is points the current 01 level, which is PARM-LIST itself.
TCTUA-PTR points to next 01 level, which is TCTCA-PTR
TWS-PTR points to next 01 level, which is TWA-PTR.

In the procedure division, code the following:
EXEC CICS ADDRESS TWA(TWA-PTR) TCTUA(TCTUA-PTR) END-EXEC.
SERVICE RELOAD TCTUA-AREA
SERVICE RELOAD TWA-AREA

This will store the pointer of TWA in TWA-PTR and TCTUA in TCTUA-PTR.
As the mapping between BLL cells and areas is already done in linkage section, TCTUA-AREA can access m bytes of TCTUA, which exist outside your working storage and similarly TWA-AREA to access n bytes of TWA, which exist outside your working storage area.

To ensure the addressability, SERVICE RELOAD statement should follow whenever the content of BLL cell is changed in anyway. So there should be a SERVICE RELOAD statement after the ADDRESS statement as follows.

Enhancement by VS COBOL2

ADDRESS OF operator of VS COBOL2 easies the access of external items.
The above requirement can be met in VS COBOL2 as follows. There is no need for PARM-LIST or SERVICE RELOAD.

```
LINKAGE SECTION.
01 TCTUA-AREA.
       05 TCTUA-INFORMATION PIC X(m).
01 TWA-AREA.
       05 TWA-INFORMATION PIC X(n).

PROCEDURE DIVISION.
EXEC CICS ADDRESS TCTUA (ADDRESS OF TCTUA-AREA)
                        TWA(ADDRESS OF TWA-AREA)
END-EXEC.
```

ADDRESS OF maps the TCTUA-AREA with TCTUA exists outside your working storage.

ASSIGN statement

It is used to access the system value outside of application program.
Some of them are length of common areas, user-id.
CWALENG, TCTUALENG, TWALENG – assign to full word binary working storage item.

USERID - assign to X(08) field.

```
EXEC CICS ASSIGN
          USERID(WS-USERID)
END-EXEC
```
The above command fetches the user id into WS-USERID.

EIB-Executive Interface Block

When CICS translator translates your program, it adds DFHEIBLK copybook as the first entry in your linkage section. Whenever a task is initiated, the task related information could be accessed using the fields in the copybook. EIB is acronym for Executive Interface Block.

Some of the fields in EIB are:

EIBCALEN	– Length of DFHCOMMAREA.
EIBDATE and EIBTIME	– Date and Time of task initiation.
EIBAID	- The last attention identifier pressed by the user.
EIBFN	- Function code of the last command. (2 bytes)
EIBRCODE	- Response code of the last command. (6 bytes)
EIBTRMID	- Terminal ID
EIBTRNID	- Transaction ID

CICS PROGRAM LINKAGE SECTION

If you look into the translated source listing, the linkage section of CICS program might have the following.

1. DFHEIBLK (Introduced by Translator)
2. DFHCOMMAREA (Coded by you as the first entry in linkage section.)
3. PARM-LIST (BLL cells coded by you)
4. Multiple 01 levels (The number of 01 entries is equal to number of BLL cells you have coded in PARM-LIST excluding the first BLL cell that points to PARM-LIST itself.)

GETMAIN and FREEMAIN.

GETMAIN command is used to obtain a certain amount of storage in the main memory.

```
EXEC CICS GETMAIN SET(ADDRESS OF LK-ITEM)
              LENGTH(data-value)|FLENGTH(data-value)
              INITIMG(data-value)
END-EXEC.
```

LK-ITEM is a linkage 01 item whose length is equal to length mentioned in the LENGTH parameter and that much memory will be allocated for you in the main memory. The initial value of this field will be set by INITIMG parameter.

```
EXEC CICS FREEMAIN DATA(LK-ITEM)
END-EXEC
```
The above command will release the main memory occupied by you using GETMAIN.

These commands are highly useful in macro level coding where the quasi-reentrancy is the responsibility of the application programmer. Quasi-reentrancy is

guaranteed in the CICS command level COBOL programs and so the need and usage of these commands are almost obsolete.

LINK XCTL and CALL

It is good practice to develop several functional modules rather than building one lengthy program that does all the functions. It makes the program tough to maintain, understand and debug. The size of the program will be large and it also occupies more resources. Functional splitting of sub modules and calling or transferring the control to them is achieved by LINK, XCTL and CALL.

CICS programs are usually run at various levels. The first program, which gets the control from CICS, that is the program registered against the transaction in PCT, runs at first level. Linked and Called programs are running one level lower than linking or calling program and so the RETURN or GO BACK in these programs will give the control back to Linking or calling program. XCTL programs run at the same level and so the RETURN gives control to next upper level.

```
Syntax: EXEC CICS LINK|XCTL PROGRAM(name) COMMAREA(ws-area)
                    LENGTH(ws-length)  END-EXEC.
```

Ws-length is length of ws-area and ws-area has the information that needs to be passed to LINK/XCTL program. VS COBOL 2 allows reentrant program and so whatever is achieved using LINK or XCTL can be also achieved by COBOL CALL statement. The called programs should be attached to the main program during link edit and so the size of the load module will be large in this case.

Prefer XCTL whenever possible as it involves fewer overheads. If that is not possible then based on sub-program size and possible modification, go for LINK or CALL. If the size of the program is less and used by only very few programs go for CALL as it wont increase the size much at the same time application will run fast.

If the program is expecting more changes, then calling program also need to be compiled for every change in sub program. In case of LINK, the compilation of linked programs is enough. Called CICS programs need not be registered in PPT whereas LINK and XCTL programs should be.

LOAD and RELEASE

LOAD is used to load the program or table that is compiled or assembled, link edited and registered in PPT. It is mostly used for loading the assembler table.

```
EXEC CICS LOAD PROGRAM(PGM1) SET (ADDRESS OF LK-ITEM)
                                        LENGTH(WS-LENGTH)
END-EXEC
```

Program PGM1 is loaded and the address of the program or table is mapped to LK-ITEM and so the table can be accessed using the linkage are LK-ITEM. The size

of the linkage item is WS-LENGTH. If 'HOLD' keyword is coded in the LOAD command, then loaded program or table will be permanently resident until explicitly released. If it is not mentioned, then termination of the task release the program or table.

EXEC CICS RELEASE PROGRAM(PGM1) END-EXEC is used to release the program or table.

AID and HANDLE AID

There are certain attention identifiers (AID) associates with every screen. Entry screens have ENTER and PF1 (Help) keys. Query screens have PF7 (Page Up), PF8 (Page down), PF1 (Help) and so on. Based on the key pressed by the user, the processing should happen in the program. EIBAID of DFHEIBLK has the last user pressed key. It can be verified in the program after the RECEIVE command.

For verification of values in EIBAID, Copy the standard AID list copybook provided by CICS in your program (COPY DFHAID). Now you can easily verify the key pressed as follows.

```
EVALUATE EIBAID
WHEN DFHENTER PERFORM PARA-1
WHEN DFHPF1 PERFORM PARA-2
WHEN OTHER PERFORM PARA-3
END-EVALUATE.
```

The above checking is done in COBOL and this kind of check has to be done after every RECEIVE command to properly route the flow. CICS provides its own routing processing based on the key pressed by HANDLE AID command. This will be effective through out the program and reduce the code redundancy. The routing will be automatically invoked on every RECEIVE.

```
EXEC CICS HANDLE AID
        DFHENTER(PARA-1)
        DFHPF1(PARA-2)
        ANYKEY(PARA-3)
END-EXEC.
```

HANDLE CONDITION and NO HANDLE

Every CICS command has its own exception list. One example is MAPFAIL exception of RECEIVE command. HANDLE CONDITION is used to transfer control to the proper procedure on expected exceptions. HANDLE condition cannot track program interrupted ABENDS like S0C4 or S0C7. It deals only with exceptions of CICS commands.

The conditions handled are effective from where it appears to the end of the program. One Handle Condition can be overridden by another condition. The main program conditions will not be effective in sub-programs.

```
EXEC CICS HANDLE CONDITION
        MAPFAIL(PARA-1)
        PGMIDERR(PARA-2)
        LENGERR(PARA-3)
        ERROR(PARA-X)
END-EXEC.
```

= >Any error condition other than MAPFAIL,PGMIDERR and LENGERR will transfer the control to PARA-X.

If you want to reset the diversion done by previous HANDLE condition, code the condition name but don't mention the paragraph name. Maximum 12 conditions can be coded in one HANDLE CONDITION statement.

If you want to deactivate all the conditions for a particular CICS command, code NOHANDLE. There are possibilities for infinite loop when the CICS command in the exception routine of the HANDLE CONDITION ends with same exception. In the above example, if the there is LENGERR in PARA-3 for any of the CICS command coded there, then the control again come to PARA-3 and forms infinite loop. In such cases, NOHANDLE will be useful (in the ABEND routine CICS commands).

IGNORE CONDITION.

The syntax is same as HANDLE CONDITION but it causes no action to be taken if the specified condition occurs in the program. The control will be returned to the next instruction following the command, which encountered the exceptional condition. It will be effective from the place where it appears to the end of the program or until any HANDLE condition overrides it.
Maximum 12 conditions can be coded in one IGNORE CONDITION statement.

RESP

Like the file status verification of batch COBOL program, SQLCODE verification of DB2 program, the success or failure of CICS command can be done using COBOL statements and this give more structured look for the program.

To achieve this, code RESP along with CICS command. CICS will place the result into the variable coded in RESP. It can be checked against DFHRESP(NORMAL) or DFHRESP(condition) and appropriately control the flow following the command.

```
EXEC CICS RECEIVE MAP(A) MAPSET(B) RESP(WS-RCODE) END-EXEC
EVALUATE WS-RCODE
WHEN DFHRESP(NORMAL) PERFORM PROCESS-PARA
WHEN DFHRESP(MAPFAIL) PERFORM PARA-1
WHEN OTHER PERFORM PARA-X
END-EVALUATE.
```

WS-RCODE should be defined as full word binary in working storage. If you code RESP, then HANDLE CONDTION will not be effective for those CICS commands.

PUSH and POP.

They are used to suspend and reactivate respectively, all the HANDLE CONDITION requests currently in effect. In real life programming, you may want to deactivate all the active handle conditions and diversions for a subroutine (section) embedded in the program. This can be achieved using PUSH HANDLE. When you come out of that section, you can reactivate all the pushed commands and that can be done using POP HANDLE.

HANDLE ABEND

HANDLE CONDITION command intercepts only abnormal conditions of the CICS command execution whereas HANDLE ABEND takes care of any abnormal termination within the program.

```
EXEC CICS HANDLE ABEND
          LABEL(ABEND-ROUTINE)|RESET|CANCEL
END-EXEC.
```

LABEL is to activate an exit. CANCEL is used to cancel the previously established HANDLE ABEND request. RESET is to reactivate the previously cancelled HANDLE ABEND request.

USER-ABEND

In batch COBOL, the user ABENDS can be thrown by calling the assembler routine ILBOABN0 using AB-CODE whereas AB-CODE is working storage field of half word binary.
The following command is used to throw user ABENDS in CICS.
EXEC CICS ABEND ABCODE('9999')
END-EXEC is used to throw user ABEND 9999.

FILE-HANDLING.

CICS supports only VSAM and BDAM files. All the files, that you want to use in your CICS application, should be registered in FCT with their complete attributes. CICS commands refer the FCT entry name for doing operation on the file.

As all the files are already declared and defined in tables, coding File-Control Paragraph of ENVIRONMENT division or FILE SECTION of DATA DIVISION is meaningless and not required. Thus CICS frees the application program from any data dependant coding.

The unit of IO during READ is one control interval. So even you read one record into your program, the complete control interval is read into main memory buffer. The size of control interval is preferred to be large for sequential processing and small for random processing.

The file should be OPEN to issue an I-O command. The status of the file can be queried using the master transaction CEMT. It can be OPENED, CLOSED, ENABLED or DISABLED. This explicit opening or closing can be done using CEMT. But this needs human intervention.

CICS Version 1.7 introduces automatic opening of the file if it is not in open status during the access. It is always better to close it the when you no longer need it. DFOC (Dynamic File Open Close) can be done in program using the SET command or linking to DFHEMTP.

DFOC BY LINK COMMAND

The application program links the master terminal program (DFHEMTP) and passes CEMT parameter data required for the file open/close through COMMAREA. Receiving the data, DFHEMTP program opens/closes the files specified, after which the control is returned to the application program.

```
EXEC CICS LINK PROGRAM(DFHEMTP)
                    COMMAREA(WS-COMMAREA)
                    LENGTH(WS-LENGTH)
END-EXEC
```

The file open command is written into WS-COMMAREA and its length is populated in WS-LENGTH.
File Open Command: SET DATASET(FCT-NAME) OPENED

DFOC BY SET COMMAND

```
EXEC CICS SET
      DATASET(name) | FILE(name)
      OPEN|CLOSED
END-EXEC
```

DFOC BY BATCH JOB

DFOC can be achieved from a batch job, which issues the CEMT as the data for the JES Modify (F) command against the system console terminal.

```
// F cicsjob, 'CEMT SET DATASET(FCTNAME) OPENED'
```

RANDOM ACCESS OF FILES

```
READ: EXEC CICS   READ FILE(FCT-NAME)  [UPDATE]                    S1
                  INTO(WS-ITEM) | SET (ADDRESS OF LK-ITEM)         S2
                  LENGTH(WS-LENGTH)                                S3
                  RIDFLD(WS-KEY)                                   S4
                  KEYLENGTH(WS-KEY-LENGTH) GENERIC                 S5
                  SYSID(SYSTEM-NAME)                               S6
                  GTEQ | EQUAL                                     S7
      END-EXEC.
```

S1. Reads the file and if update is intended then code UPDATE clause. This will get exclusive access over the complete control interval where the specific record exists, during the READ. Thus CICS ensures data integrity.

S2. Read the file into working storage variable WS-ITEM. Alternatively, the address of the record in the input-output area can be mapped to linkage section variable by using SET command. Performance-wise SET is better than READ INTO. VS COBOL2 supports ADDRESS of keyword and makes the code easier. In the prior version COBOL, PARM list and SERVICE RELOAD should be used to achieve the same.

S3. After the READ, the LENGTH of the record READ is updated into WS-LENGTH. WS-LENGTH is the working storage half word binary item.

S4. Key of the record to be read is moved into WS-KEY, a working storage item and a part of record structure (WS-ITEM).

S5. If partial key is used, then the length of the partial key should be moved to WS-KEY-LENGTH and the keyword GENERIC should be coded. This is optional for full-key read.

S6. Remote system name where the request to be directed, is coded here. (1-4 character name)

S7. EQUAL is default. GTEQ can be coded when you know the full key, but you are not sure the record with that key exists in the file.

REWRITE

```
EXEC CICS REWRITE FILE(FCT-NAME)
                  FROM(WS-ITEM)
```

```
                    LENGTH(WS-LENGTH)
                    SYSID(SYSTEM-NAME)
END-EXEC.
```

To release the exclusive access acquired during the READ with UPDATE, the record should be REWRITTEN using the above syntax. The parameters are self-explanatory.

After you read record, if you feel that you don't want to update it, then inform the same to CICS by issuing UNLOCK command so that CICS release the lock acquired by you on the record.

EXEC CICS UNLOCK FILE(FCT-NAME) END-EXEC.

WRITE

The syntax of WRITE is same as REWRITE. There is a parameter RIDFLD with Key-value is coded in the WRITE command. (Like S4 for READ command)

When you want to add a set of records whose keys are in ascending order, then qualify your first WRITE with another parameter MASSINSERT. This will get exclusive control over the file and provide high performance during the mass insert.

If you use MASSINSERT, then you should issue UNLOCK to inform the completion of your additions. CICS releases the file on UNLOCK.

DELETE

1.The record read using READ with UPDATE can be deleted using

EXEC CICS DELETE DATASET(FCT-NAME) END-EXEC.

2.The record in the file can directly be deleted by providing complete key.

EXEC CICS DELETE DATASET(FCT-NAME) RIDFLD(WS-KEY) END-EXEC.

3.Group of records can be deleted using partial key. After the deletion, the number of records deleted is updated in the variable coded in NUMREC parameter. (WS-DEL-COUNT)

```
EXEC CICS DELETE   DATASET(FCT-NAME)
                   RIDFLD(WS-KEY)
                   KEYLEGNTH(WS-KEY-LENGTH) GENERIC
                   NUMREC(WS-DEL-COUNT)
END-EXEC
```

SEQUENTIAL READ:

Sequential access of VSAM file under CICS is called Browsing. It has FIVE Commands associated with it.

STARTBR.

It establishes a position to start browsing.

```
EXEC CICS STARTBR FILE(FCT-NAME) RIDFLD(WS-KEY)
                  KEYLENGTH(WS-KEY-LENGTH) GENERIC
                  REQID(Integer-value)
                  SYSID(SYSTEM-NAME)
                  GTEQ | EQUAL
END-EXEC.
```

Meaning of the parameters is same as READ operation explanation. When you want to do multiple browsing concurrently over the same file, then use REQID. The

first STARTBR can be coded with REQID(1) and the second STARTBR can be coded with REQID(2).

One browse operation occupies one string of VSAM. If all VSAM strings are exhausted for one VSAM file, then the other transactions will have to wait for one of the strings to become free. So it is not recommended to use multiple browsing. Instead, once the browsing is completed, using RESETBR set the position to another place and start browsing. The syntax of RESETBR is same as STARTBR.

READNEXT and READPREV.

```
EXEC CICS READNEXT| READPREV FILE(FCT-NAME) RIDFLD(WS-KEY)
                              INTO(WS-ITEM)
                              LENGTH(WS-LENGTH)
                              KEYLENGTH(WS-KEY-LENGTH)
                              REQID(INTEGER-VALUE)
                              SYSID(SYSTEM-NAME)      END-EXEC.
```

READNEXT is used to read the records in the forward direction from the position established by STARTBR. READPREV is used to read the records in the reverse direction. READPREV followed by READNEXT retrieves the same record once again.

During the browse, if you want to skip set of records, then move the key of the next record you intended to read to RIDFLD and then issue READNEXT. This is called skip sequential read. Thus RIDFLD can be used as both input as well as output field.

As VSAM files are variable length in most of the cases, WS-LENGTH should be populated with maximum record length before issuing READ command. KEYLENGTH(0) will position the cursor at the beginning of the file.

If you specify READPREV immediately after STARTBR, then you should code key of a record that exists on the dataset. Otherwise NOTFND condition occurs on READPREV.

ENDBR

It is used to terminate the current browse function.

```
EXEC CICS ENDBR FILE(FCT-NAME) REQID(INTEGER-VALUE) SYSID(SYSTEM-NAME)
END-EXEC.
```

UPDATE during BROWSE

Update and Browse are mutually exclusive functions. So if you want to update a record during the browse operation, first issue ENDBR. Then give random read using the key in hand with UPDATE option. REWRITE the record. Then issue STARTBR with the key in hand and continue the browsing.

ESDS-BROWSING

Define a full-word binary item in working storage for the RBA of ESDS file. Move LOW-VALUES or HIGH-VALUES to the field for the forward or reverse read respectively. Issue STARTBR with RBA and EQUAL option. RIDFLD should point to

defined RBA field. Now issue READNEXT or READPREV for forward or REVERSE read respectively.
GTEQ is invalid for ESDS.

```
MOVE LOW-VALUES TO ESDS-RBA
EXEC CICS STARTBR FILE(FCT-NAME) RIDFLD(ESDS-RBA) RBA EQUAL END-EXEC
MOVE 226 to WS-LENGTH.
EXEC CICS READNEXT FILE(FCT-NAME) INTO(WS-ITEM) RIDFLD(ESDS-RBA)
LENGTH(WS-LENGTH) RBA
END-EXEC.
```

ESDS-WRITE

The syntax is same as KSDS WRITE command. Qualify the WRITE with RBA option. The record will be appended to the file and the RBA of the record is placed into RBA field mentioned in the WRITE command. (ESDS-RBA)

```
MOVE 225 TO WS-LENGTH
EXEC CICS WRITE FILE(FCT-NAME) RIDFLD(ESDS-RBA) LENGTH(WS-LENGTH) RBA
END-EXEC.
```

ESDS-RANDOM ACCESS

If you know the RBA value of the record you want to access, then you can randomly access the ESDS file. The syntax is same as READ of KSDS file but qualify it with RBA option. RIDFLD should point to full-word binary item pre-filled with RBA of the record to be accessed.

RRDS ACCESS

RRDS file can be accessed using RRN in place of RBA and populating the RIDFLD with RRN number. The field should be full word binary.

Alternate Index access

Register the path in FCT and use the path name as file name to read the file randomly using alternate index. Please refer VSAM section of the book for understanding what exactly PATH is.

As the alternate key need not be unique, DUPKEY condition is very common during the alternate index usage and so it has to be properly handled.

ASKTIME

EIBDATE and EIBTIME have the values at task initiation time. Upon the completion of ASKTIME, these two fields are populated with current date and time.

```
EXEC CICS ASKTIME END-EXEC.
```

FORMATTIME

FORMATTIME is used to receive the information of date and time in various formats.

```
EXEC CICS FORMATTIME FORMAT-TYPE(data-area) END-EXEC
Format-type:  YYDDD, YYMMDD, YYDDMM, MMDDYY, DDMMYY, DAYOFWEEK,
              DAYOFMONTH, MONTHOFYEAR, YEAR, TIME, TIMESEP, and
              DATESEP.
Example:
    EXEC CICS
        FORMATTIME MMDDYY(WS-DATE) DATESEP TIME(WS-TIME) TIMESEP
    END-EXEC
```

WS-DATE contains the current date in MM/DD/YY (Default DATESEP is ‘/’)
WS-TIME contains the current time in HH:MM:SS (Default TIMESEP is ‘:’)

DELAY and SUSPEND

It is used to delay the processing of a task for a specified time interval or until the specified time. If your task is doing some heavy CPU bounded work, it is a good to place the DELAY command in order to allow the other tasks to proceed.

```
EXEC CICS DELAY INTERVAL(HHMMSS) | TIME(HHMMSS) END-EXEC.
```

SUSPEND is used to suspend a task. During the execution of the command, the task will be suspended and the control will be given to other tasks with higher priority. As soon as all higher priority tasks have been executed, control will be returned to the suspended task.

```
EXEC CICS SUSPEND END-EXEC.
```

POST and WAIT EVENT

POST is used to request notification when the specific time has expired and the WAIT EVENT command is used to wait for an event to occur. Usually these two commands are used in pair, as alternative to the DELAY command.

```
EXEC CICS POST INTERVAL(HHMMSS) | TIME(HHMMSS)
        SET (ADDRESS OF POST-ECB) END-EXEC.
END-EXEC
EXEC CICS WAIT EVENT ECAADDR(ADDRESS OF POST-ECB)
END-EXEC
```

CANCEL

It is used to cancel the interval control commands such as DELAY, POST and START which have been issued. The interval commands to be cancelled are identified using REQID.

```
EXEC CICS START REQID('STARTS') END-EXEC.
EXEC CICS CANCEL REQID('STARTS') END-EXEC.        Cancels all the starts issued.
```

START and RETRIEVE

START command is used to start a transaction at the specified terminal and at the specified time or interval. Optionally, data can be passed to the to-be-initiated transaction.

```
EXEC CICS START    TRANSID('EMPC')  TERMID('TST1')                S1
                   TIME(123000)|INTERVAL(003000)                  S2
                   FROM(WS-ITEM) LENGTH(WS-LENGTH)                S3
                   RTRANSID('EMPD') RTERMID('TST0')               S4
                   QUEUE('EMPDET')                                S5
END-EXEC.
```

S1. EMPC is to be started at the terminal TST1

S2. TIME and INTERVAL are mutually exclusive. If TIME(HHMMSS) is coded, then the task start at the terminal at the specific time. In the above example it is 12:30. If INTERVAL(HHMMSS) is coded then the task will start after the expiry of the interval. In the example, the task starts after 30 minutes.

S3. This program passes WS-ITEM of size WS-LENGTH to the task EMPC.

S4. Transaction ID EMPD and Terminal Id TST0 are passed to EMPC. EMPC can receive this information and may use for next triggering.

S5. TSQ queue name can be passed from this transaction. EMPDET is passed in our case.

The data passed by START command is received by the new transaction using RETRIEVE command upon the expiry of START command.

```
EXEC CICS RETREIVE INTO(WS-ITME)| SET(ADDRESS OF LK-ITEM)
                   LENGTH(WS-LENGTH)
                   RTRANSID(WS-TRAN) RTERMID(WS-TERM)
                   QUEUE(WS-QUEUE)
END-EXEC.
```

ENQ and DEQ

ENQ gets the exclusive control over the resource and DEQ releases the exclusive control acquired over the resource. As CICS takes care of exclusive access over the resource whenever needed, it is advisable not to use it in application programs.

But sometimes it may be needed in few cases like updating TSQ records or getting access of the printer.

```
EXEC CICS ENQ/DEQ RESOURCE(QUEUE-NAME) END-EXEC.
```

SYNCPOINT and TWO-PHASE-COMMIT

The updates done by a task is automatically committed at task termination. SYNCPOINT is used to commit portion of work completed without terminating the task.

EXEC CICS SYNPOINT END-EXEC.

During the program processing, if you want to roll back the changes made by you, then you can code SYNPOINT with ROLLBACK. This will backed out all resource modifications done since the last sync point.

EXEC CICS SYNCPOINT ROLLBACK END-EXEC.

SYNCPOINT in CICS-DB2 environment is called as two-phase commit. The changes made in the resources that are directly under the control of CICS are committed in the first phase and the changes made to DB2 environment are committed in the second phase.

Queues

There are two types of queues available in CICS.
1.Temporary Storage Queue (TSQ)
2.Transient Data Queue (TDQ)

Their properties are listed below:

TSQ	No need to predefine anywhere. By default, TS queues are created in main memory. So the content cannot be recovered after system crash. If you want to recover, then you should define them in TST. Records can be randomly accessed using ITEM option of READQ. READ is not destructive. Deletion of queue deletes all the records in it. Deletion of recoverable TSQ should follow sync point before next WRITEQ. Primarily used as scratch pad memory facility for any purpose. Example: Page-up and Page-down logic, Passing huge data in between the phases of transaction, Review and correction of multiple order entry screens.
TDQ	TDQ should be predefined in DCT. READ is destructive and only sequential. Once the record is READ, it will be logically deleted and cannot be read again. Automatic Task Initiation: When number of records in the queue exceeds the TRIGLEV defined in the DFHDCT entry of the queue, the transaction coded in the TRANSID of DFHDCT is automatically triggered. This ATI is possible only in TDQ. There are two types of TDQ – Intra and Extra partition. The DCT entry identifies the type of queue from TYPE parameter. Intra Partition Queues are used within a CICS region and DELETEQ deletes all the records physically in the queue. Extra partition Queues are used across the regions and systems. If you want to pass some data from CICS to batch or receive data from Batch to CICS,

	you should go for extra partition queue. In the same program TDQ cannot be opened in both input and output mode. TSQ are preferred over TDQ for data passing. TDQ is used for batch interface or on ATI requirement.

MRO and ISC

Within one processor, there could be more than one CICS region, each of which runs independently under the same operating system. This is called Multi region Operation. (MRO) .It is achieved by four inter communication facilities.

1.Function Request Shipping.
2.Asynchronous Processing.
3.Transaction routing.
4.Distributed Transaction Processing.

One CICS in one processor can communicate with other CICS in other processor or other non-CICS system regardless of where the other processor physically located. This is called Inter System Communication (ISC). It requires sophisticated communication networks based on the System Network Architecture (SNA).

MASTER TRANSACTIONS

CEMT:

This is Master Terminal transaction. It is menu driven and easy-to-use transaction but due to its nature of manipulating the CICS environment, most of the functions are restricted to application programmer or end user.

CEMT INQUIRE|SET|PERFORM
CEMT INQ TRAN|PRO|FILE – Display information from PCT|PPT|FCT
CEMT INW TASK – Display the active running tasks in the region

CEMT SET command can be used to reset the values of PCT|PPT|FCT. PERFORM has to be handled carefully. CEMT PERFORM SHUTDOWN will shut down the entire CICS region.

Frequently used commands:
CEMT SET PR(PGM-NAME) NEW - To create a new copy of an application program
CEMT SET DA(file-name) CLO/OPEN - To close/open a file from CICS.

CECI:

This is Command Interpreter transaction. CICS commands can be pre-tested using this command before placing them into the program.

CEBR:

This is browse transaction. TSQ can be browsed using this transaction.

CEDF:

This is Diagnostic Facility transaction. If you want to debug your program step by step, then before typing transaction ID type CEDF so that debug mode will be set.

It intercepts a transaction at initiation and termination of a program, before and after execution of any EXEC CICS or EXEC SQL commands.

CMAC:

This transaction is used to get ABEND codes and messages.

Different ways of initiating a transaction in CICS

1.By entering transaction identifier.
2.By START command.
3.By RETURN TRANSID
4.By registering the programs in PLT, they will be automatically initiated during CICS startup.
5.Automatic Task Initiation of TDQ.
6.PF or PA keys could be defined in PCT to initiate a CICS transaction.

CICS Dump Analysis

Dump usually consists of Dump Header Area, TCA(User Area), TCA(System Area), Trace Table, Transaction Storage and Program Storage.

At the top of the ABEND dump, there is a dump header area, which provides ABEND code, TASK (the transaction ID), Program Status Word (PSW) and the contents of registers at the time of ABEND.

PSW.

It is a double word field containing the status information at the time of ABEND. 32nd bit says the addressing mode in use. If the addressing mode is 31 bits, then the bit will be '1' and the bits 33 to 63 contains the next sequential instruction (NSI), which would have been executed if the ABEND had not occurred.

First make sure that you have program compilation and link edited SYSPRINTS of the program ABENDED and dump prints of the ABENDED transaction is with you. Program should be compiled with LIST option.

Finding the statement caused the ABEND

1.Identify the address of next sequential instruction from the PSW.

2. From the program storage, identify the starting address of program storage.

3.Subtract the starting address of program storage from the NSI address identified in the first step, you will get the displacement of the NSI from the beginning of program storage.

4.Find the displacement of application program from the beginning of the load module in the link edit list and subtract this value from the calculated NSI displacement to get the real NSI displacement.

5.Find the statement in the compilation listing that corresponds to the OFFSET calculated in the previous step. The statement one above is the statement that caused the ABEND.

6.Analyse the statement for the cause of ABEND. This might have involved one or more fields. Most of the times, you can easily come to conclusion the cause of the

ABEND once you identified the statement. If you could not, then you may check the value of the field at the time of ABEND as follows.

Finding the value of the field in error.
1.Find the Base Locator (BL), Displacement and length of the field from the MAP of compilation listing.
2. Find the register number corresponding to the base locator from the bottom of compilation listing.
3. Find the address of this register in the dump.
4. Add the displacement to the register of the address to get the address of the field.
5. Locate the address in the dump to get the value of the field.

CICS TABLE ENTRIES

FCT

All the VSAM files, PATH between base cluster and alternate index and any BDAM datasets are to be registered in FCT to access them in the application program.

```
DFHFCT    TYPE     =DATSET| FILE,
          ACCMETH=BDAM | VSAM,
          DATASET= name | FILE=name,
          SERVREQ = (ADD, BROWSE, DELETE, READ, UPDATE),
          [FILESTAT= (ENABLED|DISABLED, OPENED|CLOSED)],
          [BUFND= n+1],
          [BUFNI=n],
          [STRNO=n],
          [DSNAME=name],
          [DISP=OLD | SHR],
          [BASE=name],
```

ACCMETH defines data access method.

Name mentioned in the FILE parameter is the one with which we use in the CICS file commands. The name should correspond to the DDNAME of the file specified in the JCL of the CICS job itself. If DSNAME and DISP are coded, then JCL entry is not needed.

SERVREQ lists the authorized input/output operations against the file. If other than the specified services is requested in the file control command, then INVREQ condition will occur.

FILESTAT indicates the initial file status when CICS initially starts. BUFND and BUFNI indicate the number of data buffers and index buffers respectively. Default value is one for BUFNI and two for BUFND. STRNO is the number of VSAM strings by which concurrent access to the file is allowed.

If the file being registered is path, then BASE indicates the FCT name of the base cluster.

PPT

All the CICS application programs and BMS map sets must be registered in PPT. If the program is not registered here, then the program is unrecognizable to CICS.

```
DFHPPT      TYPE=ENTRY,
            PROGRAM=name| MAPSET= name
            [PGMLANG=(ASSEMBLER | COBOL | PLI)],
            options..
```

ASSEMBLER is the default program language. All the map-sets are considered as written in assembler.

PCT

The control information of all CICS transactions must be registered in program control table (PCT).

```
DFHPCT          TYPE=ENTRY,
                TRANSID=name,
                TASKREQ=xxxx,              (PF1-PF24, PA1-PA3)
                PROGRAM=name,
                [DTIMOUT=mmss],
                [RTIMOUT=mmss],
                [RESTART= NO | YES],
                [TRANSEC=1 | DECIMAL],    (1-64)
                [DUMP=YES|NO],
                …Other options..
```

TRANSID – Transaction identifier name (1-4 characters)
PROGRAM – Name of the program associated with the transaction.
TASKREQ - Key associated with the transaction initiation.
DTIMOUT - Defines the timeout (waiting time) in case of deadlock.
RTIMOUT - Defines the time limit within which the user has to respond. If he didn't then the task will be cancelled.
RESTART – Automatic transaction is applied after the completion of the transaction recovery from the abnormal termination of the transaction.
TRANSEC –Transaction security code.
DUMP - Whether DUMP is to be taken in case of abnormal termination of the transaction.

TCT

All the terminals, which are to be under CICS control, should be registered in TCT. Terminal Control Program uses this table for identifying the terminals and performs all input/output operations against these terminals.

```
DFHTCT      TYPE=ENTRY,
            ACCMETH=VTAM,
            TRMIDNT=name,
            TRMTYPE=type,
```

FEATURE=(UCTRAN,..)
Options..

TRMIDNT is (1-4 characters) terminal ID.
TRMTYPE is type of terminal. Ex. IBM3270.
FEATURE indicates the terminal services CICS offers. UCTRAN is used to translate all the lowercase characters to upper case characters.

DCT

TDQ control information is registered here. Destination Control Program uses this table for identifying all TDQs and performs input/output operations against them.

Intra partition queue can be defined as follows.

```
DFHDCT    TYPE=INTRA,
          DESTID=name,
          [TRANSID=name],
          [TRIGLEV=number],
          [REUSE= YES|NO]
```

TYPE can be INTRA (Intra partition TDQ)

ATI:
TRANSID and TRIGLEV are used for automatic task initiation. If the number of records in TDQ exceeded the number of records mentioned against TRIGLEV, then transaction ID mentioned against TRANSID will be invoked automatically.

Once a record of intra partition TDQ is read by a transaction, the record is logically removed from, the queue. If REUSE = YES is coded, then the space occupied by it can be used by other records in future but the logically deleted records are not recoverable.

Extra partitioned queue can be defined as follows.

```
DFHDCT    TYPE=EXTRA,
          DESTID=name,
          DSCNAME=name-2,
          [OPEN=INITIAL|DEFERED]
DFHDCT    TYPE=SDSCI,
          DSCNAME=name-2,
          [TYPFILE=INPUT|OUTPUT|RDBACK]
```

OPEN=INITIAL means the file will be open at the CICS start-up time and DEFERED means the file will be closed until it is specifically opened by CEMT.
DSCNAME defines data control block and for one DSCNAME, a corresponding DFHDCT entry must be made with TYPE=SDSCI and the same DSCNAME, which in effect indicates DDNAME of extra partition dataset in JCL or CICS JOB itself.
TYPFILE indicates the file to be INPUT, OUTPUT or READ BACKWARD (RDBACK)

How to submit JCL from a CICS program?

JCL can be submitted using the SPOOLOPEN, SPOOLWRITE and SPOOLCLOSE commands.

- First you must open a spool for output using SPOOLOPEN
  EXEC CICS
  SPOOLOPEN OUTPUT NODE('*') USERID(INTRDR)
  TOKEN(report_token) RESP(response field)
  END-EXEC.

NODE('*') specifies a destination node to send the JCL to. USERID is set to the name of the internal reader INTRDR. A unique token will be allocated by CICS when the SPOOLOPEN is executed and placed in the field you specify using TOKEN(report_token). The token will be used as a sending field on subsequent commands. The token will be 8 bytes long.

- To write each line of the job to the spool use the SPOOLWRITE command.
  EXEC CICS
  SPOOLWRITE FROM(io_area) TOKEN(report_token)
  RESP(resonse_field)
  END-EXEC.

The "io_area" field should be the name of a data item containing the line of JCL. The "report_token" field should be the same as the 8 byte token returned from SPOOLOPEN. An end of job statement ('//' or '/*EOF') should be written as the last line.

- Finally you must close the spool using SPOOLCLOSE. (Note if you do not explicitly close the spool it will be closed when the transaction terminates, However it is good practice to close the spool explicitly)
  EXEC CICS
  SPOOLCLOSE TOKEN(report_token) RESP(response_field)
  END-EXEC.

Again the "report_token" field must be the same as the one allocated at SPOOLOPEN.

Notes:

- The RESP option should be coded on all the Commands and it is recommended that you also code RESP2, because additional information is returned for some exception conditions in this field.
- DCT entries and JCL DD statements are not needed when using this method.
- Note that in order to use this method DFHSIT SPOOL=YES must be coded in the CICS System Initialization Table. Check with your friendly local SysProg if you are unsure.
- Under OS/VS COBOL the SPOOLWRITE command had to have FLENGTH specified. FLENGTH specifies the length of the data being written in a fullword binary field. This field is optional in newer versions, if it is omitted then the length is assumed to be the length of the data item (io-area) specified in FROM.

Be aware of the performance considerations for writing to the spool, IBM says "Transactions that process SYSOUT data sets larger than 1000 records, either for INPUT or for OUTPUT, are likely to have a performance impact on the rest of CICS". (Source: CICS/ESA V4R1 Application Programming Reference SC33-1170-00)