



CICS

customer information control system

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

CICS stands for Customer Information Control System. CICS was developed in 1968 by IBM. CICS allows users to develop and execute online application in an MVS environment. CICS has become the most commonly used server for Internet applications.

CICS is a transaction processing system which is also called as Online Transaction Processing (OLTP) Software. CICS is a data communication system that can support a network containing hundreds of terminals.

Audience

This tutorial is designed for software programmers who would like to understand the concepts of CICS starting from scratch. This tutorial will give you enough understanding on CICS from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of COBOL programming. A basic knowledge of MVS and TSO/ISPF subsystem will help you grasp the concepts of CICS better.

Disclaimer & Copyright

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute, or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher. We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness, or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience	i
Prerequisites	i
Disclaimer & Copyright	i
Table of Contents	ii
1. CICS – OVERVIEW	1
Functions of CICS	1
Features of CICS.....	2
2. CICS – ENVIRONMENT	3
CICS Environment	3
Data Communication Services	4
Data Handling Services	4
Application Programming Services	4
Monitoring Services.....	5
3. CICS – BASIC TERMS.....	6
IBM 3270 Terminal	6
CRT Monitor	6
Keyboard.....	7
Transaction.....	8
Task.....	8
LUW.....	9
Application	9
4. CICS – NUCLEUS	10
Control Programs	10
TCP	10
KCP	10

PCP	10
FCP	10
SCP	11
Control Tables	11
TCT	11
PCT	11
PPT	11
FCT	12
Transaction.....	12
Transaction Life Cycle	13
5. CICS – TRANSACTIONS	16
CESN	16
CEDA	16
CEMT	16
CECI	17
CEDF	18
CMAC	18
CESF.....	18
CEBR	18
CICS Concepts	19
Multitasking	19
Multi-threading.....	19
Re-entrancy.....	19
Quasi-reentrancy	19
6. CICS – COBOL BASICS	20
CICS Program	20
Program Compilation	22
7. CICS – BMS	24
Formatted Screen	24
BMS Basic Terms	25
Map.....	25

Mapset	25
BMS Macros	25
8. CICS – MAP	31
Physical Map	31
Symbolic Map	32
Skipper and Stopper Field	32
Skipper Field	32
Stopper Field	33
Attribute Byte	33
Modified Data Tag	34
Send Map	34
Receive Map	36
Mapset Execution	36
9. CICS – INTERFACE BLOCK	38
Restricted COBOL Verbs	38
Execute Interface Block	38
EIB Fields	39
CICS Programs Classification	39
Non Conversion Programs	39
Conversion Program	40
10. CICS – PSEUDO PROGRAMMING	43
Pseudo-Conversion Program	43
Pseudo Conversion Techniques	44
COMMAREA	44
DFHCOMMAREA	45
Pseudo Code	45
Advantages of Pseudo Conversion	48
Return Statements	48
Return-1	48

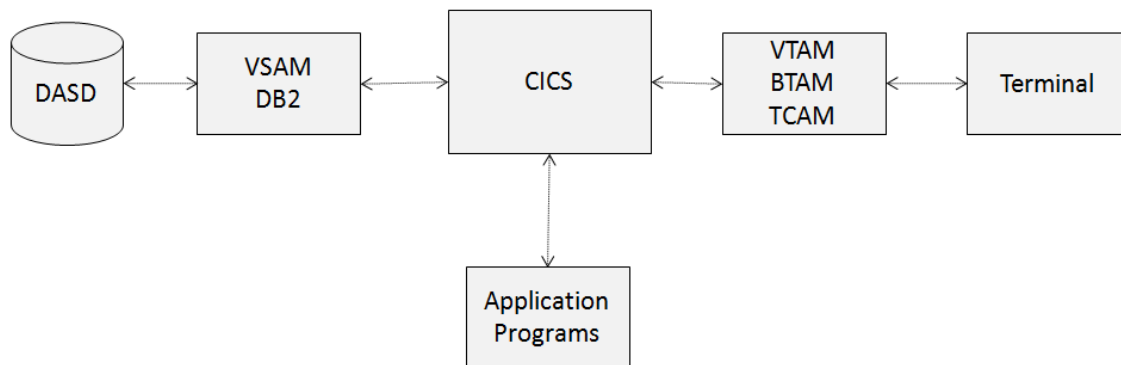
Return-2	48
11. CICS – AID KEYS	49
Validating AID keys	49
DFHAID	49
Cursor Positioning	51
Dynamically Modifying Attributes	51
12. CICS – FILE HANDLING	54
Random Access.....	54
Read	54
Read Command Options	56
Read Command Exceptions	57
Write	57
Write Command Exceptions	58
Rewrite.....	59
Rewrite Command Exceptions.....	60
Delete.....	60
Delete Command Exceptions.....	62
Sequential Access	62
STARTBR.....	62
READNEXT / READPREV	63
RESETBR	63
ENDBR	64
13. CICS – ERROR HANDLING	65
CICS Errors	65
Error Handling Commands.....	65
Handle Condition.....	65
Handle Abend.....	66
Abend	67

Ignore Condition.....	68
Nohandle.....	68
14. CICS – CONTROL OPERATIONS.....	70
Program Logical Levels	70
XCTL	71
Link.....	72
Load	72
Release	73
Return	73
Interval Control Operations	73
ASKTIME.....	73
FORMATIME.....	73
15. CICS – TEMPORARY STORAGE.....	75
COMMAREA	75
Common Work Area	75
Transaction Work Area	75
Temporary Storage Queue	75
WRITEQ TS.....	76
READQ TS	76
DELETEQ TS	77
Transient Data Queue.....	77
WRITEQ TD	77
READQ TD.....	77
DELETEQ TD	78
16. CICS – INTERCOMMUNICATION	79
Benefits of Intercommunication	79

Basic Terminologies	79
Intercommunication Methods	80
17. CICS – STATUS CODES.....	81
18. CICS – INTERVIEW QUESTIONS	82

1. CICS – OVERVIEW

CICS is a DB/DC system which is used in online applications. CICS was developed because batch operating system can execute only batch programs. CICS programs can be written in COBOL, C, C++, Java, etc. These days, users want information within seconds and in real time. To provide such quick service, we need a system which can process information online. CICS allows users to communicate with the back-end system to get the desired information. Examples of online programs include online banking system, flight reservation, etc. Following image shows the components of CICS and how they are inter-related:



Functions of CICS

The main functions performed by CICS in an application are as follows:

- CICS manages requests from concurrent users in an application.
- Although, multiple users are working on CICS system but it gives a feel to user that he is the single user only.
- CICS gives the access to data files for reading or updating them in an application.

Features of CICS

The features of CICS are as follows:

- CICS is an operating system in itself, as it manages its own processor storage, has its own task manager which handles execution of multiple programs, and provides its own file management functions.
- CICS provides online environment in batch operating system. Jobs submitted are executed immediately.
- CICS is a generalized transaction processing interface.
- It is possible to have two or more CICS regions at the same time, as CICS runs as a batch job in the operating system at the back-end.

2. CICS – ENVIRONMENT

CICS itself acts as an operating system. Its job is to provide an environment for online execution of application programs. CICS runs in one region or partition or address space. CICS handles scheduling for programs running under it. CICS runs as a batch job and we can view it in the spool by issuing the command PREFIX CICS*. There are five major services which are provided by CICS. All these services together perform a task.

CICS Environment

Following are the services which we will be discussing in detail step by step:

- System Services
- Data Communication Services
- Data Handling Services
- Application Programming Services
- Monitoring Services
- System Services

CICS maintains control functions to manage the allocation or de-allocation of resources within the system which are as follows:

- **Task Control** – Task control provides task scheduling and multitasking features. It takes care of the status of all CICS tasks. Task Control allocates the processor time among concurrent CICS tasks. This is called **multitasking**. CICS tries to prioritize the response time to the most important task.
- **Program Control** – Program Control manages loading and releasing of application programs. As soon as a task begins, it becomes necessary to associate the task with the appropriate application program. Although many tasks may need to use the same application program, CICS loads only one copy of the code into memory. Each task threads its way through this code independently, so many users can all be running transactions that are concurrently using the same physical copy of an application program.

- **Storage Control** – Storage Control manages acquiring and releasing of main storage. Storage control acquires, controls, and frees dynamic storage. Dynamic storage is used for input/output areas, programs, etc.
- **Interval Control** – Interval Control offers timer services.

Data Communication Services

Data Communication Services interface with telecommunication access methods such as BTAM, VTAM, and TCAM for handling data communication requests from application programs.

- CICS releases application programs from the burden of dealing with terminal hardware issues through the use of Basic Mapping Support (BMS).
- CICS provides Multi Region Operation (MRO) through which more than one CICS region in the same system can communicate.
- CICS provides Inter System Communication (ISC) through which a CICS region in a system can communicate with the CICS region on another system.

Data Handling Services

Data Handling Services interface with data access methods such as BDAM, VSAM, etc.

- CICS facilitates servicing of data handling requests from application programs. CICS provides application programmers a set of commands for dealing with data set and database access and related operations.
- Data Handling Services interfaces with database access methods such as IMS/DB, DB2, etc. and facilitate servicing of database requests from application programs.
- CICS facilitates management of data integrity by control of simultaneous record updates, protection of data as task ABENDs and protection of data at system failures.

Application Programming Services

Application Programming Services interface with application programs. The application programming services of CICS provide features such as command level translation, CEDF (the debug facility) and CECI (the command interpreter facility). We will be discussing more in detail in upcoming modules.

Monitoring Services

Monitoring Services monitor various events within CICS address space. It provides series of statistical information that can be used for system tuning.

3. CICS – BASIC TERMS

We must have knowledge of the basic terms used in CICS to get a better understanding of how it works. Application programs use CICS for communication with remote and local terminals and subsystems.

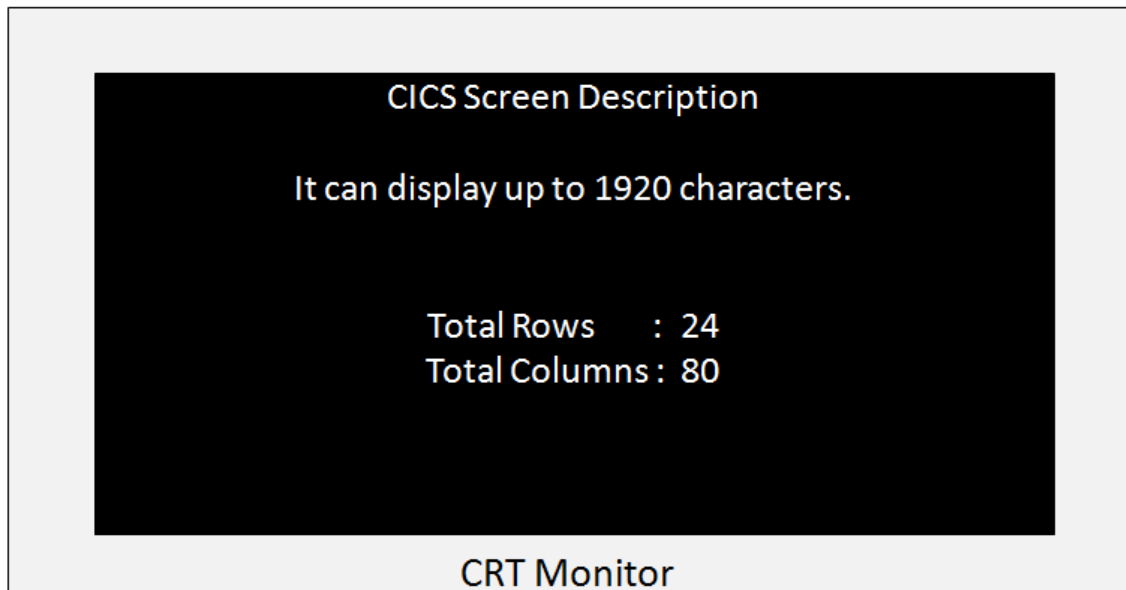
IBM 3270 Terminal

The 3270 Information Display System is a family of display and printer terminals. 3270 terminals were being used to connect to the mainframe via IBM controllers. Today, 3270 emulation software is available which means that even normal PCs can be used as 3270 terminals. 3270 terminals are dumb terminals and do not do any processing themselves. All processing needs to be done by the application program. IBM terminals consist of the following components:

CRT Monitor

The CRT monitor displays the output or the input fields of the application program. A screenshot of a 3278 Model of CRT monitor is shown below. It has the following characteristics:

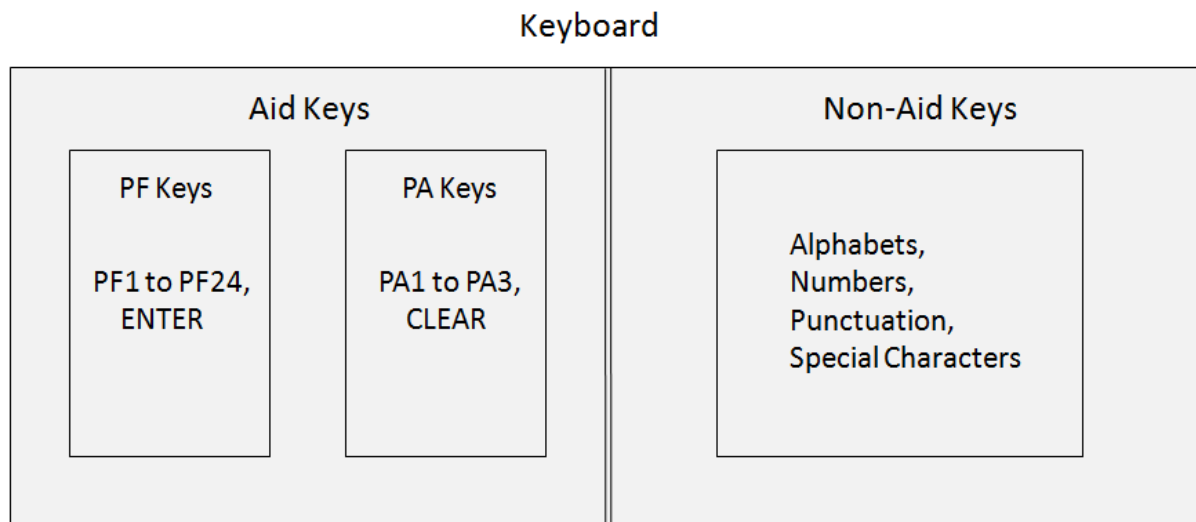
- It is capable of displaying 1920 characters.
- Each of these 1920 character positions is individually addressable.
- A COBOL application program can send data to all the positions on the screen.
- The display characteristics like intensity, protected, non-protected of the field can be set using BMS which we will be discussing in detail in upcoming modules.



Keyboard

IBM keyboard keys are divided into following two categories:

- **Non-AID Keys** – All other keys for alphabets, numeric, punctuation etc. are Non-Aid keys. When the user types text or numbers using non-aid keys, CICS will not even know if the user is typing anything or not.
- **AID Keys** – AID keys are known as Attention Identifier Keys. CICS can detect only AID keys. After typing all the input, only when the user presses one of the AID keys, CICS takes control. AID Keys : ENTER, PF1 to PF24, PA1 to PA3, CLEAR. AID keys are further divided into two categories:
 - **PF Keys** – PF keys are known as function keys. PF keys allow transfer of data from terminal to CICS. PF Keys are ENTER and PF1 to PF24.
 - **PA Keys** – PA keys are known as Program Access keys. PA keys do not allow transfer of data between terminal and CICS. PA Keys are PA1 to PA3 and CLEAR.



Transaction

A CICS program is invoked through a transaction. A CICS transaction is a collection of logically related programs in an application. The whole application could be logically divided into several transactions.

- Transaction identifiers which are 1 to 4 characters long are used to identify the transactions which the users want to do.
- A programmer links one program to the transaction identifier which is used to invoke all the application programs for that particular transaction.

Task

A Task is a unit of work which is specific to a user.

- Users invoke an application by using one of the transaction identifiers. CICS looks up for the transaction identifier to find out which program to invoke first to do the work requested. It creates a task to do the work, and transfers control to the mentioned program.
- A transaction can be completed through several tasks.
- A task can receive data from and send data to the terminal that started it. It can read and write files and can start other tasks also.

Task vs. Transaction

The difference between a transaction and a task is that several users can invoke a transaction but each user initiates his own task.

LUW

LUW stands for Logical Unit of Work. LUW states that a piece of work should be done completely or not done at all. A task can contain several Logical Unit of Works in CICS. We will discuss more about it in upcoming modules.

Application

An application is a series of logically grouped programs to form several transactions which is used to complete a specific task for the end-user.

4. CICS – NUCLEUS

The five CICS system components described earlier are a convenient grouping of CICS system programs, each of which performs its own specialized functions. The core of CICS known as the CICS Nucleus which consists of IBM-supplied CICS Control Programs and Control Tables.

Control Programs

CICS nucleus is constructed by the control programs and corresponding control tables. It provides unique advantages. It makes the CICS system highly flexible and thus easy to maintain. Following are the important control programs of CICS:

TCP

TCP is known as Terminal Control Program.

- TCP is used to receive messages from the terminal.
- It maintains hardware communication requirements.
- It requests CICS to initiate the tasks.

KCP

KCP is known as Task Control Program.

- KCP is used to simultaneously control the execution of tasks and its related properties.
- It handles all the issues related to multi-tasking.

PCP

PCP is known as Program Control Program.

- PCP is used to locate and load programs for execution.
- It transfers the control between programs and in the end, it returns the control back to the CICS.

FCP

FCP is known as File Control Program.

- FCP is used to provide application programs with services like read, insert, update or delete records in a file.
- It keeps exclusive control over the records in order to maintain data integrity during record updates.

SCP

SCP is known as Storage Control Program. It is used to control allocation and de-allocation of storage within a CICS region.

Control Tables

CICS consists of IBM-supplied CICS control programs and tables. These tables need to be updated accordingly with the application information for successful execution of CICS application programs. Following are the important Control Tables:

TCT

TCT is known as Terminal Control Table.

- When we login to a CICS terminal, an entry is made in the TCT table.
- TCT contains the terminal ID's that are connected to current CICS region.
- Terminal Control Program along with terminal control table recognize the incoming data from the terminal.

PCT

PCT is known as Program Control Table.

- It contains the Transaction IDs (TRANSID) and the corresponding program names or program IDs.
- TRANSID is unique in PCT table.

PPT

PPT is known as Processing Program Table. PPT contains Program name or Mapset name, Task Use Counter, Language, Size, Main storage address, Load library address, etc.

- Program or Mapset name is unique in a PPT table.
- CICS receives the transaction and a corresponding program name is allocated to the transaction from the PCT. It checks if the program is loaded or not. If it is loaded, then the task use counter is increased by 1. If the

program is not loaded, then the program is first loaded and the task use counter is set to 1. It gets the load library address from the PPT table.

FCT

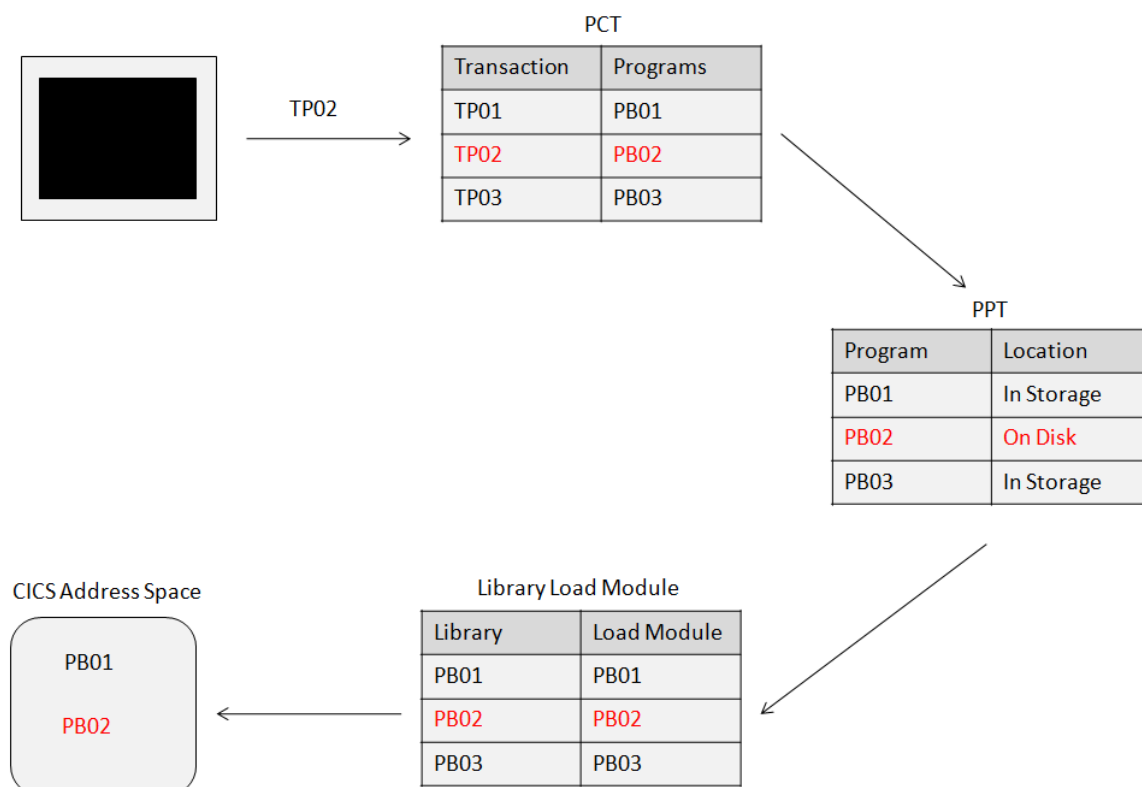
FCT is known as File Control Table.

- It contains File names, File type, record length, etc.
- All the files used in a CICS program must be declared in FCT and they are opened and closed by CICS itself.

Transaction

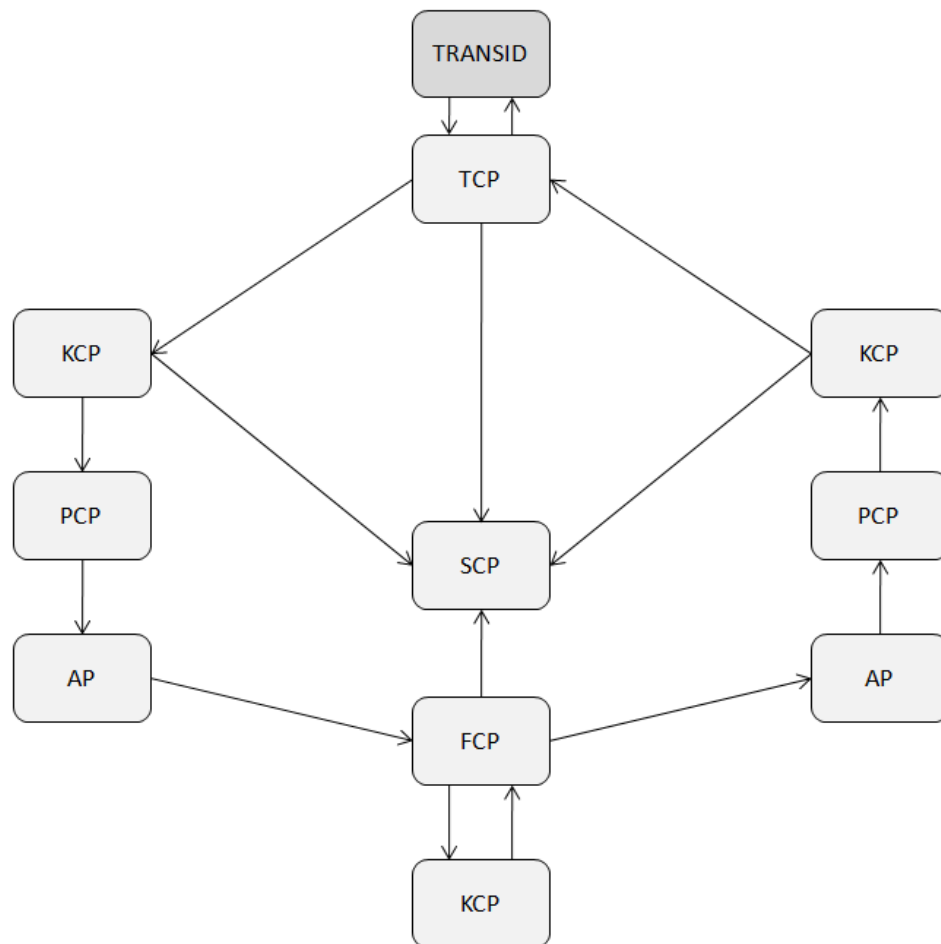
When a transaction identifier TP02 is entered on the CICS terminal, first it checks if there is a program associated with this Transaction identifier in the PCT table. If it finds one, then it checks in the PPT table to find the location of the Program to execute it.

If the program is already available in the memory, it starts executing that particular program; if not, it loads the program to the memory from the secondary storage and then starts executing it.



Transaction Life Cycle

The transaction life cycle has the following steps:



Step 1

The terminal operator initiates the transaction by typing a 1 to 4 character transaction-id and pressing the ENTER key.

Step 2

The TCP periodically checks all the terminals for input. When a message is received, it does the following:

- Instructs the SCP to create a TIOA.
- Places the message in the TIOA.
- Passes the control to the KCP.

Step 3

The KCP takes control from the TCP and does the following:

- Validates the transaction-id and security.
- Instructs the SCP to create a task control area.
- Assigns priority to the task based on Terminal priority (Set in TCT), Operator priority (Set in SNT), and Transaction priority (Set in PCT).
- Adds the task to the queue of waiting programs.
- Dispatches waiting programs in the order of priority.
- Passes the control to the PCP.

Step 4

The PCP takes control from the KCP and does the following:

- Locates the program and loads it, if necessary.
- Transfers the control to the Application program.

Step 5

The Application program takes control from the PCP and does the following:

- Requests the TCP to place the message into the program's WORKING STORAGE area.
- Requests the FCP to retrieve records from the files.

Step 6

The FCP takes control from the Application program and does the following:

- Requests a File work area from the SCP.
- Informs the KCP that this task can wait until the I/O is complete.

Step 7

The KCP does the following:

- Dispatches the next task in the queue.
- Re-dispatches the old task when I/O is complete.
- Transfers the control to the FCP.

Step 8

The FCP returns control to the Application program.

Step 9

The Application program does the following:

- Processes the file data.

- Requests TCP to send an I/O message.
- Returns control to the PCP.

Step 10

The PCP returns the control back to the KCP requesting it to end the task.

Step 11

The KCP instructs the SCP to free all the storage allocated to the task (except TIOA).

Step 12

The TCP does the following:

- Sends the output to the terminal.
- Requests the SCP to release the TIOA.

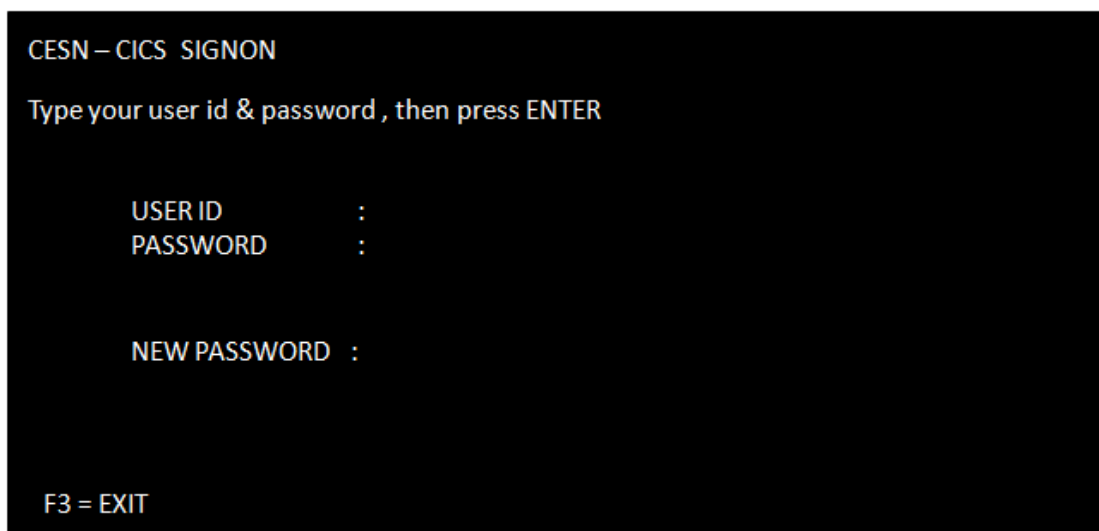
5. CICS – TRANSACTIONS

CICS transactions are used to perform multiple operations in the CICS region. We will be discussing the important CICS transactions supplied by IBM in detail.

CESN

CESN is known as CICS Execute Sign On.

- CESN is used to Sign on to the CICS region.
- We need to provide the User-Id and Password given by the CICS administrator to log on to CICS. The following screenshot shows how the sign-on screen looks like:



```
CESN - CICS SIGNON
Type your user id & password , then press ENTER

USER ID      :
PASSWORD     :

NEW PASSWORD :

F3 = EXIT
```

CEDA

CEDA is known as CICS Execute Definition and Administration. It is used by CICS System Administrators to define CICS table entries and other administration activities.

CEMT

CEMT is known as CICS Execute Master Terminal. It is used to inquire and update the status of CICS environments and also for other system operations.

- Using CEMT command, we can manage transactions, tasks, files, programs, etc.
- To get all the possible options, type CEMT and press ENTER. It will display all the options.
- CEMT is basically used for loading a new program into the CICS or for loading a new copy of the program into the CICS after the program or mapset is changed.

Example

One can overwrite the status of the file displayed to change it. Following example shows how to close a file:

```
CEMT

** Press ENTER & Following Screen is displayed **

STATUS: ENTER ONE OF THE FOLLOWING
Inquire
Perform
Set

** Command to close a file **

CEMT SET FILE (file-name)
CEMT I FILE (file-name)
```

CECI

CECI is known as CICS Execute Command Interpreter. Many CICS commands can be executed using CECI.

- CECI is used to check the syntax of the command. It executes the command, only if the syntax is correct.
- Type the CECI option on the empty CICS screen after having logged in. It gives you the list of options available.

Example

Following example shows how to send mapped output data to terminal. We will be discussing about MAPS in the upcoming modules.

```
CECI SEND MAP (map-name) MAPSET (mapset-name) ERASE
```

CEDF

CEDF is known as CICS Execute Debug Facility. It is used for debugging the program step by step, which helps in finding the errors.

Type CEDF and press enter in the CICS region. The terminal is in EDF mode message will be displayed. Now type the transaction id and press the enter key. After initiation, with each enter key, a line is executed. Before executing any CICS command, it shows the screen in which we can modify the values before proceeding further.

CMAC

CMAC is known as CICS Messages for Abend Codes. It is used to find the explanation and reasons for CICS Abend Codes.

Example

Following example shows how to check details for an Abend code:

```
CMAC abend-code
```

CESF

CESF is known as CICS Execute Sign Off. It is used to Sign Off from the CICS region.

Example

Following example shows how to log off from the CICS region:

```
CESF LOGOFF
```

CEBR

CEBR is known as CICS Execute Temporary storage Browse. It is used to display contents of a temporary storage queue or TSQ.

CEBR is used while debugging to check if the items of the queue are being written and retrieved properly. We will discuss more about TSQ in the upcoming modules.

Example

Following example shows how to invoke the CEBR command:

```
CEBR queue-id
```

CICS Concepts

Each command could be achieved by executing a series of CICS macros. We will discuss some basic features which will help us understand the concepts better:

Multitasking

This feature of operating system allows more than one task to be executed concurrently. The task may be sharing the same program or using different programs. The CICS schedules the task in its own region.

Multi-threading

This feature of the operating system allows more than one task to be executed concurrently sharing the same program. For multi-threading to be possible, an application program should be a **re-entrant program** under the operating system or a **quasi-reentrant** under the CICS.

Re-entrancy

A re-entrant program is one which does not modify itself and can re-enter in itself and continue processing after an interruption by the operating system.

Quasi-reentrancy

A quasi-reentrant program is a re-entrant program under CICS environment. CICS ensures re-entrancy by acquiring a unique storage area for each task. Between CICS commands, the CICS has the exclusive right to use the CPU resources and it can execute other CICS commands of other tasks.

There are times when many users are concurrently using the same program; this is what we call **multi-threading**. For example, let's suppose 50 users are using a program A. Here the CICS will provide 50 working storage for that program but one Procedure Division. And this technique is known as **quasi-reentrancy**.

6. CICS – COBOL BASICS

CICS programs are written in COBOL language in Mainframes. We will be discussing about writing a simple COBOL-CICS program, compiling it, and then executing it.

CICS Program

We will be writing a simple COBOL-CICS program which displays some message on the CICS output screen. This program is to demonstrate the steps involved in executing a COBOL-CICS program. Following are the steps to code a simple program:

Step 1

Login to Mainframes and open a TSO Session.

Step 2

Create a new PDS in which we will be coding our program.

Step 3

Create a new member inside the PDS and code the following program:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. HELLO.  
DATA DIVISION.  
FILE SECTION.  
WORKING-STORAGE SECTION.  
01 WS-MESSAGE PIC X(40).  
01 WS-LENGTH PIC S9(4) COMP.  
PROCEDURE DIVISION.  
A000-MAIN-PARA.  
    MOVE 'Hello World' TO WS-MESSAGE  
    MOVE '+12' TO WS-LENGTH  
    EXEC CICS SEND TEXT  
        FROM (WS-MESSAGE)  
        LENGHT(WS-LENGTH)
```

```
END-EXEC  
EXEC CICS RETURN  
END-EXEC.
```

Step 4

After coding the program, we need to compile it. We can compile the program using the following JCL:

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C  
//CICSCOB EXEC CICSCOB,  
//          COPYLIB=ABC.XYZ.COPYLIB,  
//          LOADLIB=ABC.XYZ.LOADLIB  
//LIB       JCLLIB ORDER=CICSXXX.CICS.XXXPROC  
//CPLSTP    EXEC DFHEITVL  
//TRN.SYSIN DD DSN=ABC.XYZ.PDS(HELLO),DISP=SHR  
//LKED.SYSIN DD *  
            NAME HELLO(R)  
//
```

Step 5

Open a CICS session.

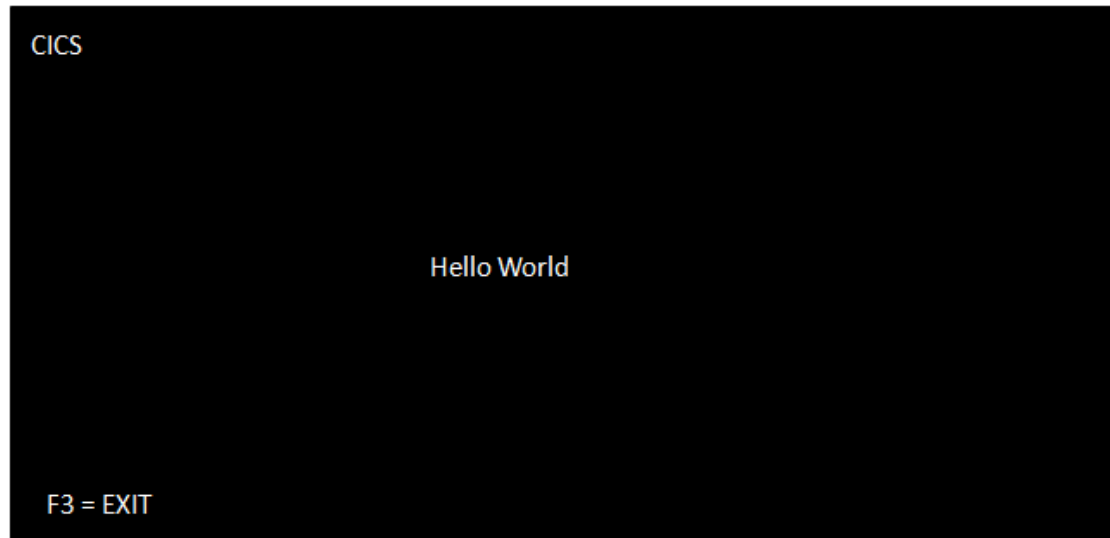
Step 6

We will now install the program using the following command:

```
CEMT SET PROG(HELLO) NEW.
```

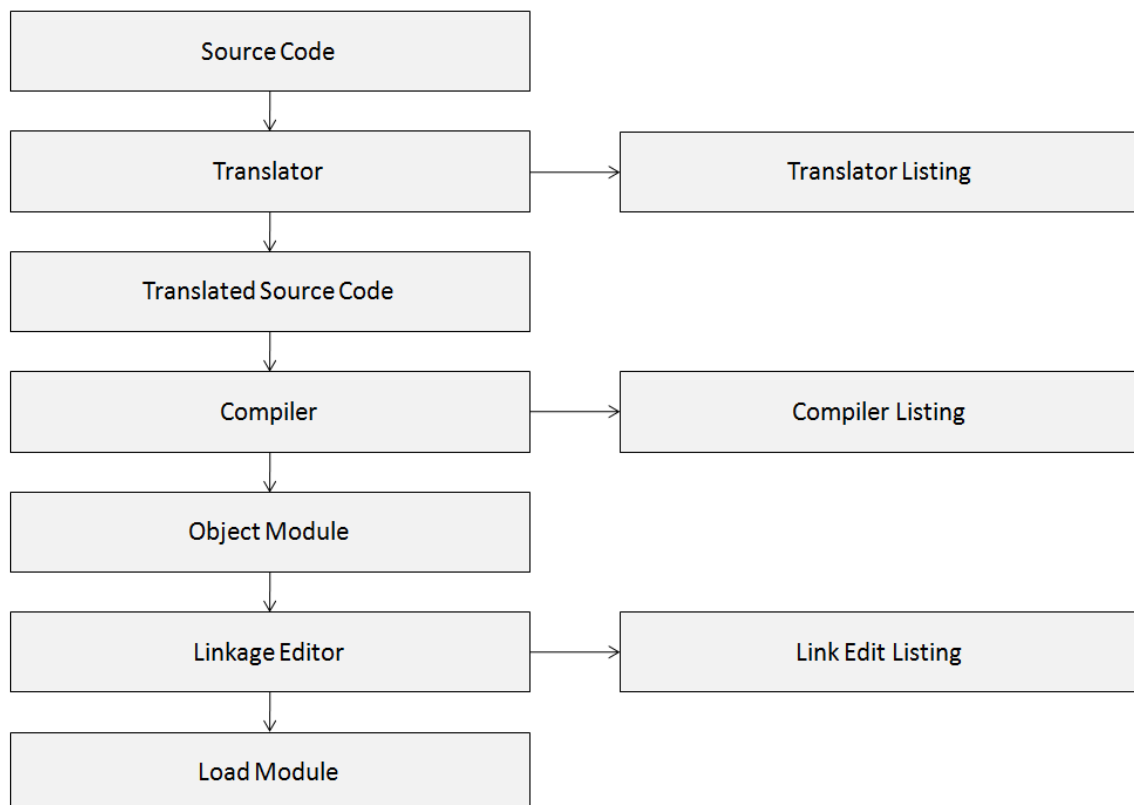
Step 7

Execute the program using the associated transaction-id. Transaction-id is provided by the Administrator. It will show the following output:



Program Compilation

The following flowchart shows the steps used in compiling a COBOL-CICS program:



Translator

The function of a translator is to check for syntax errors in CICS commands. It translates them into equivalent COBOL statements.

Compiler

The function of a compiler is to expand the COBOL copy books. It compiles the code after checking the source code for syntax errors.

Linkage Editor

The function of a Linkage Editor is to link different object modules to create a single load module.

7. CICS – BMS

BMS is known as Basic Mapping Support. An application consists of formatted screens which act as a bridge between the terminal and the CICS programs. For communication to occur between the terminal and the CICS programs, we use CICS terminal input/output services. We use BMS to create screen designs with proper positions and attributes. Following are the functions of BMS:

- BMS acts as an interface between the terminal and the CICS programs.
- The design and format of the screen is separate from the logic of application.
- BMS makes the application hardware independent.

Formatted Screen

The screen shown below is a Menu Screen and can be designed using BMS. Its key points are as follows:

- The screen could have a Title, date, and any other information that is to be displayed.
- The Option 1, 2, and 3 are the Unnamed fields which are the titles of the screen.
- In the Selection field, we need to provide the input. This input is then sent to the CICS program for further processing.
- At the bottom of the screen, Action keys are displayed.
- All the fields and the screen itself is defined with BMS macros. When the whole map is defined, we can use JCL to assemble it.

```
SCREEN-1                                MAIN MENU                                DATE: 11/13/2014

Type your user id & password , then press ENTER

1. BOOK FLIGHT
2. BOOK HOTEL
3. EXIT

SELECTION (1/2/3) : ____

F1 = HELP      F3 = EXIT      ENTER = Process
```

BMS Basic Terms

Following are the basic terms which we will be using in the upcoming modules:

Map

Map is a single screen format which can be designed using BMS macros. It can have names containing 1 to 7 chars.

Mapset

Mapset is a collection of maps which are linked together to form a load module. It should have a PPT entry. It can have names from 1 to 7 chars.

BMS Macros

BMS map is a program which is written in Assembly language to manage screens. The three macros that are used to define the screen are DFHMSD, DFHMDI, and DFHMDF.

DFHMSD

DFHMSD macro generates Mapset definition. It is macro identifier which shows that we are starting a mapset. The mapset name is the load module name and an entry in PPT table must be present. The following table shows the list of parameters which can be used in DFHMSD:

Parameter	Description
TYPE	TYPE is used to define the map type. If TYPE= MAP - Physical map is created DSECT - Symbolic map is created &&SYSPARM - Physical & Symbolic, both are created FINAL - To indicate the end of a mapset coding
MODE	MODE is used to indicate input/output operations. IF MODE= IN - For an input map only OUT - For an output map only INOUT For both input & output map
LANG	LANG=ASM/COBOL/PL1 It decides the language of the DSECT structure, for copying into the application program.
STORAGE	If STORAGE= AUTO - To acquire a separate symbolic map area for each mapset BASE - To have the same storage base for the symbolic maps of from more than one mapset
CTRL	CRTL is used to define the device control requests. If CTRL= FREEKB - To unlock the keyboard FRSET - To reset MDT to zero status ALARM - To set an alarm at screen display time PRINT - To indicate the mapset to be sent to the printer
TERM	TERM=type ensures device independence,required if other than 3270 terminal is being used
TIOAPFX	TIOAPFX=YES/NO YES - To reserve the prefix space (12 bytes) for BMS commands to access TIOA properly. Required for the CICS command level.

Example

The following example shows how to code a mapset definition:

MPST01	DFHMSD TYPE=&SYSPARM,	X
	CTRL=(FREEKB,FRSET),	X
	LANG=COBOL,	X
	STORAGE=AUTO,	X
	TIOAPFX=YES,	X
	MODE=INOUT,	X
	TERM=3270	
	DFHMSD TYPE=FINAL	
	END	

DFHMDI

DFHMDI macro generates map definitions. It shows that we are starting a new map. Mapname is followed by the DFHMDI macro. Mapname is used to send or receive maps. The following table shows the parameters which we use inside a DFHMDI macro:

Parameter	Description
SIZE	SIZE=(Line,Column) This parameter gives the size of the map. BMS allows us to build a screen using several maps, and this parameter becomes important when we are using more than one maps in a single mapset.
LINE	It indicates the starting line number of the map.
COLUMN	It indicates the starting column number of the map.
JUSTIFY	It is used to specify the entire map or the map fields to be left or right justified.
CTRL	CRTL is used to define the device control requests. If CTRL= FREEKB - To unlock the keyboard FRSET - To reset MDT to zero status

	ALARM - To set an alarm at screen display time PRINT - To indicate the map to be sent to the printer
TIOAPFX	TIOAPFX=YES/NO YES - To reserve the prefix space (12 bytes) for BMS commands to access TIOA properly. Required for the CICS command level.

Example

The following example shows how to code a map definition:

MAPSTD	DFHMDI	SIZE=(20,80),	X
		LINE=01,	X
		COLUMN=01,	X
		CTRL=(FREEKB,FRSET)	

DFHMDF

DFHMDF macro is used to define field names. The field name is mentioned against which DFHMDF macro is coded. This field name is used inside the program. We do not write field name against constant field which we do not want to use inside the program. The following table shows the list of parameters which can be used inside a DFHMDF macro:

Parameter	Description
POS	This is the position on the screen where the field should appear. A field starts with its attribute byte, so if you code POS=(1,1), the attribute byte for that field is on line 1 in column 1, and the actual data starts in column 2.
LENGTH	This is the length of the field, not counting the attribute byte.
INITIAL	This is the character data for an output field. We use this to specify labels and titles for the screen and keep them independent of the program. For the first field in the menu screen, for example, we will code: INITIAL='MENU'.
JUSTIFY	It is used to specify the entire map or the map fields to be left or right justified.

ATTRB	<p>ATTRB=(ASKIP/PROT/UNPROT, NUM, BRT/NORM/DRK, IC, FSET) It describes the attributes of the field.</p> <p>ASKIP - Autoskip. Data cannot be entered in this field. The cursor skips to the next field.</p> <p>PROT - Protected field. Data cannot be entered into this field. If data is entered, it will cause the input-inhibit status.</p> <p>UNPROT - Unprotected field. Data can be entered and this is used for all input fields.</p> <p>NUM - Numeric field. Only numbers (0 to 9) and special characters('.') and '-') are allowed.</p> <p>BRT - Bright display of a field (highlight).</p> <p>NORM - Normal display.</p> <p>DRK - Dark display.</p> <p>IC - Insert cursor. The cursor will be positioned in this field. In case, IC is specified more than once, the cursor is placed in the last field.</p> <p>FSET - Field set. MDT is set on so that the field data is to be sent from the terminal to the host computer regardless of whether the field is actually modified by the user.</p>
PICIN	PICIN applies to the data field which is used as input like PICIN = 9(8).
PICOUT	PICIN applies to the data field which is used as output like PICOUT = Z(8).

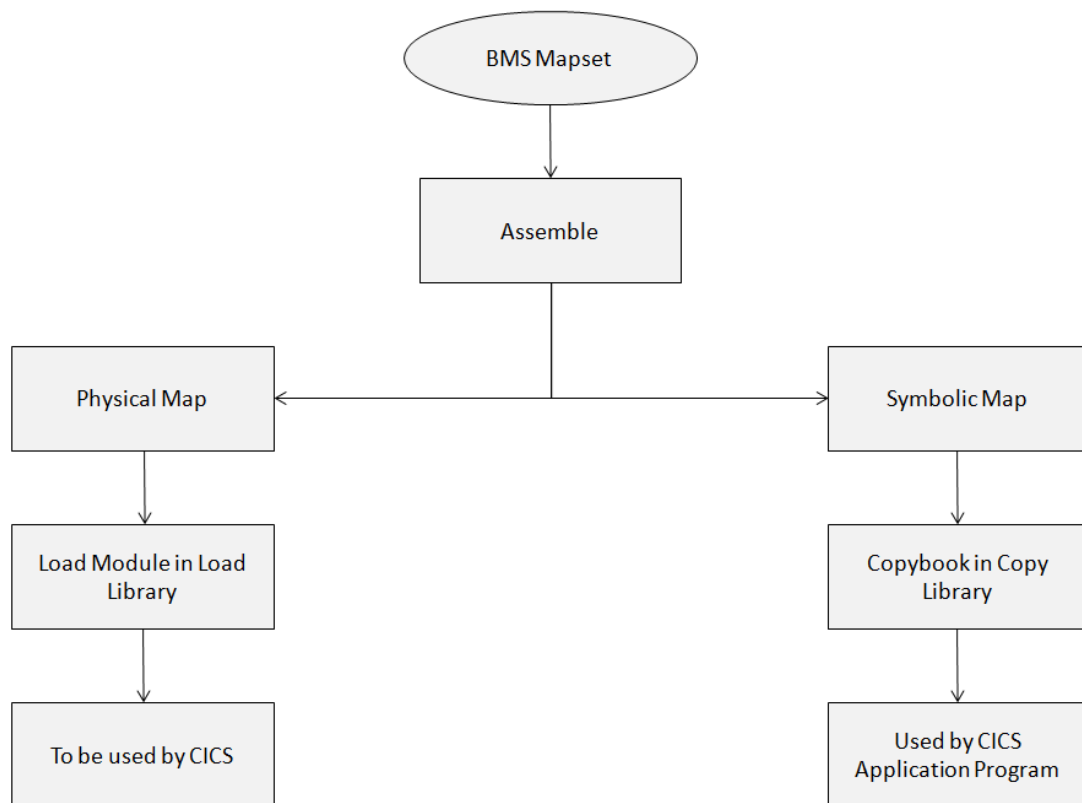
Example

The following example shows how to code a field definition:

	DFHMDF	POS=(01,01),	X
		LENGTH=7,	X
		INITIAL='SCREEN1',	X
		ATTRB=(PROT,NORM)	
STDID	DFHMDF	POS=(01,70),	X
		LENGTH=08,	X
		ATTRB=(PROT,NORM)	

8. CICS – MAP

BMS receives the data entered by the user and then formats it into a symbolic map area. The application program has access only to the data present in the symbolic map. The application program processes the data and the output is sent to the symbolic map. BMS will merge the output of the symbolic data with the physical map.



Physical Map

Physical Map is a load module in the load library which contains information about how the map should be displayed.

- It contains the details about the attributes of all the fields in the map and their positions.
- It contains the display format of the map for a given terminal.
- It is coded using BMS macros. It is assembled separately and link edited into the CICS library.

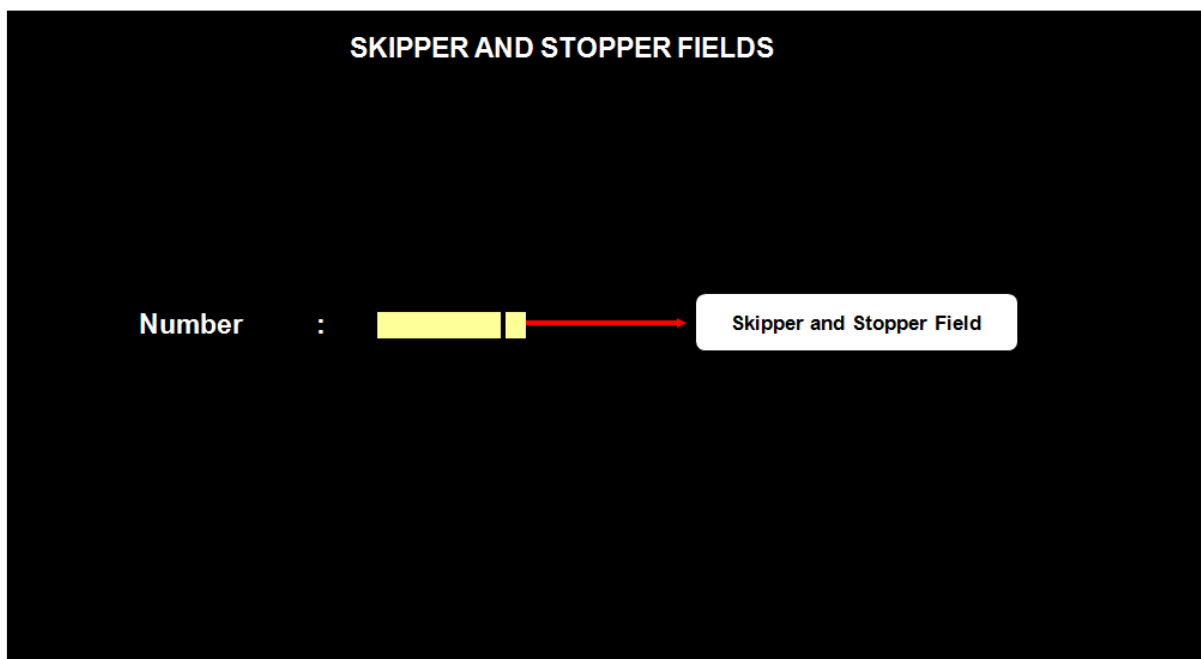
Symbolic Map

A Symbolic Map is a Copy book in the library. The Copy book is used by the CICS application program to send and receive data from the terminal.

- It contains all the variable data which is copied into program's WORKING-STORAGE section.
- It has all the named fields. The application programmer uses these fields to read and write data into the map.

Skipper and Stopper Field

For an unprotected named field, in a map, if we have specified a length of 10, this means that the name field can take values whose length cannot exceed 10. But when you display this map using CICS and start entering values for this field on the screen, we can enter more than 10 Characters, i.e., till the end of the screen and we can enter even in the next line. To prevent this, we use Skipper field or stopper field. A Skipper field would generally be an Unnamed field of length 1, specified after a named field.



Skipper Field

If we place a skipper field after the named unprotected field, then while entering the value, once the specified length is reached, the cursor will automatically position to the next unprotected field. The following example shows how to add a skipper field:

NUMBER	DFHMD	POS=(01,01),	X
		LENGTH=5,	X
		ATTRB=(UNPROT,IC)	
	DFHMD	POS=(01,07),	X
		LENGTH=1,	X
		ATTRB=(ASKIP)	

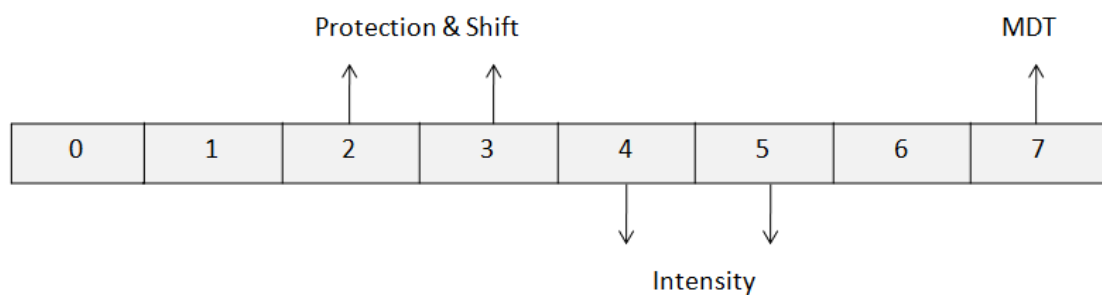
Stopper Field

If we place a stopper field after the named unprotected field, then while entering the value, once the specified length is reached, the cursor will stop its positioning. The following example shows how to add a stopper field:

NUMBER	DFHMD	POS=(01,01),	X
		LENGTH=5,	X
		ATTRB=(UNPROT,IC)	
	DFHMD	POS=(01,07),	X
		LENGTH=1,	X
		ATTRB=(PROT)	

Attribute Byte

The attribute byte of any field stores information about the physical properties of the field. The following diagram and the table explain the significance of each bit.



Bit Position	Description	Bit Settings
0 & 1		Determined by contents of bit 2 to 7
2 & 3	Protection & Shift	00 - Unprotected Alphanumeric 01 - Unprotected Numeric 10 - Protected Stop 11 - Protected Skip
4 & 5	Intensity	00 - Normal 01 - Normal 10 - Bright 11 - No-display (Dark)
6		Must be Zero Always
7	Modified Data Tag	0 - Field has not been modified 1 - Field has been modified

Modified Data Tag

Modified Data Tag (MDT) is the last bit in the attribute byte.

- MDT is a flag which holds a single bit. It specifies whether the value is to be transferred to the system or not.
- Its default value is 1, when the field value is changed.
- If MDT is 0, then data cannot be transferred; and if MDT is 1, then data can be transferred.

Send Map

The send map command writes formatted output to the terminal. It is used to send the map to the terminal from the application program. The following code segment shows how to send a map to the terminal:

```
EXEC CICS SEND
  MAP('map-name')
  MAPSET('mapset-name')
  [FROM(data-area)]
```

```
[LENGTH(data_value)]  
[DATAONLY]  
[MAPONLY]  
[CURSOR]  
[ERASE/ERASEAUP]  
[FREEKB]  
[FRSET]  
END-EXEC
```

The following table lists the parameters used in a send map command along with their significance.

Parameter	Description
Map-name	It is the name of the map which we want to send. It is mandatory.
Mapset-name	It is the name of the map set that contains the mapname. The mapset name is needed unless it is the same as the map name.
FROM	It is used if we have decided to use a different DSECT name, we must use the option FROM (dsect-name) along with SEND MAP command.
MAPONLY	It means that no data from your program is to be merged into the map and only the information in the map is transmitted.
DATAONLY	It is the logical opposite of MAPONLY. We use it to modify the variable data in a display that has already been created. Only the data from your program is sent to the screen. The constants in the map are not sent.
ERASE	It causes the entire screen to be erased before what we are sending is shown.
ERASEUP	It causes only unprotected fields to be erased.

FRSET	Flag Reset turns off the modified data tag in the attribute byte for all the fields on the screen before what you are sending is placed there.
CURSOR	It can be used to position the cursor on the terminal screen. Cursor can be set by moving -1 to the L part of the field and then sending the map.
ALARM	It causes the audible alarm to be sounded.
FREEKB	The keyboard is unlocked if we specify FREEKB in either the map or the SEND command.
PRINT	It allows the output of a SEND command to be printed on a printer.
FORMFEED	It causes the printer to restore the paper to the top of the next page before the output is printed.

Receive Map

When we want to receive input from a terminal, we use the RECEIVE MAP command. The MAP and MAPSET parameters have exactly the same meaning as for the SEND MAP command. The following code segment shows how to receive a map:

```
EXEC CICS RECEIVE
    MAP('map-name')
    MAPSET('mapset-name')
    [INTO(data-area)]
    [FROM(data-area)]
    [LENGTH(data_value)]
END-EXEC
```

Mapset Execution

The following steps are necessary to develop and execute a mapset:

- Step 1 - Open a TSO session.

- Step 2 - Create a new PDS.
- Step 3 - Code a mapset in a new member according to the requirement.
- Step 4 - Assemble the mapset using the JCL provided by the CICS administrator.
- Step 5 - Open a CICS Session.
- Step 6 - Install the program using the command:
CEMT SET PROG(mapset-name) NEW
- Step 7 - Type the following command to send the Map to the terminal:
CECI SEND MAP(map-name) MAPSET(mapset-name) ERASE FREEKB

9. CICS – INTERFACE BLOCK

Any application program would require an interface to interact with the CICS. EIB (Execute Interface Block) acts as an interface to allow application programs communicate with the CICS. EIB contains the information required during the execution of a program.

Restricted COBOL Verbs

While coding a CICS program, we cannot use the commands which return the control directly to the MVS. If we code these COBOL verbs, it will not give any compilation error, but we may get unpredictable results. Following are the COBOL verbs which should not be used in a CICS program:

- File I/O statements like Open, Read, Write, Rewrite, Close, Delete, and Start. All file I/O in CICS is handled by the file control module and they have their own set of statements like READ, WRITE, REWRITE, and DELETE which we will be discussing in the upcoming modules.
- File Section and Environment Division is not required.
- COBOL statements that invoke operating system functions like Accept, Date/Time cannot be used.
- Do not use DISPLAY, MERGE, STOP RUN, and GO BACK.

Execute Interface Block

Execute Interface Block (EIB) is a control block which is loaded automatically by the CICS for every program.

- The EIB is unique to a task and it exists for the duration of the task. It contains a set of system related information corresponding to the task.
- It contains information about transaction identifier, time, date, etc., which is used by the CICS during the execution of an application program.
- Every program that executes as a part of the task has access to the same EIB.
- The data in EIB at runtime can be viewed by executing the program in CEDF mode.

EIB Fields

The following table provides a list of fields which are present in EIB:

EIB Field	PIC Clause	Description
EIBAID	X(1)	Aid key Pressed
EIBCALEN	S9(4) COMP	It contains length of DFHCOMMAREA
EIBDATE	S9(7) COMP-3	It contains Current System Date
EIBRCODE	X(6)	It contains Return code of the last transaction
EIBTASKN	S9(7) COMP-3	It contains Task number
EIBTIME	S9(7) COMP-3	It contains Current System Time
EIBTRMID	X(4)	Terminal Identifier
EIBTRNID	X(4)	Transaction Identifier

CICS Programs Classification

CICS Programs are classified in the following three categories which we will discuss one by one:

- Non-Conversion Programs
- Conversion Programs
- Pseudo-conversion Programs – We will discuss in the next module

Non Conversion Programs

While executing non-conversion programs, no human intervention is required. All the necessary inputs are provided when the program is started.

- They are similar to batch programs that run in the batch mode. So in CICS, they are rarely developed.
- We can say they are used just for displaying a sequence of screens at regular intervals of time.

Example

The following example shows a non-conversion program which will simply display "HELLO WORLD" on the CICS terminal as output:

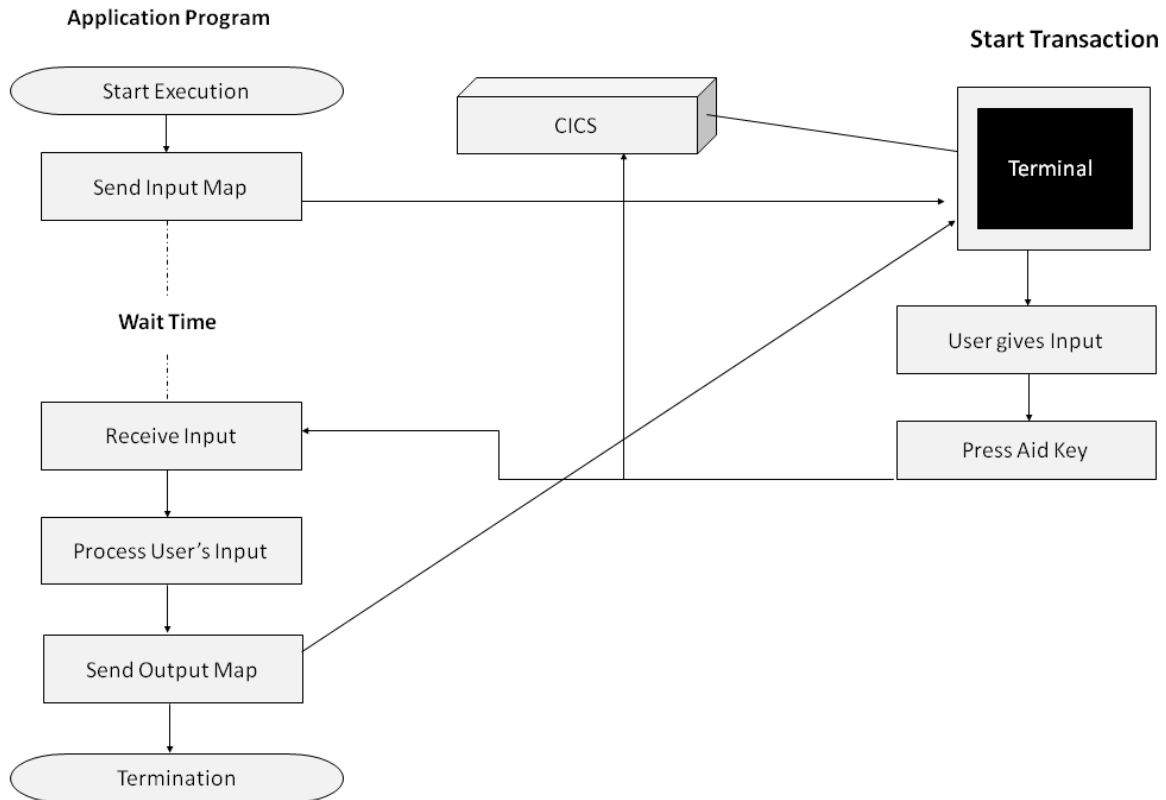
```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-MESSAGE          PIC X(30).
PROCEDURE DIVISION.
*****
*  SENDING DATA TO SCREEN                                *
*****
      MOVE 'HELLO WORLD' TO WS-MESSAGE
      EXEC CICS SEND TEXT
          FROM (WS-MESSAGE)
      END-EXEC
*****
*  TASK TERMINATES WITHOUT ANY INTERACTION FROM THE USER*
*****
      EXEC CICS RETURN
      END-EXEC.
```

Conversion Program

Sending a message to the terminal and receiving a response from the user is called a **conversation**. An online application achieves a conversation between the user and the application program by a pair of SEND and RECEIVE command. The key points of a conversion program are as follows:

- The system sends a message to the screen and waits for the user's response.
- The time taken by user to respond is known as **Think Time**. This time is considerably high, which is a major drawback of conversion programs.
- The user provides the necessary input and presses an AID key.
- The application processes the user's input and sends the output.

- The program is loaded into the main storage at the beginning and is retained till the task ends.



Example

The following example shows a conversion program which takes input from the user and then simply displays the same input on the CICS terminal as output:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-MESSAGE          PIC X(30) VALUE SPACES.
PROCEDURE DIVISION.
    MOVE 'ENTER MESSAGE' TO WS-MESSAGE
*****
* SENDING DATA FROM PROGRAM TO SCREEN          *
*****
EXEC CICS SEND TEXT
```

```
FROM (WS-MESSAGE)
END-EXEC
```

```
*****
```

```
* GETTING INPUT FROM USER *
```

```
*****
```

```
EXEC CICS RECEIVE
      INTO(WS-MESSAGE)
END-EXEC
EXEC CICS SEND TEXT
      FROM (WS-MESSAGE)
END-EXEC
```

```
*****
```

```
* COMMAND TO TERMINATE THE TRANSACTION *
```

```
*****
```

```
EXEC CICS RETURN
END-EXEC.
```

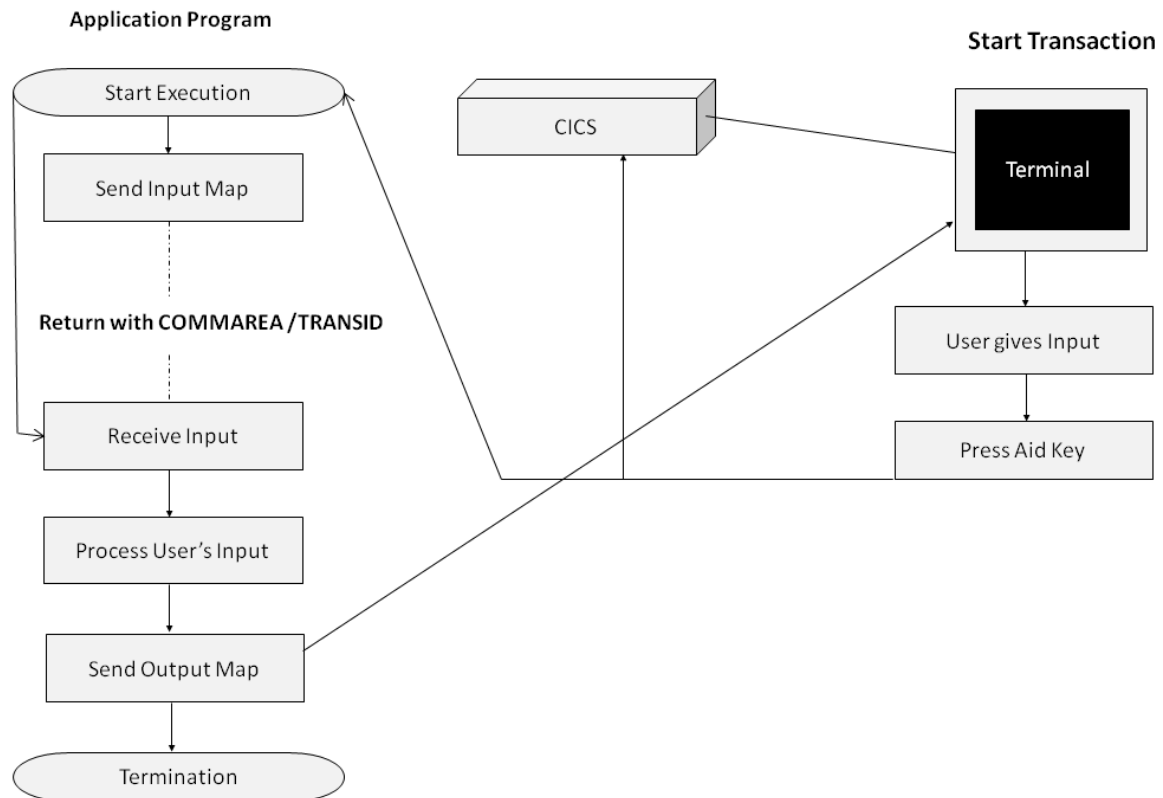
10. CICS – PSEUDO PROGRAMMING

As of now, we have covered non-conversion and conversion programs. Conversion programs have a major drawback as their **think time** is considerably high. To overcome this problem, pseudo-conversion programming came into the picture. We will now discuss more about pseudo-conversion programs.

Pseudo-Conversion Program

Following is the sequence of events which take place in a pseudo-conversion program:

1. The system sends a message to the screen and terminates the transaction, specifying the transaction to be started when the user input is received.
2. The system allocates the resources used by this transaction to other transactions running in the system. So we can utilize the resources in a pseudo-conversion program till the user gives the input.
3. The system polls the terminal input at regular intervals of time. When the input is received, it is processed and the output is displayed.
4. The application program is loaded into the main storage when needed and released when not in use.



Pseudo Conversion Techniques

The important point to note in pseudo-conversation is passing of data between every task. We will discuss about the techniques for passing data.

COMMAREA

COMMAREA is known as communication area. COMMAREA is used to pass data between tasks. The following example shows how to pass COMMAREA where WS-COMMAREA and WS-COMMAREA-LENGTH are declared in Working Storage Section:

```
EXEC CICS RETURN
    TRANSID ('transaction-id')
    COMMAREA (WS-COMMAREA)
    LENGTH  (WS-COMMAREA-LENGTH)
END-EXEC.
```

DFHCOMMAREA

DFHCOMMAREA is a special memory area which is provided by CICS to every task.

- It is used to pass data from one program to another program. The programs can exist in the same transaction or in different transaction also.
- It is declared in the Linkage Section of the program at 01 level.
- It should have the same picture clause as WS-COMMAREA.
- Data can be moved back from DFHCOMMAREA to WS-COMMAREA using a MOVE statement.

```
MOVE DFHCOMMAREA TO WS-COMMAREA.
```

Example

After sending the map, the task ends and waits for the user response. At this stage, the data needs to be saved, because though the task has ended, the transaction has not. When this transaction is to be resumed, it would require the prior status of the task. User enters the input. This now has to be received by the RECEIVE MAP command and then validated. The following example shows how to declare COMMAREA and DFHCOMMAREA:

```
WORKING-STORAGE SECTION.
```

```
01 WS-COMMAREA.
```

```
05 WS-DATA PIC X(10).
```

```
LINKAGE SECTION.
```

```
01 DFHCOMMAREA.
```

```
05 LK-DATA PIC X(10).
```

Pseudo Code

Given below is the logic of pseudo code which we use in pseudo programming:

```
MOVE DFHCOMMAREA TO WS-COMMAREA
```

```
IF EIBCALEN = 0
```

```
    STEP1: SEND MAP
```

```
    STEP2: MOVE to WS-COMMAREA
```

```
    STEP3: ISSUE CONDITIONAL RETURN
```

```
ELSE
```

```

IF WS-COMMAREA =
    STEP4: RECEIVE MAP
    STEP5: PROCESS DATA
    STEP6: SEND OUTPUT MAP
    STEP7: MOVE  to WS-COMMAREA
    STEP8: ISSUE CONDITIONAL RETURN
END-IF
END-IF
STEP9: REPEAT STEP3 TO STEP7 UNTIL EXIT

```

Example

The following example shows a pseudo-conversion program:

```

*****
* PROGRAM TO DEMONSTRATE PSEUDO-CONVERSION                                *
*****

IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-MESSAGE          PIC X(30).
01 WS-COMMAREA         PIC X(10) VALUE SPACES.
LINKAGE SECTION.
01 DFHCOMMAREA         PIC X(10).
PROCEDURE DIVISION.
    MOVE DFHCOMMAREA TO WS-COMMAREA
    IF WS-COMMAREA = SPACES
*****
* TRANSACTION GETTING EXECUTED FOR THE FIRST TIME                        *
*****

        MOVE 'HELLO' TO WS-MESSAGE
        EXEC CICS SEND TEXT
            FROM (WS-MESSAGE)
        END-EXEC

```

```

MOVE 'FIRST' TO WS-COMMAREA

*****
* TASK ENDS AS A RESULT OF RETURN. IF AID KEY PRESSED, NEXT      *
* TRANSACTION SHOULD BE TP002. DATA PASSED FROM WS-COMMAREA TO  *
* DFHCOMMAREA                                                    *
*****

EXEC CICS RETURN
      TRANSID('TP002')
      COMMAREA(WS-COMMAREA)
END-EXEC

*****
* IF COMMAREA IS NOT EMPTY , THEN TP002 HAS BEEN EXECUTED ONCE  *
* ALREADY, USER INTERACTION IS FACILITATED BY RECEIVE          *
*****

ELSE

EXEC CICS RECEIVE
      INTO(WS-MESSAGE)
END-EXEC

EXEC CICS SEND TEXT
      FROM (WS-MESSAGE)
END-EXEC

*****
* TASK ENDS AS A RESULT OF RETURN, NO NEXT TRANSACTION SPECIFIED *
* TO BE EXECUTED                                                  *
*****

EXEC CICS RETURN
END-EXEC

END-IF.

```

Advantages of Pseudo Conversion

Following are the advantages of pseudo conversion:

- The resources are best utilized. Resources are released as soon as the program is suspended temporarily.
- It looks as if it is in conversational mode.
- It has better response time.

Return Statements

Following are the two types of return statements which are used in CICS:

Return-1

When the following unconditional return statement is issued, the task and the transaction (program) is terminated.

```
EXEC CICS RETURN  
END-EXEC.
```

Return-2

When the following conditional return, i.e., return with TRANSID statement is issued, the control returns to the CICS with the next transid to be executed. The next transaction starts when the user presses an AID key.

```
EXEC CICS RETURN  
      TRANSID ('trans-id')  
      [COMMAREA(WS-COMMAREA)]  
END-EXEC.
```

11. CICS – AID KEYS

As we have discussed in earlier modules, AID keys are known as Attention Identifier Keys. CICS can detect only AID keys. After typing all the input, only when the user presses one of the AID keys, the CICS takes control. AID Keys include ENTER, PF1 to PF24, PA1 to PA3, and CLEAR.

Validating AID keys

The key pressed by the user is checked by using EIBAID.

- EIBAID is one byte long and holds the actual attention identifier value used in the 3270 input stream.
- CICS provides us with a pre-coded set of variables which can be used in the application program by writing the following statement:
COPY DFHAID

DFHAID

DFHAID is a copybook which is used in application programs to include CICS pre-coded set of variables. The following content is present in the DFHAID copybook:

```
01    DFHAID.
      02 DFHNULL    PIC X  VALUE IS ' '.
      02 DFHENTER   PIC X  VALUE IS ' '.
      02 DFHCLEAR   PIC X  VALUE IS '_'.
      02 DFHCLRP    PIC X  VALUE IS '|'.
      02 DFHPEN     PIC X  VALUE IS '='.
      02 DFHOPIID   PIC X  VALUE IS 'W'.
      02 DFHMSRE    PIC X  VALUE IS 'X'.
      02 DFHSTRF     PIC X  VALUE IS 'h'.
      02 DFHTRIG    PIC X  VALUE IS ' '.
      02 DFHPA1     PIC X  VALUE IS '%'.
      02 DFHPA2     PIC X  VALUE IS '>'.
      02 DFHPA3     PIC X  VALUE IS ', '.
      02 DFHPF1     PIC X  VALUE IS '1'.
```

```

02 DFHPF2    PIC X VALUE IS '2'.
02 DFHPF3    PIC X VALUE IS '3'.

02 DFHPF4    PIC X VALUE IS '4'.
02 DFHPF5    PIC X VALUE IS '5'.
02 DFHPF6    PIC X VALUE IS '6'.
02 DFHPF7    PIC X VALUE IS '7'.
02 DFHPF8    PIC X VALUE IS '8'.
02 DFHPF9    PIC X VALUE IS '9'.
02 DFHPF10   PIC X VALUE IS ':'.
02 DFHPF11   PIC X VALUE IS '#'.
02 DFHPF12   PIC X VALUE IS '@'.
02 DFHPF13   PIC X VALUE IS 'A'.
02 DFHPF14   PIC X VALUE IS 'B'.
02 DFHPF15   PIC X VALUE IS 'C'.
02 DFHPF16   PIC X VALUE IS 'D'.
02 DFHPF17   PIC X VALUE IS 'E'.
02 DFHPF18   PIC X VALUE IS 'F'.
02 DFHPF19   PIC X VALUE IS 'G'.
02 DFHPF20   PIC X VALUE IS 'H'.
02 DFHPF21   PIC X VALUE IS 'I'.
02 DFHPF22   PIC X VALUE IS '¢'.
02 DFHPF23   PIC X VALUE IS '.'.
02 DFHPF24   PIC X VALUE IS '<'.

```

Example

The following example shows how to use DFHAID copybook in an application program:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY DFHAID.
PROCEDURE DIVISION.

```

```
A000-AIDKEY-PARA.
```

```
EVALUATE EIBAID
```

```
    WHEN DFHAID
```

```
        PERFORM A000-PROCES-PARA
```

```
    WHEN DFHPF1
```

```
        PERFORM A001-HELP-PARA
```

```
    WHEN DFHPF3
```

```
        PERFORM A001-EXIT-PARA
```

```
END-EVALUATE.
```

Cursor Positioning

There are two ways to override the position specified in the map definition.

- One way is to specify the screen position relative to line and column number in the CURSOR option on the send map command.
- Other way is to move -1 to the symbolic map variable suffixed with L. Then, send the map with a CURSOR option in the SEND MAP.

Example

The following example shows how to override the cursor position for the NAME field:

```
MOVE -1 TO NAMEL
```

```
EXEC CICS SEND
```

```
    MAP ('map-name')
```

```
    MAPSET ('name-field')
```

```
    ERASE
```

```
    FREEKB
```

```
    CURSOR
```

```
END-EXEC.
```

Dynamically Modifying Attributes

While sending a map, if we want to have different attributes for a field other than that is specified in the map, then we can override that by setting the field in the program. Following is the explanation to override attributes of a field:

- To override the attributes of a field, we must include DFHATTR in the application program. It is provided by CICS.
- The attribute required can be chosen from the list and moved to the symbolic field variable suffixed with 'A'.

DFHATTR holds the following content:

```
01  CICS-ATTRIBUTES.
    05  ATTR-UXN          PIC X(01) VALUE SPACE.
    05  ATTR-UXMN         PIC X(01) VALUE 'A'.
    05  ATTR-UXNL         PIC X(01) VALUE 'D'.
    05  ATTR-UXMNL        PIC X(01) VALUE 'E'.
    05  ATTR-UXBL         PIC X(01) VALUE 'H'.
    05  ATTR-UXMBL        PIC X(01) VALUE 'I'.
    05  ATTR-UXD          PIC X(01) VALUE '<'.
    05  ATTR-UXMD         PIC X(01) VALUE '('.
    05  ATTR-U9N          PIC X(01) VALUE '&'.
    05  ATTR-U9MN         PIC X(01) VALUE 'J'.
    05  ATTR-U9NL         PIC X(01) VALUE 'M'.
    05  ATTR-U9MNL        PIC X(01) VALUE 'N'.
    05  ATTR-U9BL         PIC X(01) VALUE 'Q'.
    05  ATTR-U9MBL        PIC X(01) VALUE 'R'.
    05  ATTR-U9D          PIC X(01) VALUE '*'.
    05  ATTR-U9MD         PIC X(01) VALUE ')'.
    05  ATTR-PXN          PIC X(01) VALUE '-'.
    05  ATTR-PXMN         PIC X(01) VALUE '/'.
    05  ATTR-PXNL         PIC X(01) VALUE 'U'.
    05  ATTR-PXMNL        PIC X(01) VALUE 'V'.
    05  ATTR-PXBL         PIC X(01) VALUE 'Y'.
    05  ATTR-PXMBL        PIC X(01) VALUE 'Z'.
    05  ATTR-PXD          PIC X(01) VALUE '%'.
    05  ATTR-PSN          PIC X(01) VALUE '0'.
    05  ATTR-PSMN         PIC X(01) VALUE '1'.
    05  ATTR-PSNL         PIC X(01) VALUE '4'.
    05  ATTR-PSMNL        PIC X(01) VALUE '5'.
```

05	ATTR-PSBL	PIC X(01) VALUE '8'.
05	ATTR-PSMBL	PIC X(01) VALUE '9'.
05	ATTR-PSD	PIC X(01) VALUE '@'.
05	ATTR-PSMD	PIC X(01) VALUE "'".

12. CICS – FILE HANDLING

CICS allows us to access file data in many ways. Most file accesses are random in online system as the transactions to be processed are not batched and sorted into any kind of order. Therefore CICS supports the usual direct access methods: VSAM and DAM (Direct Access Method). It also allows us to access data using database managers.

Random Access

Following are the commands which are used for random processing:

- READ
- WRITE
- REWRITE
- DELETE

Read

READ command reads data from a file using primary key. Following is the syntax of the READ command:

```
EXEC CICS READ  
    FILE('name')  
    INTO(data-area)  
    RIDFLD(data-area)  
    LENGTH(data-value)  
    KEYLENGTH(data-value)  
END-EXEC.
```

The following table lists the parameters used in the READ command:

Parameter	Description
FILE	File name is the name of the file which we want to read. This is the CICS symbolic file name which identifies the FCT entry for the

	file. File names can be up to 8 characters long and should be enclosed in quotes if they are literals.
INTO	Data area is the variable into which the record is to be read, usually a structure in working storage. The INTO is required for the uses of the READ command.
RIDFLD	It has the name of the data area containing the key of the record which we want to read.
LENGTH	It specifies the maximum number of characters that may be read into the data area specified. It must be a halfword binary value (PIC S9(4) COMP). After the READ command is completed, CICS replaces the maximum value we specify with the true length of the record. For this reason, we must specify LENGTH as the name of a data area rather than a literal and must re-initialize this data area if we use it for LENGTH more than once in the program. An longer record will raise an error condition.
KEYLENGTH	It specifies the length of the key.

Example

The following example shows how to read a record from 'FL001' file where Student-id is the primary key:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-STD-REC-LEN      PIC S9(4) COMP.
01 WS-STD-KEY-LEN      PIC S9(4) COMP.
01 WS-STD-REC-KEY      PIC 9(3).
01 WS-STD-REC          PIC X(70).
PROCEDURE DIVISION.
MOVE +70              TO WS-STD-REC-LEN.
MOVE '100'            TO WS-STD-REC-KEY.
MOVE 3                TO WS-STD-KEY-LEN.
```

```
EXEC CICS READ
  FILE ('FL001')

  INTO (WS-STD-REC)
  LENGTH (WS-STD-REC-LEN)
  RIDFLD (WS-STD-REC-KEY)
  KEYLENGTH (WS-STD-KEY-LEN)

END-EXEC.
```

Read Command Options

Following options can be used with READ command:

- **GENERIC** - It is used when we do not know the complete key value. For example, we want a record whose primary key starts with '10' and the rest of the key can be anything. Although the key length is 3 characters, we are mentioning only 2. It is important to mention the key-length which gives the length for which it needs to do the matching. The first record that satisfies the criteria will get picked up.
- **UPDATE** - It specifies that we intend to update the record in the current transaction. Specifying UPDATE gives your transaction exclusive control of the requested record. It should be used when we want to rewrite the record.
- **EQUAL** - It specifies that we want only the record whose key exactly matches with what is specified by RIDFLD.
- **GTEQ** - It specifies that we want the first record whose key is greater than or equal to the key specified.

```
EXEC CICS READ
  FILE('name')
  INTO(data-area)
  RIDFLD(data-area)
  LENGTH(data-value)
  KEYLENGTH(data-value)

  GENERIC
  UPDATE
  EQUAL
  GTEQ

END-EXEC.
```

Read Command Exceptions

The following table shows the list of exceptions that arise during READ statement:

Exception	Description
NOTOPEN	File is not open.
NOTFND	Record that is being searched does not exist in the dataset.
FILENOTFOUND	File entry is not made in FCT.
LENGERR	Mismatch between the length specified in command and actual length of the record.
NOTAUTH	If the user does not have enough permissions to use the file.
DUPKEY	If more than 1 record satisfy the condition on the alternate key.

Write

Write command is used to add new records to a file. The parameters used in Write command are same as we had described before. Data is picked from the data area mentioned in the FROM clause. Following is the syntax for Write command:

```
EXEC CICS WRITE
    FILE(name)
    FROM(data-area)
    RIDFLD(data-area)
    LENGTH(data-value)
    KEYLENGTH(data-value)
END-EXEC.
```

Example

Following is the example to write a record in 'FL001' file where Student-id is the primary key and a new record with 101 student id will be written in the file:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-STD-REC-LEN      PIC S9(4) COMP.
01 WS-STD-KEY-LEN      PIC S9(4) COMP.
01 WS-STD-REC-KEY      PIC 9(3).
01 WS-STD-REC          PIC X(70).
PROCEDURE DIVISION.
MOVE +70                TO WS-STD-REC-LEN.
MOVE '101'              TO WS-STD-REC-KEY.
MOVE 3                  TO WS-STD-KEY-LEN.
MOVE '101Mohtahim M TutorialsPoint' TO WS-STD-REC.
EXEC CICS WRITE
    FILE ('FL001')
    FROM (WS-STD-REC)
    LENGTH (WS-STD-REC-LEN)
    RIDFLD (WS-STD-REC-KEY)
    KEYLENGTH (WS-STD-KEY-LEN)
END-EXEC.

```

Write Command Exceptions

The following table shows the list of exceptions that arise during a WRITE statement:

Exception	Description
NOTOPEN	File is not open.
FILENOTFOUND	File entry is not made in FCT.
LENGERR	Mismatch between the length specified in command and actual length of the record.

NOTAUTH	If the user does not have enough permissions to use the file.
DUPKEY	If more than 1 record satisfy the condition on the alternate key.
NOSPACE	There is not enough space in the dataset.

Rewrite

REWRITE command is used to modify a record that is already present in a file. Prior to this command, the record must be read with a READ UPDATE command. The parameters are same as described before. The syntax for the Rewrite command is as follows:

```
EXEC CICS REWRITE
      FILE (name)
      FROM (data-area)
      LENGTH (data-value)
END-EXEC.
```

Example

The following example shows how to write a record in 'FL001' file where Student-id is the primary key. A new record with 101 student id will be written in the file:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-STD-REC-LEN      PIC S9(4) COMP.
01 WS-STD-KEY-LEN      PIC S9(4) COMP.
01 WS-STD-REC-KEY      PIC 9(3).
01 WS-STD-REC          PIC X(70).
PROCEDURE DIVISION.
MOVE +70              TO WS-STD-REC-LEN.
MOVE '101'            TO WS-STD-REC-KEY.
MOVE 3                TO WS-STD-KEY-LEN.
```

```

EXEC CICS READ
  FILE ('FL001')
  INTO (WS-STD-REC)
  LENGTH (WS-STD-REC-LEN)
  RIDFLD (WS-STD-REC-KEY)
  KEYLENGTH (WS-STD-KEY-LEN)
  UPDATE
END-EXEC.
MOVE '100Mohtahim M TutorialsPnt' TO WS-STD-REC.
EXEC CICS REWRITE
  FILE ('FL001')
  FROM (WS-STD-REC)
  LENGTH (WS-STD-REC-LEN)
END-EXEC.

```

Rewrite Command Exceptions

The following table lists the exceptions that arise during a REWRITE statement:

Exception	Description
NOTOPEN	File is not open.
LENGERR	Mismatch between the length specified in command and actual length of the record.
NOTAUTH	If the user does not have enough permissions to use the file.
INVREQ	Rewrite without prior READ with UPDATE.
NOSPACE	There is not enough space in the dataset.

Delete

DELETE command is used to delete a record that is present in a file. Prior to this command, the record must be read with a READ UPDATE command. The

parameters are same as described before. The syntax of Delete command is as follows:

```
EXEC CICS DELETE
      FILE('name')
END-EXEC.
```

Following is the syntax to delete a record directly without reading it with Update option:

```
EXEC CICS DELETE
      FILE('name')
      RIDFLD(data-value)
END-EXEC.
```

Example

Following is the example for **Group Delete**. This can be done using **Generic** option, where all the records that satisfy the generic criteria will be deleted. **NUMREC** will hold the number of records deleted. The field mentioned here should be a S9(4) comp.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-STD-REC-LEN      PIC S9(4) COMP.
01 WS-STD-KEY-LEN      PIC S9(4) COMP.
01 WS-STD-REC-KEY      PIC 9(3).
01 WS-STD-REC          PIC X(70).
01 WS-NUM-REC-DEL      PIC S9(4).
PROCEDURE DIVISION.
MOVE '11'              TO WS-STD-REC-KEY.
MOVE 2                 TO WS-STD-KEY-LEN.
EXEC CICS READ
      FILE ('FL001')
      RIDFLD (WS-STD-REC-KEY)
      KEYLENGTH (WS-STD-KEY-LEN)
```

```
GENERIC  
NUMREC (WS-NUM-REC-DEL)  
END-EXEC.
```

Delete Command Exceptions

The following table shows the list of exceptions that arise during a DELETE statement:

Exception	Description
NOTOPEN	File is not open.
NOTFND	Record that is being searched doesn't exist in the dataset (DELETE with RIDFLD).
NOTAUTH	If the user does not have enough permissions to use the file.
INVREQ	Rewrite without prior READ with UPDATE.
FILENOTFOUND	File entry is not made in FCT.

Sequential Access

Following are the commands which are used for sequential processing:

- STARTBR
- READNEXT / READPREV
- RESETBR
- ENDBR

STARTBR

STARTBR is known as start browse. The STARTBR command gets the process started. It tells the CICS from where to start reading the file.

- The FILE and RIDFLD parameters are the same as in a READ command.
- The options allowed are GTEQ and EQUAL.
- UPDATE is not allowed and file browsing is strictly a read-only operation.

Syntax

Following is the syntax of STARTBR command:

```
EXEC CICS STARTBR
    FILE ('name')
    RIDFLD (data-value)
    KEYLENGTH(data-value)
    GTEQ/EQUAL/GENERIC
END-EXEC.
```

READNEXT / READPREV

When we issue a STARTBR command, it does not make the records available. It just tells from where to start reading the file. To get the first record and sequence after that, we need to use the READNEXT command.

- The FILE, INTO, and LENGTH parameters are defined in the same way as they are in the READ command. We only need the FILE parameter because CICS allows us to browse several files at once and this tells which one we want to read next.
- RIDFLD points to a data area into which the CICS will "feed back" the key of the record it just read.
- The READPREV command is almost like READNEXT, except that it lets us proceed backward through a data set instead of forward.

Syntax

Following is the syntax of READNEXT / READPREV command:

```
EXEC CICS READNEXT/READPREV
    FILE ('name')
    INTO (data-value)
    LENGTH (data-value)
    RIDFLD (data-value)
END-EXEC
```

RESETBR

The RESETBR command allows us to reset our starting point in the middle of a browse. It establishes a new browsing point.

Syntax

Following is the syntax of RESETBR command:

```
EXEC CICS RESETBR  
    FILE ('name')  
    RIDFLD (data-value)  
    GTEQ  
END-EXEC.
```

ENDBR

When we have finished reading a file sequentially, we terminate the browse using the ENDBR command. It tells the CICS that the browse is being terminated.

Syntax

Following is the syntax of the ENDBR command:

```
EXEC CICS ENDBR  
    FILE ('name')  
END-EXEC.
```

13. CICS – ERROR HANDLING

There are many types of **abends** and errors which one can face while using a CICS application. Errors can arise due to both hardware or software issues. We will be discussing about errors and error handling in this module.

CICS Errors

Following are the CICS errors which can arise during the execution of CICS applications:

- Some expected CICS errors arise when the conditions are not normal in the CICS system. For example, if we are reading a particular record and the record is not found, then we get the "Not Found" error. **Mapfail** is a similar error. Errors in this category are handled by explicit logic in the program.
- Logical errors arise due to some reasons like division by zero, illegal character in numeric field, or transaction id error.
- Errors that are related to hardware or other system conditions are beyond the control of an application program. For example, getting input/output error while accessing a file.

Error Handling Commands

CICS provides several mechanisms to identify the errors and to handle them in our programs. Following are the commands which are used to handle the expected CICS errors:

- Handle condition
- Handle Abend
- Abend
- Ignore Condition
- Nohandle

We will discuss each of them in detail now.

Handle Condition

Handle condition is used to transfer the control of the program to a paragraph or a procedure label. If the condition name specified in the exception block arises, the particular para will be given control and then we can handle that condition.

HANDLE CONDITION can handle only conditions related to CICS, not the ordinary program Abends like as data exceptions. It can handle conditions that are related only to CICS. It cannot handle the ordinary program Abends like as data exceptions. The syntax of Handle Condition is as follows:

```
EXEC CICS HANDLE CONDITION
    CONDITION(Label)
    CONDITION(Label)
    ERROR(LABEL)
END-EXEC.
```

Example

Following is the example of Handle condition:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
PROCEDURE DIVISION.
EXEC CICS HANDLE CONDITION
    DUPKEY(X0000-DUPKEY-ERR-PARA)
    NOTFND(X000-NOT-FOUND-PARA)
    ERROR(X0000-GEN-ERR-PARA)
END-EXEC.
X0000-DUPKEY-ERR-PARA.
DISPALY 'Duplicate Key Found'.
X0000-NOT-FOUND-PARA.
DISPLAY 'Record Not Found'.
X0000-GEN-ERR-PARA.
DISPLAY 'General Error'.
```

Handle Abend

If a program abends due to some reasons like input-output error, then it can be handled using Handle Abend CICS command. Following is the syntax of Handle Abend command:

```
EXEC CICS HANDLE ABEND
    PROGRAM(name)
```

```
LABEL(Label)
CANCEL
RESET
END-EXEC
```

Program name or label name is used to transfer the control to the program or paragraph if abend occurs. CANCEL is used to cancel previous HANDLE CONDITIONS. RESET is used to re-activate the previously cancelled HANDLE ABEND.

Example

Following is the example of Handle Abend:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
PROCEDURE DIVISION.
EXEC CICS HANDLE ABEND
    LABEL (X0000-HANDLE-ABEND-PARA)
END-EXEC.
X0000-HANDLE-ABEND-PARA.
DISPLAY 'Program Abended'.
```

Abend

Abend command is used to terminate the task intentionally. Using Abend command, we can set a user-defined abend code. Following is the syntax of Abend command:

```
EXEC CICS ABEND
    ABCODE(name)
END-EXEC.
```

Example

The following example shows how to use the Abend command in a program. It will abend when the program reaches this paragraph with the user-defined abend code. In the following example, it will abend with abend code D100:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
```

```
PROCEDURE DIVISION.  
EXEC CICS ABEND  
      ABCODE(D100)  
END-EXEC.
```

Ignore Condition

Ignore condition is used when we want no action to be taken if a particular abend or error happens which is mentioned inside the Ignore Condition. Following is the syntax of Ignore Condition:

```
EXEC CICS IGNORE CONDITION  
      CONDITION(Label)  
END-EXEC.
```

Example

The following example shows how to use Ignore Condition in a program. It will not abend the program even if the program throws length error as we have mentioned LENGERR inside the Ignore condition.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. HELLO.  
PROCEDURE DIVISION.  
EXEC CICS IGNORE CONDITION  
      LENGERR  
END-EXEC.
```

Nohandle

Nohandle can be specified for any CICS command. It will cause no action to be taken for any exceptional conditions that may occur during the execution of the CICS command. This command temporarily deactivates all the other handle conditions. If an exception arises during the execution of the command, the control will be transferred to the next statement after the Command. It can be used with Read, Write, Delete, etc. The syntax of Nohandle is as follows:

```
EXEC CICS
    program statements
    NOHANDLE
END-EXEC.
```

Example

Following is the example of Nohandle command. We are using it with a Read statement. If Read statement fails, it will not abend the program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
PROCEDURE DIVISION.
EXEC CICS READ
    FILE('FILE1')
    INTO(WS-FILE-REC)
    RIDFLD(WS-STDID)
    NOHANDLE
END-EXEC.
```

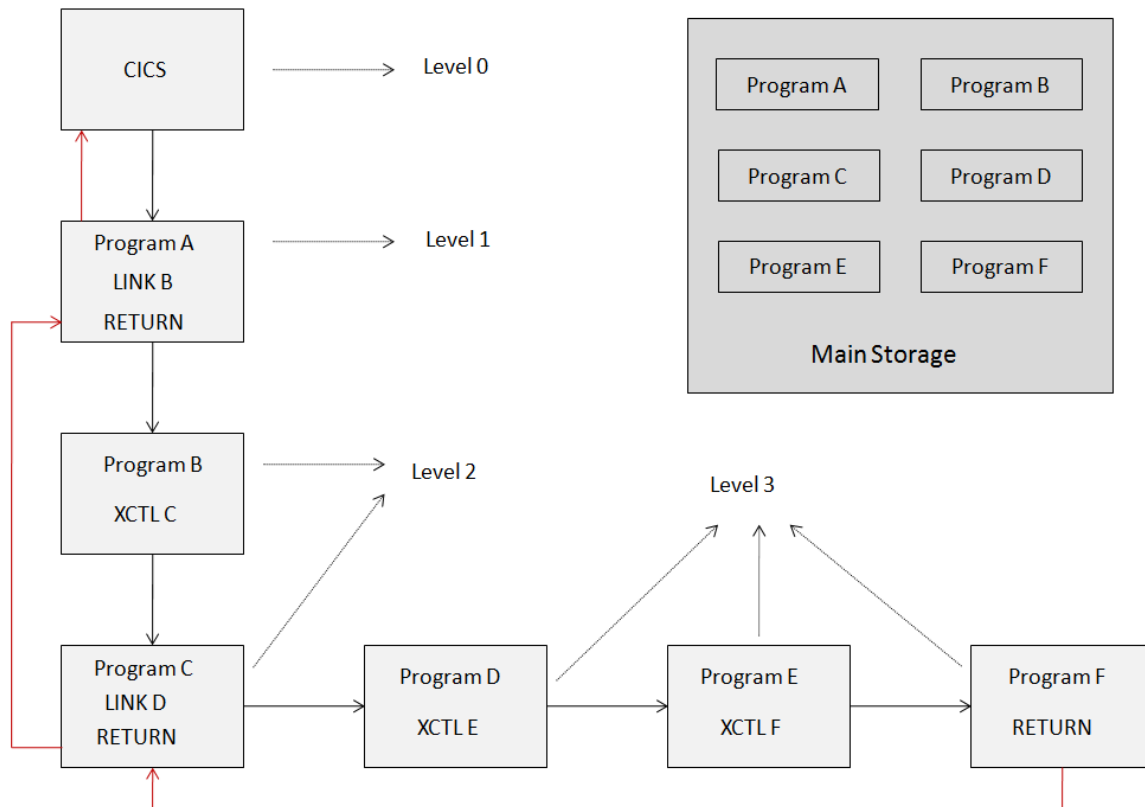
14. CICS – CONTROL OPERATIONS

CICS Program Control Program (PCP) manages the flow of application programs. All the application programs must have an entry in the Processing Program Table. Following are the commands which are used for program control services:

- XCTL
- Link
- Load
- Release
- Return

Program Logical Levels

The application programs which execute under CICS have various logical levels. The first program which receives the control directly is at highest logical level, i.e., Level 1. The Linked program is at the next logical level from the linking program. The XCTL programs run at the same level. It will be clear when we will go through Link and XCTL, later in this module. The following image shows the logical levels:



XCTL

The fundamental explanation of XCTL is as follows:

- XCTL command is used to pass the control from one program to another at the same level.
- It does not expect the control back.
- It is similar to GO TO statement.
- An XCTL program can be a pseudo-conversational.

Example

The following example shows how to use XCTL command to pass the control to another program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
WORKING-STORAGE SECTION.
01 WS-COMMAREA    PIC X(100).
PROCEDURE DIVISION.
```

```
EXEC CICS XCTL
  PROGRAM ('PROG2')
  COMMAREA (WS-COMMAREA)
  LENGTH (100)
END-EXEC.
```

This command transfers the control to be passed to program 'PROG2' with 100 bytes of data. COMMAREA is an optional parameter and is the name of the area containing the data to be passed or the area to which results are to be returned.

Link

Link command is used to transfer the control to another program at lower level. It expects the control back. A Linked program cannot be pseudo-conversational.

Example

The following example shows how to use Link command to pass the control to another program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
WORKING-STORAGE SECTION.
01 WS-COMMAREA    PIC X(100).
PROCEDURE DIVISION.
EXEC CICS LINK
  PROGRAM ('PROG2')
  COMMAREA (WS-COMMAREA)
  LENGTH (100)
END-EXEC.
```

Load

Load command is used to load a program or a table. Following is the syntax of Load command:

```
EXEC CICS LOAD
  PROGRAM ('name')
END-EXEC.
```

Release

Release command is used to release a program or a table. Following is the syntax of Release command:

```
EXEC CICS RELEASE  
    PROGRAM ('name')  
END-EXEC.
```

Return

Return command is used to return the control to the next higher logical level. Following is the syntax of Return command:

```
EXEC CICS RETURN  
    PROGRAM ('name')  
    COMMAREA (data-value)  
    LENGTH (data-value)  
END-EXEC.
```

Interval Control Operations

The interval control operations are of the following two types:

ASKTIME

ASKTIME is used to request for current time and date or timestamp. We then move this value to the working storage variable inside the program. Following is the syntax of ASKTIME command:

```
EXEC CICS ASKTIME  
    [ABSTIME(WS-TIMESTAMP)]  
END-EXEC.
```

FORMATTIME

FORMATTIME formats the timestamp into the required format based on the options, which can be YYDDD, YYMMDD, or YYDDMM for date. DATESEP indicates the separator for the DATE as does the TIMESEP variable for TIME. Following is the syntax of FORMATTIME command:

```
EXEC CICS FORMATTIME
```

```
  ABSTIME(WS-TIMESTAMP)
```

```
  [YYDDD(WS-DATE)]
```

```
  [YYMMDD(WS-DATE)]
```

```
  [YYDDMM(WS-DATE)]
```

```
  [DATESEP(WS-DATE-SEP)]
```

```
  [TIME(WS-TIME)]
```

```
  [TIMESEP(WS-TIME-SEP)]
```

```
END-EXEC.
```

15. CICS – TEMPORARY STORAGE

There are different scratch pads which are available in CICS for saving data or to transfer the data between transactions. There are five storage areas which are provided by CICS, which we will be discussing in this module.

COMMAREA

The COMMAREA behaves like a scratch pad that can be used to pass data from one program to another program, either within the same transaction or from different transactions. It should be defined in the LINKAGE SECTION using DFHCOMMAREA name.

Common Work Area

Any transaction in the CICS region can access Common Work Area and hence the format and use of it must be agreed upon by all transactions in the system that decides to use it. There is only one CWA in the entire CICS region.

Transaction Work Area

Transaction Work Area is used to pass data between the application programs that are executed with in the same transaction. TWA exists only for the duration of transaction. Its size is defined in the Program Control Table.

Temporary Storage Queue

Temporary Storage Queue (TSQ) is a feature that is provided by the Temporary Storage Control Program (TSP).

- A TSQ is a queue of records that can be created, read and deleted by different tasks or programs in the same CICS region.
- A queue identifier is used to identify TSQ.
- A record within a TSQ is identified by the relative position known as the item number.
- The records in TSQ, remains accessible until the entire TSQ is explicitly deleted.
- The records in TSQ can be read sequentially or directly.

- TSQs may be written in the main storage or the auxiliary storage in the DASD.

WRITEQ TS

This command is used to add items to an existing TSQ. Also, we can create a new TSQ using this command. Following is the syntax of WRITEQ TS command:

```
EXEC CICS WRITEQ TS
    QUEUE ('queue-name')
    FROM (queue-record)
    [LENGTH (queue-record-length)]
    [ITEM (item-number)]
    [REWRITE]
    [MAIN /AUXILIARY]
END-EXEC.
```

Following are the details of parameters used in the WRITEQ TS command:

- The QUEUE is identified by the name which is mentioned in this parameter.
- FROM and LENGTH options are used to specify the record that is to be written to the queue and its length.
- If the ITEM option is specified, CICS assigns an item number to the record in the queue, and sets the data area supplied in that option to the item number. If the record starts a new queue, the item number assigned is 1 and subsequent item numbers follow on sequentially.
- The REWRITE option is used to update a record already present in the queue.
- MAIN / AUXILIARY option is used to store records in main or auxiliary storage. Default is AUXILIARY.

READQ TS

This command is used to read the Temporary Storage Queue. Following is the syntax of READQ TS:

```
EXEC CICS READQ TS
    QUEUE ('queue-name')
    INTO (queue-record)
```

```
[LENGTH (queue-record-length)]  
[ITEM (item-number)]  
  
[NEXT]  
END-EXEC.
```

DELETEQ TS

This command is used delete the Temporary Storage Queue. Following is the syntax of DELETEQ TS:

```
EXEC CICS DELETEQ TS  
    QUEUE ('queue-name')  
END-EXEC.
```

Transient Data Queue

Transient Data Queue is transient in nature as it can be created and deleted quickly. It allows only sequential access.

- The contents of the queue can be read only once as it gets destroyed once a read is performed and hence the name Transient.
- It cannot be updated.
- It requires an entry in DCT.

WRITEQ TD

This command is used to write Transient data queues and they are always written to a file. Following is the syntax of WRITEQ TD command:

```
EXEC CICS WRITEQ TD  
    QUEUE ('queue-name')  
    FROM (queue-record)  
    [LENGTH (queue-record-length)]  
END-EXEC.
```

READQ TD

This command is used read the Transient data queue. Following is the syntax of READQ TD:

```
EXEC CICS READQ TD
    QUEUE ('queue-name')
    INTO (queue-record)
    [LENGTH (queue-record-length)]
END-EXEC.
```

DELETEQ TD

This command is used delete the Transient data queue. Following is the syntax of DELETEQ TD:

```
EXEC CICS DELETEQ TD
    QUEUE ('queue-name')
END-EXEC.
```

16. CICS – INTERCOMMUNICATION

The mutual communication that takes place between two or more systems is known as **intercommunication**.

Benefits of Intercommunication

The important benefits of intercommunication are as follows:

- We do not need to replicate the data on all the systems.
- Users need not hold connections to multiple systems for accessing the data stored on them.
- It improves the performance of the application.

Basic Terminologies

One must have a knowledge of basic terminologies used in the CICS system. Following are the basic terms:

Local System

A local system is a system that initiates a request for intercommunication.

Local Resource

A local resource is a resource that lies on the local system.

Remote System

A remote system is a system that is initiated as a result of an intercommunication request.

Remote Resource

A remote resource is a resource that lies on the remote system.

MVS Sysplex

MVS Sysplex is a configuration of multiple MVS operating systems. They work as a single system by sharing functions and programs.

CICSplex

CICSplex is commonly described as a set of interconnected CICS regions that process customer workload. A CICSplex is a set of interconnected CICS regions that own Terminals, Applications, Resources, etc.

Intercommunication Methods

There are two ways in which CICS can communicate with other systems:

- **MRO** – Multi Region Operation is used when two CICS regions within the same MVSPLEX needs to communicate with each other.
- **ISC** – Inter System Communication is used when a CICS region in a LOCAL server has to communicate with a CICS region in the REMOTE server.

17. CICS – STATUS CODES

While working with CICS, you may encounter abends. Following are the common abend codes with their description which will help you to resolve the issues:

Code	Description
ASRA	Program Check Exception
AEI0	Program ID Error
AEI9	Map Fail condition
AEIO	Duplicate Key
AEIN	Duplicate Record
AEID	End of file reached
AEIS	File is not open
AEIP	Invalid request condition
AEY7	Not authorized to use the resource
APCT	Program not found
AFCA	Dataset not found
AKCT	Time out error
ABM0	Specified map not found
AICA	Program in infinite loop
AAOW	Internal logic error

18. CICS – INTERVIEW QUESTIONS

These **CICS Interview Questions** have been designed especially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **CICS**. As per our experience, good interviewers hardly plan to ask any particular question during an interview – normally questions start with some basic concept of the subject and later, they continue based on further discussion and how you answer:

Q: What's the CICS command used to access current date and time?

A: ASKTIME command is used to access the current date and time.

Q: How do you dynamically set the CURSOR position to a specific field?

A: MOVE -1 to FIELD + L field. Mention CURSOR option in the SEND command.

Q: Which command is used to release a record on which exclusive control is gained?

A: EXEC CICS UNLOCK END-EXEC.

Q: What are the attribute values of Skipper and Stopper fields?

A: For Skipper field, use ASKIP and for stopper field use PROT.

Q: How do you set the MDT option to 'ON' status, even if data is not entered?

A: Mention FSET option in DFHMDF or set it dynamically in the program using FIELD+A attribute field.

Q: Which CICS service transaction is used to gain accessibility to CICS control tables?

A: CEDA transaction is used to gain accessibility to control tables.

Q: Into which table is the terminal id registered?

A: Terminal Control Table.

Q: What is a mapset?

A: Mapset is a collection of maps which are linked edited together to form a load module. It should have a PPT entry. It can have names from 1 to 7 chars.

Q: What is the function of the CICS translator?

A: The CICS translator converts the EXEC CICS commands into call statements for a specific programming language.

Q: What are the differences between an EXEC CICS XCTL and an EXEC CICS LINK command?

A: The XCTL command transfers the control to an application program at the same logical level and it does not expect the control back, while the LINK command passes the control to an application program at the next logical level and expects the control back.

Q: What is EIB? How it can be used?

A: CICS automatically provides some system-related information to each task in a form of EXEC Interface Block (EIB), which is unique to the CICS command level. We can use all the fields of EIB in our application programs right away.

Q: What information can be obtained from the EIBRCODE?

A: The EIBRCODE tells the application program if the last CICS command was executed successfully or not.

Q: What is the effect of including the TRANSID in the EXEC CICS RETURN command?

A: The next time the end-user presses an attention key, CICS will start the transaction specified in the TRANSID option.

Q: What is the function of the EXEC CICS HANDLE CONDITION command?

A: To specify the paragraph or program label to which the control is to be passed if the "handle condition" occurs.

Q: What is the difference between the INTO and the SET option in the EXEC CICS RECEIVE MAP command?

A: The INTO option moves the information in the TIOA into the reserved specified area, while the SET option simply returns the address of the TIOA to the specified BLL cell or "address-of" a linkage section.

Q: What is the function of DFHMDF BMS macro?

A: The DFHMDF macro defines fields, literal, and characteristics of a field.

Q: What is the difference between getting the system time with EIBTIME and ASKTIME command?

A: The ASKTIME command is used to request the current date and time. Whereas, the EIBTIME field has the value at the task initiation time.

Q: What is the function of the Terminal Control Table?

A: The TCT defines the characteristics of each terminal with which CICS can communicate.

Q: What is a deadlock?

A: A deadlock occurs when a task is waiting for a resource held by another task which, in turn, is waiting for a resources held by the first task.

Q: Explain the term Multi Region Operation.

A: MRO is the mechanism by which different CICS address spaces within the same CPU can communicate and share resources.

Q: What is meant by program reentrance?

A: A program is considered reentrant if more than one task can execute the code without interfering with the other tasks execution.

Q: What is the common work area?

A: The common work area is a storage area that can be accessed by any task in a CICS system.

Q: What is the meaning and use of the EIBAID field?

A: EIBAID is a key field in the execute interface block; it indicates which attention key the user pressed to initiate the task.

Q: What is BMS?

A: BMS stands for Basic Map Support. It allows you to code assembler level programs to define screens.

What is Next?

Further, you can go through the examples which you have practiced with the subject and make sure you are able to speak confidently on them. If you are a fresher, then the interviewer does not expect you to answer very complex questions. Hence you should have a strong hold over the fundamental concepts of the subject.