**CICS Notes**

**sites**

**Tutorials**

<http://www.mainframes360.com/2010/03/cics-tutorials.html>

<http://www.mainframestechhelp.com/tutorials/cics/page1.htm>

**Abends**

**Utilities**

**Example Programs**

-----------------------------------------------------------------------------------------------------------------------------

**Imp Points**

* **ABBREVATIONS**

**CICS** – Customer Information Control System

* On CICS you setup a transaction to run application programs. Every transaction is identified **by a unique 4-byte code called transaction ID or TRANSID.** CICS maintains a table of all the transaction definitions. This table is called the **PCT (Program control table).**
* A transaction definition contains the TRANSID and the name of the first program to run. Before you use a transaction, one must define it in the PCT table. **PCT ties together the TRANSID the user enters and the program.**
* CICS also maintains a table of all program definitions. This table is called **PPT (Processing program table).** **A program definition contains the name of the program and the load-library address of the program**. Before you use a program, one must define it in the PPT table.
* On the CICS Terminal (like a web browser), when the user enters the TRANSID, the CICS software searches the PCT table for information about the transaction. If a TRANSID is found, CICS searches the PPT table for a match on the PROGRAM. CICS loads the program from disk into main storage and runs it.
* **An active, running instance of a transaction is called task. The output results are displayed on screen and the CICS transaction ends.**
* Important CICS Utility Transactions **CEDA (To define or query the resource), CEMT, CEDF(Debug)**
* If a CICS program is modified and compiled, the new copy of the load module has to be loaded into main-storage. **CEMT is also used to refresh the load-module.**
* **CICS maintains a special table for all file definitions called FCT (File Control Table).** The FCT table contains the physical name of the dataset like SYSADM.PASSENGER.MASTER and a DD-name like PASSMAST. Before you use a file in a CICS program, you must define it in FCT.
* **CICS maintains a special table for DB2 plans as well called RCT (Resource Control table).** The RCT table ties together the TRANSID, the user enters and the Db2 PLAN of the application program. Before you run CICS applications that access DB2 database, an entry must be added in the RCT.

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

**Questions and their Explanation**

**Q. What is online processing? How does it differ from Batch processing?**

**ANS.**

The origin of the term Batch-processing dates back to era of Mainframe computers. Typically, when a huge volume of input data is processed in bulk, the input data are stored in a batch of files, and are then processed by software in batches.

Jobs are set up to complete without any manual intervention. The operator specifies

1. The program to run
2. the input files and
3. the output files for storing the results.

The computer program shall take the set of input files, process the data, and produce a set of output files. The operator submits(SUB) the job for execution. Depending upon the demand on the computer, the job may take a while to complete. On completion, the job alerts and notifies the operator.

Batch-oriented applications are offline systems. You do not get immediate outputs to the inputs you supply. During execution, there is no interaction between the program and the users.

Batch applications are still critical in many core businesses like banking, insurance, telecom. For example, telephone billing has to be done at the month-end, tallying the bank accounts balance, interest calculation, printing statements has to be done at the End-of-Day. These are high-volume, repetitive tasks which must be completed reliably on-time. They do not require human interaction.

The opposite of batch processing is interactive processing. The user types input on the screen, the program processes the input right-away and displays the output, the user types some more input, the program processes it and you get some more output. The input-Process-output cycle of events continue, until the final solution. The program gives an immediate(quick) response or output to the inputs you supply, on the fly. Such interactive applications are online systems.

**A transaction is the basic unit-of-work in online systems. A transaction runs one or more programs to process the data.** A user types the input data on the computer screen, and presses Enter to start the transaction. The transaction invokes one or more programs to process the data right away. Immediately, the output results are displayed on the terminal and the transaction ends.

A single business-transaction (such as enrolment of a customer, retirement of an employee) may involve one or more CICS transactions.

**Q. What is CICS? What is CICS Application Programming Interface(API)?**

**ANS**

CICS stands for Customer Information Control System. CICS is a giant, robust Transaction Server. CICS can manage large volumes of transactions. CICS allows transactions to run concurrently at the same time serving many online users. CICS offers ACID properties. CICS allows sharing of data and resources. Different CICS services are exposed through a rich set of API (Application programming interface) commands.

To make Cobol program(s) interactive, we code special CICS API commands in the program. CICS programs are not very different from ordinary Cobol programs.

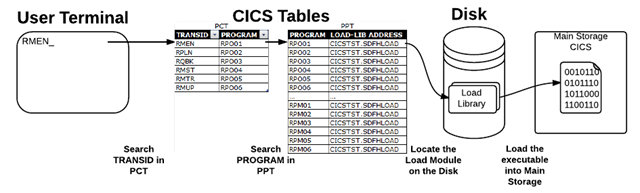
**Q. What are CICS transactions, tasks and programs? What are CICS Tables?**

**ANS**

On a PC you double-click a Program, to start it. On CICS you setup a transaction to run application programs. Every transaction is identified by a unique 4-byte code called transaction ID or TRANSID. CICS maintains a table of all the transaction definitions. This table is called the PCT (Program control table). A transaction definition contains the TRANSID and the name of the first program to run. Before you use a transaction, one must define it in the PCT table. PCT ties together the TRANSID the user enters and the program.

CICS also maintains a table of all program definitions. This table is called PPT (Processing program table). A program definition contains the name of the program and the load-library address of the program. Before you use a program, one must define it in the PPT table.

On the CICS Terminal (like a web browser), when the user enters the TRANSID, the CICS software searches the PCT table for information about the transaction. If a TRANSID is found, CICS searches the PPT table for a match on the PROGRAM. CICS loads the program from disk into main storage and runs it. An active, running instance of a transaction is called task. The output results are displayed on screen and the CICS transaction ends.



**Q. How to define transactions and programs in the PCT and PPT tables?**

**ANS.**

Just as you code your own custom CICS programs, IBM supplies some ready-made free CICS programs for doing common day-to-day tasks. For example, you can use the free transaction CEDA, to define a new TRANSID in the PCT Table. Such free transactions shipped by IBM for doing routine tasks are called utility transactions.

CEDA is the utility transaction to DEFINE(add) new entries in the CICS tables.

To define a new TRANSID in the PCT table, type CEDA DEFINE TRANSACTION on the CICS terminal.

To define a new PROGRAM in the PPT table, type CEDA DEFINE PROGRAM on the CICS terminal.

CEMT is a utility transaction to INQUIRE or query the entries(rows) in CICS Tables.

To find out a TRANSID in PCT, type CEMT INQUIRE TRANSACTION.

To find a PROGRAM in PPT, type CEMT INQUIRE PROGRAM on the CICS terminal.

If a CICS program is modified and compiled, the new copy of the load module has to be loaded into main-storage. CEMT is also used to refresh the load-module.

**CECI is another utility transaction; it is the CICS Command Level Interpreter.** To run and test some CICS commands, outside a CICS program (without writing a complete Cobol Program), use the CECI utility. For example, after designing CICS GUI screens, you would like to test them, see how they look, use CECI utility.

**Q. What are the other special CICS tables?**

**Ans**

CICS maintains special tables like PCT, PPT, FCT, RCT, TST, DCT and several others. **CICS maintains a special table for all file definitions called FCT(File Control Table).** The FCT table contains the physical name of the dataset like SYSADM.PASSENGER.MASTER and a DD-name like PASSMAST. Before you use a file in a CICS program, you must define it in FCT.

**CICS maintains a special table for DB2 plans as well called RCT (Resource Control table).** The RCT table ties together the TRANSID, the user enters and the Db2 PLAN of the application program. Before you run CICS applications that access DB2 database, an entry must be added in the RCT.

**Q. 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 worry about OS, hardware and others.

Initially CICS was on macro and later upgrades to command level.

This book is written on **CICS/MVS V2R3.**

**Q. Conversation & Pseudo Conversation.**

Program can communicate with user by a pair of SEND & RECEIVE commands. But in

between the SEND & 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 & pass control**

**information to next transaction & the next transaction 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 throughout the communication.) IRC as

Pseudo conversation (Once the message sent, the line is freed and it again gets the

resource when the other side replied.)

**Q. Multitasking & Multithreading**

Operating systems allows execution of more than one task concurrently & this is

called Multitasking. If the one or more concurrent tasks use the same copy of the program,

then is called Multithreading. So it is obvious that multitasking is a Subset of

multithreading.

**Q. Reentrant & Quasi- Reentrant**

If the same copy of the program is used by multiple tasks, then the system should

take care of 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 & continue processing after an interruption by the SVC call of OS. Batch program are non-Reentrant. Reentrancy under CICS environment is called quasi entrant as the

interruption in CICS may involve more than once SVC calls or no SVC at all.

**COBOL program should be complied with RENT option for Reentrancy in Multithreading environment.**

**Q. supervisor call (SVC)**

In computers, especially IBM mainframes**, a supervisor call (SVC) is a processor instruction that directs the processor to pass control of the computer to the operating system's supervisor program.** Most SVCs are requests for a specific operating system service from an application program or another part of the operating system. Application program developers usually use a language function or macro instruction to make the request (for example, to get allocated more memory for the program to work with). The language compiler or assembler generates the instruction that includes the specific SVC request. Each service has a preassigned SVC number. **When the computer's processor executes the instruction that contains the SVC, the code representing "SVC" causes a program interrupt to occur, which means that control of the processor is immediately passed to the operating system supervisor program.** The supervisor then passes control to programming that performs the service that goes with the specified SVC number.

An SVC routine is a program within the supervisor that performs the service indicated by the specific SVC instruction.

Q. CICS Control Program & Control Tables

CICS Nucleus consists of IBM-Supplied CICS control programs & corresponding user specified CICS control Tables. The important tables with respect to development point of

view are given below:

|  |  |
| --- | --- |
| Control Table | Function |
| PCT | The transaction & the main program associate with the transaction should be  registered in program table. Task control Program(KCP) refers PCT. |
| PPT | All the CICS programs & maps have to be registered in Processing Program Table.  File Control Table(PCP) refers PPT. |
| FCT | All VSAM file used in the CICS program 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 & Password should be registered in the Sign On Table. |
| TCT | All the terminals should be registered in Terminal Control Table. |
| PLT | All the program that need to be automatically started during CICS start up &  Shut down should be listed in Program List Table. |
| JCL | Control Information of system logs & journal files is stored in Journal Control  Table. Journal Control Program refers to JCT. |

**Q. General Syntax Of CICS Statement**

EXEC CICS function

[(option(argument value)]

[(option(argument value)]

END-EXEC.

**Q. Restricted COBOL commands in CICS environment**

|  |  |
| --- | --- |
| OS SVC Triggering statements | ACCEPTS 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 & that returns control back to CICS.

**Q. MAP DESIGN**

Most of all installation **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 & MAPSET**

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

**PHYSICAL MAP & SYMBOLIC MAP**

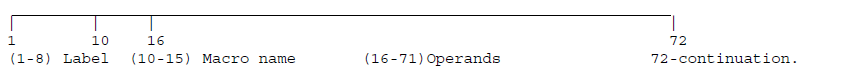
Physical maps control the screen alignment, sending & receiving of CONSTANTS & data to & from a terminal. They are coded using BMS macros, assembled & link edited into CICS LOAD LIBRARY. **They ensure the device independence in application programs.**

**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 using assembling, they are place in **a COPY library** & then copied into CICS programs using COPY statement. They ensure device & format independence to the application programs.

BMS Macro Coding Sheet

Since BMS map definitions are purely assembler macros, the following coding

convention must be maintained.



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 is 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 & continues from the 16th column of next time.

**DFHMSD**

It is used to define a mapset with its character & 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 & FINAL to indicate the end of mapset**. Alternatively, symbolic parameter &SYSPARM can be coded in the TYPE of defining DFSMSD & 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 & OUT for output maps like screens & INOUT for input-output maps like update screens.

**LANG**.

In 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 1 mapset to share same storage area. MAP-IOAREA will be redefined multiple times to achieve it.

**TIOAPEX**.

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

**CTRL**.

Device control request are place here. Multiple parameters are separated by comma.

**FREEKB** is used 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 & 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 any

number of DFHMDI. Some of the important operands of DFHMDI are below:

**SIZE** - It has two arguments namely length & breadth & as a whole the size of the map

is specified here.

**LINE** - The map starting line is mentioned here.

**COLOUMN** - The map starting column width the LINE is mentioned here.

**JUST** - RIGHT or LEFT is coded here to inform the justification of the map within

mapset,

1. **The above four parameters decides the size & location of map within map**

**set CTRL & TIOPEX 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 fields. The two arguments are

**line & column**. It is the position where the attribute byte of the fields starts.

**LENGTH**.

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

**ATTRIB**.

All the input & 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 fields. UNPROT is coded for input & output fields. PROT is coded for output &

stopped field. ASKIP is coded for screen literals & skipper fields. The cursor

automatically skipped to next fields & so you cannot enter data into skipper

field.

2. NUM – 0-9, Periods & - are the only allowed characters.

3. BRT/NORM/DRK – Mutually exclusive parameter 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 place 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 default value. Code LEFT for numeric fields.

**PICIN & PICOUT**

If defines the picture clause of the symbolic map in COBOL & useful for numeric field

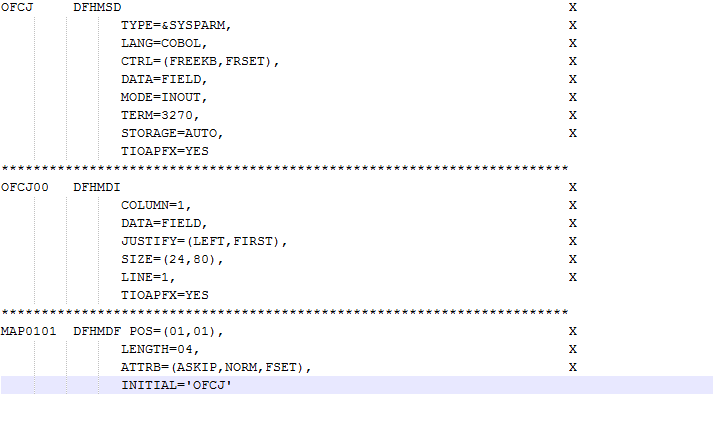
editing.

**INITIAL**

The default value of the fields 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.

**Sample BMS macro**

****

**Q. How a field looks like in a Symbolic map?**

Let the label of one DFHMDF is EMPNAME in the map EMPDET & 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 EMPDFTOO 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 fields 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.**

If 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) & OUTPUT(SEND) are with respect to program.

**Q. Attribute Field and its values**

We have just seen that attribute field is of 1 byte that specified the attributes of

the field. 1 byte is of eight bits & the values in each bit has its own meaning.

For Example,

Bit 2 – ‘0’ indicates the field is unprotected & ‘1’ indicates the field is protected.

Bit 3 – ‘0’ indicates the field is alphanumeric & ‘1’ indicates the field is numeric.

Bit 2 & 3 – ‘11’ indicates that the field is Auto-skip.

Bit 4 & 5 – ‘00’ indicates Normal intensity & non-detectable.

‘01’ indicates Normal intensity & detectable.

‘10’ indicates High intensity & detectable.

‘11’ indicates Dark & Non-detectable.

Bit 7 – ‘0’ indicates MDT is OFF ‘1’ indicated MDT is ON.

Modified Data Tag(MDT).

MDT is 1 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 used 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 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 & it is fine with his requirement, then he won’t touch the field & 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.

**Q. CURSOR Positioning**

Positioning of cursor is an important area in screen design. By default, the cursor

will be place in the first unprotected field.

**Static Positioning**

If IC is coded in the attribute operand of DFHMDF macro, then the cursor will be

place in that field. If IC is coded in more than 1 field, then last field with IC will get

the cursor.

**Symbolic Positioning**

Move -1 to length of the fields where you want place the cursor & send the map with

CURSOR option. This is device independent method & 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 & 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 column in 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.

**Q. Dynamic attributes 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 & 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 FIELD(A) \_ will achieve this.

**Q. SKIPPER & STOPPED field**

**Skipped is unlabelled 1 byte field with the auto-skip attribute & stopped is**

**unlabelled 1 byte field with protected attribute.**

Skipped field skips the cursor to next field when the current field is filled up to

its full length whereas stopped field stops entering any more character after the length of the field is reached. The user has to press RESET key to proceed further.

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

**Q. 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) & places the mapped data. The values of the field can be read in the program by referring to filename+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) & takes the data to the terminal.

EXEC CICS

SEND MAP (map name)

MAPSET (mapset name)

END-EXEC.

Other options in SEND MAP are:

ERASE: Erase 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 message 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, ALARM & FRSET can be also coded & the meaning is already discussed.

**Q. TEXT Building – SEND-TEXT, ACCUM & 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 & 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 & the building of message can

happen in multiple stages. If you want to send the text stream only after the building of the entire paragraph being built. They are accumulated & will not be sent immediately.

Once the page is built with complete message of the entire paragraph, issue SEND PAGE command that will send the complete text stream to the terminal. HEADER is place in the first SEND TEXT & 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 & PF8 with page up & down commands respectively. PURGE MESSAGE is used to purge the message that is accumulated but not yet send.

Format of HEADER & TRAILER

If TEXT-HEADER is the variable used in HEADER command & 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. => 1 byte control field used by BMS

05 FILLER PIC X(27) VALUE ‘ORDER INVENTORY – Page && - ‘.

=>Actual text in the header.

TRIALER also can be coded in the same way. Automatic –page-number- identifying

character will be spaces for TRIALER working storage variable.

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

**Q. 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. If is used to establish the device control options dynamically.

Syntax:

EXEC CICS SEND CONTROL

[CURSOR(data value)] [ERASE|ERASEUP] [FREEKB] [ALARM] [FRSET]

END-EXEC.

**Q. 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 & not to be

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 & blank character 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.

**Q. 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 & 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 & OPCLASS name the route list & 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 defines in the working storage section & **ERRTERM specify the terminal ID where the error message (if any) to be sent.**

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

**TTTTrrOOOsrrrrrr – 8 bytes named ‘r’ are declared as spaces. TTTT names the terminal identifier as in TCT & OOO specify the operator id as in SNT. S is status flag. Code as many 16 bytes fields as the destinations & 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 which users of the specified operator

class are signed on. This is done using OPCLASS.

**Q. Pseudo logic in CICS program is achieved in the 3 ways.**

1. Multiple programs & Multiple Transaction Ids.

Mainframe Refresher Part-1 CICS-Page: 10

The conversation program is logically & physically divided into multiple programs

**utility transactions**

**CEDA**

CEDA is the utility transaction to DEFINE(add) new entries in the CICS tables.

To define a new TRANSID in the PCT table, type **CEDA DEFINE TRANSACTION** on the CICS terminal.

To define a new PROGRAM in the PPT table, type **CEDA DEFINE PROGRAM** on the CICS terminal.

**CEMT**

CEMT is a utility transaction to INQUIRE or query the entries(rows) in CICS Tables.

To find out a TRANSID in PCT, type **CEMT INQUIRE TRANSACTION**.

**CECI**

CECI is another utility transaction; it is the CICS Command Level Interpreter. To run and test some CICS commands, outside a CICS program (without writing a complete Cobol Program), use the CECI utility.

For example, after designing CICS GUI screens, you would like to test them, see how they look, use CECI utility.

**ABENDS**