Unit 2. Maps

Objectives

- **BMS Maps**
- **Map Definition**
- **Map Preparation**
- **Map Usage**

Figure: 2-1. Objectives

Notes :

BMS Maps

- **Map is a screen, defined through BMS.**
- **BMS map is nothing but a program written in Assembly language.**
- **A set of Assembler macros(BMS macros) are provided by CICS for the BMS map coding.**
- **A map represents a BMS coding for a screen panel.**

Figure: 2-2. BMS Maps

Notes :

Types of Maps

- **Physical Map**

- **Symbolic Map**

Figure: 2-3. Types of Maps

Notes :

The Physical map is a table of Screen display field information.
It resides in the program Load Library.
The Physical Mapset is loaded when the executing program requests SEND or RECEIVE BMS Services.

The Symbolic map description is a group structure of COBOL field definitions.
It description resides in the COPY library.
It is placed in the source program by the compiler.
If map changes affect the symbolic map description, recompile all programs which reference the COPY book.

Physical Map

Physical Map defines

- **Starting Position**
- **Length**
- **Field Characteristics(Attribute byte)**
- **Default data for each field**

It Allows BMS to add BCC's and commands for output in order to construct an o/p NMDS. A Physical map will be in the form of a load module.

Figure: 2-4. Physical Map

Notes :

The Mapset is a resource which must be defined to CICS by the systems programmer.

BMS uses the physical map for both input and output operations.

NMDS: Native Mode Data Stream, is a mixture of Buffer Control Characters(BCCs) and text data. Therefore, in order to send or receive the formatted screen, the application program must format all BCCs and text for the output screen or must interpret all BCCs for the input screen, respectively, based on the hardware protocol of the terminal.

BCC : Buffer Control Characters

Symbolic Map

A symbolic map is a copy library member which can be included in the application program for defining screen fields.

It defines the map fields used to store variable data referenced in COBOL program.

For each field name in DFHMDF macro BMS creates 2 fields for i/p , 2 fields for o/p and 1 field for both input and output, by placing a character suffix to the original field name.

Figure: 2-5. Symbolic Map

Notes :

The symbolic map starts with the 01 level definition of map name defined in the DFHMDF macro with suffix ‘I’ for input map an ‘O’ for o/p map.

Next a filler of pic x(12) which is the TIOAPrefix and is created if the option TIOAPFX=YES is specified in DFHMSD macro.

Symbolic Map Fields

Name + L → Half word binary field pic s9(4) comp.

Name + F → Flag byte, for i/p field X'80' if modified else X'00'.

Name + A → Attribute byte for both i/p and o/p fields.

Name + I → I/p data field X'00' will be placed if no data is entered.

Name + O → The o/p data field.

Name + C → A single Character field that contains the attribute for extended color.

Name + H → A single –character field that contains the attribute for Extended highlighting.

Figure: 2-6. Symbolic Map Fields

Notes :

A 12- byte TIOA(Terminal Input/Output Area) prefix is crated when you code TIOAPFX = YES in the DFHMSD macro definition of the BMS.

Map Definition Macros

There are 3 CICS supplied macros for coding BMS maps.

DFHMSD Mapset definition macro.

DFHMDI Map definition macro.

DFHMDF Field definition macro.

Figure: 2-7. Map Definition Macros

Notes :

DFHMSD Macro

Syntax :

```
Mapsetname   DFHMDF   TYPE=&SYSPARM,                x
                      MODE=INOUT,                   x
                      LANG=COBOL,                   x
                      STORAGE=AUTO,                 x
                      TIOAPFX=YES,                  x
                      CNTL=(FREEKB,FRSET,PRINT)
```

Figure: 2-8. DFHMSD Macro

Notes :

Mapsetname is of 1 to 7 characters in length.

DFHMSD macro is used to define a mapset and its characteristics or to end a mapset definition.

TYPE : To define the Map type.

1. DSECT : For symbolic map.
2. MAP : For physical map.
3. &SYSPARM : For symbolic and Physical map.

MODE : TO indicate input/output operation.

1. IN : For the input map.
2. OUT : For the output map.
3. INOUT : For the input/output map

LANG : To define the language of the application program.(COBOL,ASM,PLI)

STORAGE : To acquire a separate symbolic map area for each mapset

TIOAPFX : To reserve the prefix space (12 bytes) for BMS commands to access TIOA properly.

CNTL : To define the device control requests.

1. FREEKB : To unlock the keyboard.
2. FRSET : To reset the MDT to zero.
3. ALARM : To set an alarm at screen display time.
4. PRINT : To indicate the mapset to be sent to the printer.

DFHMDI Macro

Syntax :

```
Mapname   DFHMDI   SIZE=(line,column),          x
                   LINE=number,                  x
                   COLUMN=number,                x
                   JUSTIFY=LEFT
```

Figure: 2-9. DFHMDI Macro

Notes :

Mapname is of 1 to 7 characters in length.

DFHMDI macro is used to define a map and its characteristics in a mapset. It can be used often as you wish within one DFHMSD macro.

DFHMDF Macro

Syntax :

```
Fieldname   DFHMDF   POS=(line,column),                    x
                     LENGTH=number,                        x
                     JUSTIFY=RIGHT,                        x
                     INITIAL=Char data,                    x
                     ATTRB=(UNPROT/PROT, NUM, FSET, IC)
```

7

Figure: 2-10. DFHMDF Macro

Notes :

FSET – Sets the Flag field .
FRSET – Resets the Flag field.
IC- Initial Cursor will be paced in that field

Field Concepts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	.	.	.	.	80
1																						
2	%	N	A	M	E				:	%						%	|					
3	%	A	G	E					:	%		%	|									
4	%	P	L	A	C	E			:	%									%	|		
5																						
6																						
7																						
8																						
9																						
10																						
11	%	M	S	G	:														%	|		
12																						
13																						
14																						
15																						
16																						
17																						
.																						
.																						
.																						
.																						
.																						
24																						

\| ⟶ **Stopper Field**

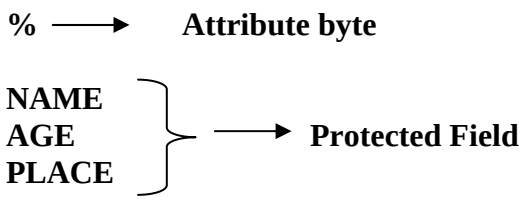


Figure: 2-11. Field Concepts

Notes :

Stopper Field – indicates the ending position of its preceding unprotected field

Attribute Byte Concept

PROTECTED/UNPROTECTED	**Operator allowed/disallowed to enter data.**
ASKIP	**Protected and Cursor will automatically skip over this field to the next unprotected field.**
NUM	**Using appropriate hardware, the operator may Only key in numbers 0-9 and special characters.**
MDT/NOMDT	**If the modified data tag (MDT) is ON in the attribute for a field then the data is transmitted to the program. If it is OFF no data is transmitted.**
IC	**Insert Cursor is not an attribute byte characteristic, it is one way to position the cursor under the first position of a field.**

Figure: 2-12. Attribute Byte Concept.

Notes :

Attribute Byte - used to specify the characteristics for a field in a map.

Sample Map Definition

```
MAPSET1  DFHMSD  TYPE=&SYSPARM,MODE=INOUT,TERM=ALL                    X
                 LANG=COBOL,TIOAPFX=YES,STORAGE=AUTO
MAP1     DFHMDI  SIZE=(24,80),LINE=1,COLUMN=1
         DFHMDF  POS=(2,2),LENGTH=8,INITIAL='NAME   :',ATTRB=ASKIP
NAME     DFHMDF   POS=(2,11),LENGTH=7,ATTRB(UNPROT,IC)
         DFHMDF  POS=(3,2),LENGTH=8,INITIAL='AGE    :',ATTRB=ASKIP
AGE      DFHMDF  POS=(3,11),LENGTH=3,ATTRB(UNPROT,NUM)
         DFHMDF  POS=(4,2),LENGTH=8,INITIAL='PLACE  :',ATTRB=ASKIP
PLACE    DFHMDF  POS=(4,11),LENGTH=10,ATTRB(UNPROT)
         DFHMDF  POS=(11,2),LENGTH=4,INITIAL='MSG:',ATTRB=ASKIP
MSG      DFHMDF  POS=(11,17),LENGTH=25,ATTRB(UNPROT)
         DFHMSD TYPE=FINAL
         END
```

Figure: 2-13. Sample Map Definition.

Notes :

A mapset can hold several map but most mapsets hold one map, hence the map can sometimes generically refer to the mapset with its map.

Maps are defined using assembler macros, and therefore you must obey assembler-coding rules:

- Label begins in column 1.

- Macro begins in column 10.
- Operands begin in column 16.
- Continued lines must have a non-blank continuation character in column 72,operands continue in column 16.

Map Preparation

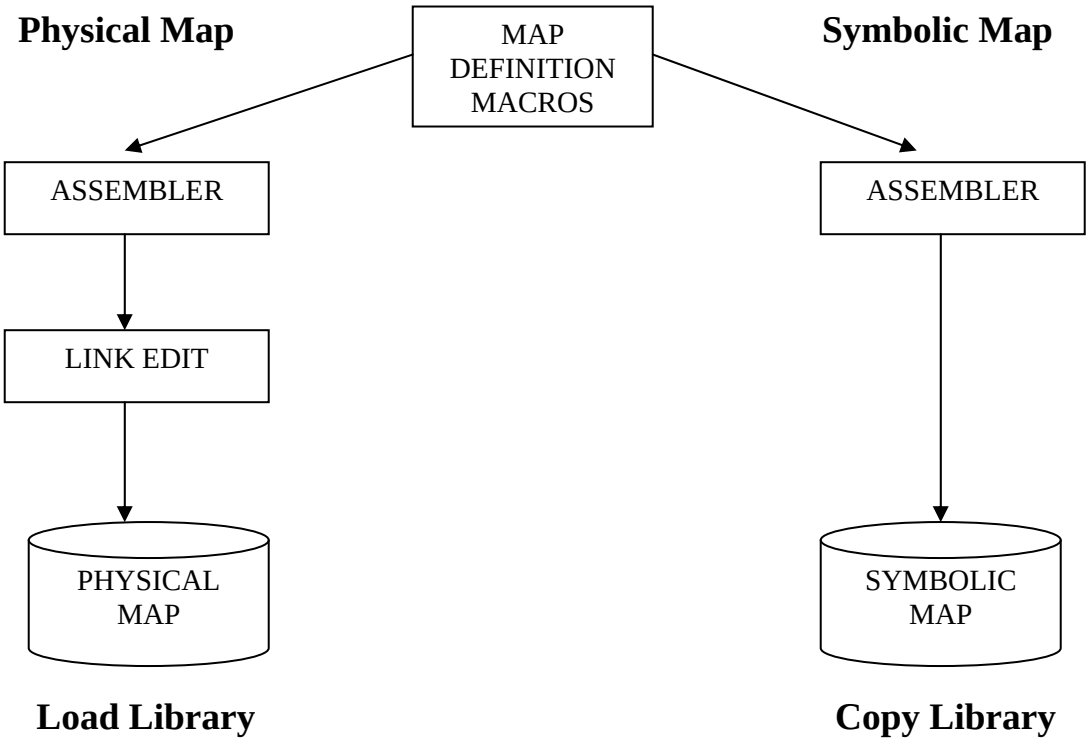
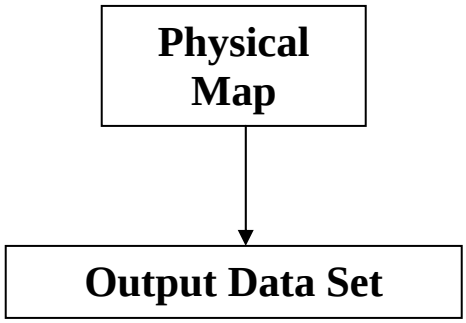


Figure: 2-14. Map Preparation.

Notes :

BMS Macro generates the Physical map(it stores in Load Library) and Symbolic map (it stores in Copy Library)

Output Mapping



“MAPONLY”

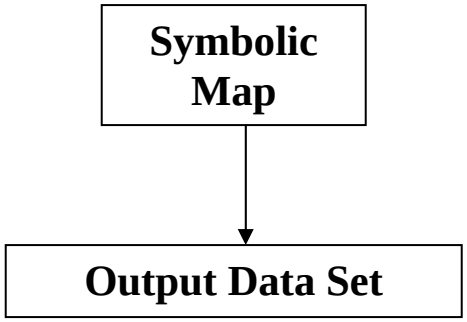
```
EXEC CICS SEND MAP(‘mapname’)
               MAPSET(‘mapsetname’)
               MAPONLY
END-EXEC.
```

Figure: 2-15. Output Mapping

Notes :

SEND MAPONLY will use physical map only. No information about the symbolic map is sent.

Output Mapping



“DATAONLY”

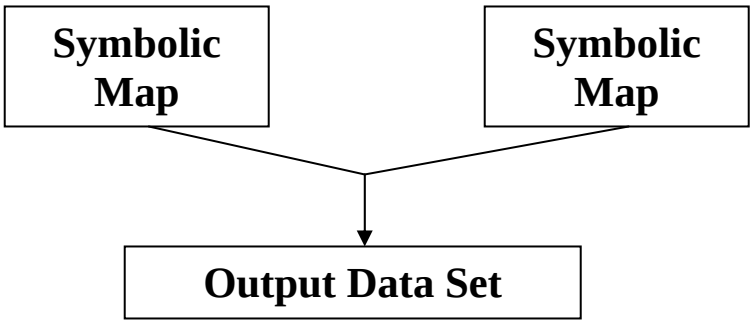
```
EXEC CICS SEND MAP(‘mapname’)
               MAPSET(‘mapsetname’)
               DATAONLY
END-EXEC.
```

Figure: 2-16. Output Mapping.

Notes :

SEND DATAONLY will use only the Symbolic description map fields.

Output Mapping



Neither "MAPONLY" nor "DATAONLY"

```
EXEC CICS SEND MAP('mapname')
               MAPSET('mapsetname')
END-EXEC.
```

Figure: 2-17. Output Mapping.

Notes :

BMS uses both physical map and the information in your symbolic area. It merges the two together. This is an appropriate map for sending an initial map with program data included.

Input Mapping

```
EXEC CICS RECIEVE MAP('mapname')
                  MAPSET('mapsetname')
                  RESP(ERR-CODE)
END-EXEC.
```

Figure: 2-18. Input Mapping.

Notes :

BMS uses the physical map, parses the data stream into appropriate fields in the program's symbolic map description.

Common Declarations

WORKING-STORAGE SECTION.

COPY DFHAID

COPY DFHBMSCA.

COPY MAP1.

PROCEDURE DIVISION.

……

……

……

Figure: 2-19. Common Declarations.

Notes :

COPY DFHAID declares attention identifier (AID)constants for determining which key was pressed by the operator (e.g. CLEAR key, PF keys etc.)

COPY DFHBMSCA declares certain needed constants, which can be used to alter field attributes bytes.

COPY ‘name’ brings in COBOL source statements (Symbolic maps description is copied into the Program).