1.3 Data Division

1.3.1 *Introduction*

The data division contains declarations which define the type, length and value of data items used as operands in the program, together with declarations which define the interface between the application program and the rest of the PTS System. The use of directives in the data division is discussed in Section 1.2. The general layout of the data division is shown below.

There must be one terminal class declaration for each terminal class used by the program. The terminal class declarations may be made in any sequence, but they must all appear at the start of the data division.

There must be one begin block declaration for each block identifier specified in the work block declarations. The begin block declarations may be made in any sequence and must appear after the last terminal class declaration.

```
┌ TERM etc.               – Terminal class declaration(s)
│   CWB etc.  ┐
│   TWB etc.  │
│   UWB etc.  ├           – Workblock declarations
│   DWB etc.  │
│   SWB etc.  ┘
│ [DSET] etc.             – Data set declarations
│ [FMTCTL] etc.           – Format control I/O declaration
│ [START] etc.            – Start point declaration
│ [REENTER] etc.          – Reenter point declaration
└ STACK etc.              – Stack declaration
┌ BLK or DBLK             – Begin block or begin dummy block
│                           declaration
│   BIN etc.   ┐
│   BINI etc.  │
│   BCD etc.   │
│   BCDI etc.  ├.....     – Data item declaration(s)
│   BOOL etc.  │
│   STRG etc.  │
└   STRGI etc. ┘
```

1.3.2 *Terminal Class Declaration*

The TERM declaration identifies the terminal class with a unique two character task identifier. It is followed immediately by the relevant work block declaration(s), start point directive, data set declaration(s) and stack declaration.

1.3.3 *Work Block Declarations*

There may be a maximum of 15 work block declarations in each terminal class. These declarations refer to the block identifiers specified in the begin block declarations (BLK or DBLK).

Several work block declarations may refer to a single block identifier. This implies that these work blocks are defined under a single begin block declaration. For terminal work blocks, user work blocks and swappable work blocks this facility is merely a form of programming shorthand. It does not mean that these work blocks are "overlayed" or "common". However, the facility saves the effort of writing out identical sets of declarations a number of times. For common work blocks, more than one reference to a single block identifier results in the generation of a single work block which can be used by several tasks.
Dummy work blocks are used to redefine a terminal-, common-, user-, or swappable work block in the same terminal class.
If a block identifier is used in more than one work block declaration, each declaration must always occupy the same position relative to the other work block declarations in the same terminal class. See the Declaration Reference Section (1.3.9) for an illustration of this rule.

1.3.4 *DSET, FMTCTL, START, REENTER and STACK*

The DSET declaration specifies those data sets which are to be used in this terminal class. The declaration is optional.
The FMTCTL declaration indicates the format control I/O feature will be used in this terminal class.
The START directive indicates the start point for the task(s) associated with this terminal class.
The REENTER directive can be used with memory management, to refer to a closing routine which will be executed when it is impossible to read a code segment in main memory or from disk.
The STACK declaration specifies the size of the stack to be allocated to this terminal class. The declaration is optional.

1.3.5 *Begin Block Declaration*

1.3.5.1 *General*

The BLK or DBLK declaration identifies the work block defined by the data item and array declarations which follow.
Work blocks are used to define areas of memory which may be used for input/output buffers and/or work locations. Dummy work blocks do not occupy memory locations but are redefining existing work blocks in the same terminal class.
The instruction of a CREDIT program are held in as a single memory copy during execution. However, the number of copies of work blocks which exist in memory depends upon the configuration of the PTS System used by the application. The number of work blocks needed depends upon the number of work positions and upon the number of users who will operate these work positions. The work blocks are actually generated when the application program is loaded into core.
They are generated by a part of TOSS System Software, the SYStem LOADing program (SYSLOD). The use of SYSLOD is discussed in part two of this manual.

1.3.5.1 *Terminal Work Blocks*

One or more terminal work blocks (TWB) may be defined for each terminal class within a program. A separate copy of the terminal work blocks will exist in memory for each task activated for a particular terminal class. Each copy may be used only by its associated task and cannot be used for exchanging data with other tasks.

1.3.5.3 [illegible]

One or [illegible] blocks [illegible] may be defined for each terminal class with a [illegible] these common work blocks is generated. They may be referenced by all [illegible] the terminal class. If the same block identifier is used in the work block [illegible] than one terminal class, then the tasks belonging to these terminal [illegible] the same common work block.

1.3.5.4 [illegible]

In this [illegible] position operator. It may be necessary for the application [illegible] information, e.g. cash accumulators, for each user. However [illegible] one-to-one relationship between work positions and users [illegible] number of users as work positions. The users may rotate [illegible]
To maintain [illegible] areas of memory are required which are associated with individual [illegible] positions. These areas of memory are called user work [illegible]
One or more [illegible] with different block-identifier, may be defined for each terminal [illegible]
Tasks [illegible] work block if they belong to a terminal class which contains [illegible] user work blocks. Furthermore, a task may only refer to a user work [illegible] USE instruction specifying the block identifier and index identifier [illegible] block. An index identifier is a reference to a decimal integer from [illegible] used to differentiate between user work blocks of the same format. [illegible] work block type may be connected to one task.
It is possible [illegible] the same user workblock type at the same time.
This [illegible] with care. It is the responsibility of the application program [illegible] which may result from this type of access.

1.3.5.5 [illegible]

A dummy [illegible] work block referenced by the block identifier, between the [illegible] declaration. [illegible]nition starts at the beginning of the referenced [illegible] terminal class, and continues sequentially.
Redefinition [illegible] type of data items but is restricted to the size of the defined [illegible]

Example:

```
        [illegible]
        [illegible]
TB1     [illegible]
B1      [illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

```
DB1       DBLK
DU1       BCD     8           redefines BN1 and BN2
STRU1     STRG    11          redefines D1, D2 and ST1
```

When data item DU1 is processed the memory locations for BN1 and BN2 are used as decimal-data-item. The user must be aware of this situation when data-item BN1 or BN2 is processed afterwards, while it is to be processed as binary data item. Dummy work blocks may contain data-items of type BOOL, BIN, BCD, STRG, BINI, BCDI or STRGI. No initial values may be assigned to these data-items as no memory space is allocated. For data-items of type decimal and string, the length of each item must be declared explicitly.

1.3.5.6 *Swappable Work Blocks (SWB)*

Swappable work blocks are disk resident and in use similar to user work blocks. A swappable work block type, different [illegible]identifier, is read into memory only when it is attached to a task with the USE instruction.
One copy of a swappable work block type may be present in main memory at the same time and is allocated to one task as long as it stays in main memory. However, they can be used by a multiple of tasks but not simultaneously. One area in main memory is reserved for each swappable work block type per task.
The swappable work block is set under exclusive access on the diskfile $SWAP. Exclusive access is released when the instruction UNUSE is executed for that particular work block type. Data which is not frequently used can be stored in these work blocks. A swappable work block is rewritten on disk after it is detached from the task, by executing the UNUSE instruction. The work blocks are copies onto a disk file $SWAP, at system loading time by the system loading program (SYSLOD).
File $SWAP must be a standard file (S type) with record length 400 and blocking factor 1.

1.3.6 *Data Item and Array Declarations*

1.3.6.1 *General*

These declarations describe the data items and arrays of which the work blocks are composed.

Data items may be binary coded decimal (BCD), binary (BIN), boolean (BOOL) or string (STRG). The mnemonics BCD, BIN, BOOL and STRG are known as "data item types". Arrays may be binary coded decimal (BCDI), binary (BINI) or String (STRGI). The mnemonics BCDI, BINI and STRGI are known as "array types". These types should be distinguished from the "value types" described below.

Any BIN or BINI declarations must appear at the start of the work block.

1.3.6.2 *Data Item Declarations*

A data item declaration has the following general format:

$$\text{identifier}␣\left\{\begin{matrix}\text{BCD}\\ \text{STRG}\end{matrix}\right\}␣\quad \text{data-item-specification}$$

$$\text{identifier}␣\ \text{BIN}\quad ␣\quad [\text{data-item-specification}]$$

$$\text{identifier}␣\ \text{BOOL}␣\quad \left[\left\{\begin{matrix}\text{TRUE}\\ \text{T}\\ \text{FALSE}\\ \text{F}\end{matrix}\right\}\right]$$

Data-item-specification specifies the length, value type and value of the data item. It has the following general format in both data item declarations and array declarations:

$$\left\{\begin{array}{l}\text{length [[value-type] ['value']]}\\ \text{value-type ['value']}\\ \text{'value'}\end{array}\right\}$$

where:

Length is a decimal integer indicating the length of the item. It has the following significance:

BCD and BCDI — Length indicates the number of (4 bit) digits in the data item.

BIN and BINI — Length need not be used. If value-type is W, length may only be 1. If value-type is X or D, length indicates the number of (4 bit) digits in the data item. If value-type is C, length indicates the number of (8 bit) ISO-7 characters in the data item.

Value-type is one of the characters D, W, X or C, and specifies whether value is to be stored in memory as decimal (D), binary (W), hexadecimal (X) or string (C). The following combinations of data item type and value-type are allowed:

BCD and BCDI	—	D or X
BIN and BINI	—	C, D, W or X
STRG and STRGI	—	C or X

If value-type is not specified, the following defaults are assumed:

BCD and BCDI	—	D
BIN and BINI	—	W
STRG and STRGI	—	C

Value specifies the value to be assigned to the data item. If value is longer than the specified length will allow, then:

BCD, BCDI, BIN and BINI— value is truncated at the left. The least significant digits are placed in the data item.

STRG and STRGI — value is truncated at the right. The leftmost characters in value are placed in the data item.

If value is shorter than the specified length will allow, then:

BCD, BCDI, BIN and BINI — value is right adjusted and zero filled on the left.

STRG and STRGI — value is left adjusted and the data item is filled on the right by repeating the rightmost character of value.

If value is not specified, then:

BCD, BCDI, BIN and BINI – value is assumed to be zero.

STRG and STRGI — value is assumed to be ISO-7 spaces.

Value must be written as a decimal number, a hexadecimal integer or a character string, depending upon value-type. The rules are as follows:

D	—	decimal number
X	—	hexadecimal integer
W	—	decimal number
C	—	character string

Value will be stored in the data item in the following way:
BCD and BCDI – The digits in value are placed directly in the data item without conversion. Each digit occupies four bits.
BIN and BINI – If value-type is W, value is converted from decimal to binary representation and then stored. If value-type is X or D, value is placed in the data item without conversion. Each digit occupies four bits. If value-type is C, the ISO-7 hexadecimal representation of the characters is placed in the data item. One character occupies eight bits
STRG and STRGI – If value-type is C, the ISO-7 hexadecimal representation of the character is placed in the data item. One character occupies eight bits. If value-type is X, value is placed directly in data item without conversion. Each digit occupies four bits.

If length is omitted, the length of the data item is implied by the specified value-type and value.

Binary (BIN) data items have a fixed length of one word. For this type of data item length, value-type and value may all be omitted.

1.3.6.3 *Array Declarations*

An array declaration has the following general formats.

$$\text{array-identifier} \sqcup \left\{\begin{matrix}\text{BCDI}\\ \text{STRGI}\end{matrix}\right\} \sqcup \begin{matrix}\text{(dimension [,dimension]),}\\ \text{data-item-specification [, 'value'] ....}\end{matrix}$$

$$\text{array-identifier} \sqcup \text{BINI} \sqcup \begin{matrix}\text{(dimension [,dimension])}\\ \text{[,data-item-specification] [,'value'] ....}\end{matrix}$$

The use of the data-item specification has been described above. If there are fewer "values" in data-item-specification than there are data item occurrences in the array, the last "value" specified is repeated in the remaining data item occurrences.

If a value is defined, longer than the value defined by the first list element, it is truncated according to the rules defined in 1.3.6.2.

"dimension" (or the product of the dimensions if there are two) indicates the number of occurrences of the data item in the array. Each dimension must be a decimal integer. A maximum of two dimensions is allowed.

When setting an index to refer to an array, the first occurrence in each dimension is always counted as one. For a two dimensional array both indices must lie in the range 1 to 255. A one dimensional array is not restricted in this way.

An initial value may be assigned to an array. This is done by listing, in the array declaration, the values to be assigned to each occurrence. Each value must appear in quotes and must be separated from the next value by a comma. For example:

```
TAB⊔BINI (4,4),  '1', '2', '3', '4', '5', '6', '7', '8', '9', X'A', X'B', X'C', X'D', X'E',
                 X'F', X'0'
```

The Translator will set these values up in the array row by row from left to right, as follows:

```
0001      0002      0003      0004
0005      0006      0007      0008
0009      000A      000B      000C
000D      000E      000F      0000
```

The array will be referenced in the following manner:

TAB (X, Y) has the value 2 (where contents of binary-data-items, X = 1 and Y = 2)

TAB (X, Y) has the value X 'A' (where contents of binary-data-items, X = 3 and Y = 2)

1.3.7 *Data Items*

A data item is the most elementary unit of data that can be used as an operand in a CREDIT instruction. Data items are defined by data item or array declarations and are refered to by their data item identifier or array identifier and dimension(s):

Type of declaration	Type of reference
data-item-declaration	data-item-identifier
array-declaration	array-identifier (index-identifier-1 [,index-identifier-2])

In this Manual data items are referred to as "decimal data items", "string data items" etc. **These data items may be defined by *either* data items *or* array declarations (except for boolean):**

Type of declaration	Type of data item
binary-data-item-declaration binary-array-declaration	binary-data-item
boolean-data-item-declaration	boolean-data-item
decimal-data-item-declaration decimal-array-declaration	decimal-data-item
string-data-item-declaration string-array-declaration	string-data-item

Data items which have a special significance in a group of instructions have been given special names. There are three data items in this category: index, pointer and size. These names are used in order to simplify the descriptions of these groups of instructions. However, the data items are held in memory and operated upon in exactly the same way as other data items.

An index is used in the IB and PERFI instructions.
It is used in these instructions to select a particular statement identifier or external identifier from a specified identifier list. Index must always be defined as a binary data item.

A pointer is used in the COPY, DLETE, EDIT, INSRT, MATCH and XCOPY instructions. It is used in these instructions to point to an individual character in a data item. Pointer must always be defined as a binary data item.

A size is used in the COPY, DLETE, INSRT, KI, MATCH, NKI, WRITE, IREAD, IWRITE, RREAD, RWRITE and READ instructions. It is used in these instructions to indicate the size of the data item segment upon which the instruction is operating. Size must always be defined as a binary data item.

1.3.8 *Work Blocks*

For each work block the translator generates a 16 or 256 entry pointer table maximum, depending on the value assigned to ADRMOD in the OPTNS directive. The entries in each of these tables point to the non-boolean (i.e. binary, decimal or string) data items and arrays in the associated work block. Each non-boolean data item in a work block occupies one entry in the pointer table. Each array occupies two entries. The number of non-boolean data items and array declarations in each work block must be such that the number of entries in the pointer table does not exceed 16 or 256.

The following combination is possible (ADRMOD=1):

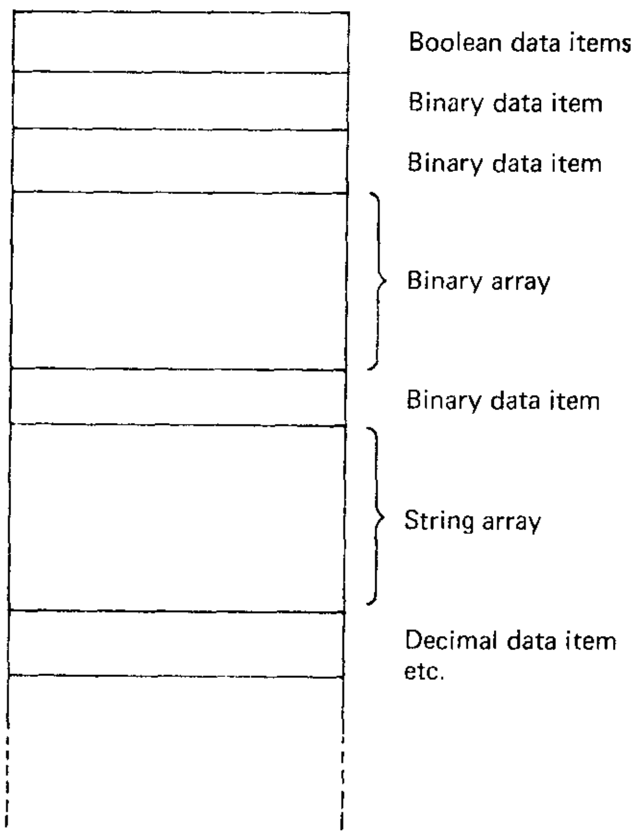
4 data item declarations	– requiring	4 entries
6 array declarations	– requiring	<u>12</u> entries
Total		<u>16</u> entries

The pointer table is not used for boolean data items. This is because all boolean data items in a work block are stored in the first word of that block. The maximum number of boolean data items in a work block is therefore 16.

Memory for non-boolean data items and arrays is allocated in the same sequence as the corresponding data item and array declarations appear in the CREDIT listing.

An example work block layout is shown below:

Sequence of data items and arrays is the same as sequence of declarations except for boolean data items which are all in the first word. Binary declarations are always made at the start of a work block.



The last data item declared in a work block may not have a start from a displacement higher than 32767.

1.3.9 *Declaration Reference*

This section describes the syntax and use of each declaration. The possible values of the variables in declarations is given in appendix 1. The notation conventions are described in section 1.1.5.

BCD *Decimal data item* BCD

Syntax:	data-item-identifier ␣ BCD ␣ data-item-specification
Description:	Decimal data items are of variable length, varying from 1 to 510 decimal digits (sign included) and occupying a whole number of bytes.
Example:	

```
                                    memory representation:
R       BCD       4'–9'                X'D009'
S       BCD       X'BFF40'             X'BFF400'
```

The sign is the most significant tetrad: D = negative, B = positive

The character 'F' in the above example denotes a "BCD space". This character is treated as zero in arithmetic operations. However, if the data item is edited and printed, BCD spaces will appear as blanks. Under certain circumstances BCD spaces are generated by the MOVE instruction.

BCDI *Decimal array* BCDI

Syntax: array-identifier ␣BCDI␣ (dimension [,dimension]),
data-item-specification [,'value'] · · ·

Description: Within the array all elements have a fixed size. The size for decimal array elements may vary from 1 to 510 digits (sign included). The size of an element is derived from the first element in the list. A one dimensional array may contain maximum 32767 elements. A two dimensional array may vary from 1 to 255 elements per row and 1 to 255 elements per column.
(Maximum 255*225 elements)
When an array is partly initialized, the last defined element will be copied until all remaining elements are filled.

Example:

```
TAB1      BCDI    (25),6D'–94278'
TAB2      BCDI    (3), '1','2','3'
TAB3      BCDI    (2, 5), 6D'0'
```

BIN — *Binary data item* — BIN

Syntax: data-item-identifier ␣ BIN ␣ [data-item-specification]

Description: Binary data items are allocated full words and thus have a fixed length of 16 bits. All binary data items within a block must be declared before the decimal and string data items and arrays.
The data item value must be within the range −32768 to 32767.

Example:

```
                                   memory representation:
A        BIN        W'20'                X '0014'
B        BIN        X'FF'                X '00FF'
C        BIN        3D'100'              X 'B100'
D        BIN        2C'NO                X '4E4F'
E        BIN                             X '0000'
```

BINI *Binary array* **BINI**

Syntax:	array-identifier ␣BINI␣ (dimension [,dimension]) [,data-item-specification] [,'value'] · · ·
Description:	Binary array elements are allocated full words and each element has a fixed length of 16 bits. The value of an element in the array must be within the range –32768 to 32767. A one dimensional array may contain maximum 32767 elements. A two dimensional array can vary from 1 to 255 elements per row and 1 to 255 elements per column. (Maximum 255*255 elements). When an array is partly initialized, the last defined element will be copied until all remaining elements are filled.
Example:	TAB1 BINI (100),'0' TAB2 BINI (5), '1', '2', '3', '4', '5'

BLK	*Begin block*	BLK

Syntax: block-identifier ␣ BLK

Description: Defines the beginning of a work block.
The three leading characters of the block-identifier should be unique since the identifier is truncated if longer. The block-identifier has to correspond with the one specified in the work block directive (TWB, CWB, UWB or SWB).
The begin block declaration must be followed by at least one data item declaration or array declaration.

BOOL *Boolean data item* **BOOL**

Syntax:

$$\text{data-item-identifier} \sqcup \text{BOOL} \sqcup \left[\left\{\begin{array}{l}\text{TRUE}\\ \text{T}\\ \text{FALSE}\\ \text{F}\end{array}\right\}\right]$$

Description: The length of a boolean variable is always one bit. A value may be declared by selecting one of the following:-
TRUE, T, FALSE or F.
TRUE or T corresponds with "1" and FALSE or F corresponds with "0". The default value is always FALSE (zero).

Boolean variables may not be indexed.

Example:

```
FLAG          BOOL          TRUE
```

CWB *Common work block* **CWB**

Syntax: ␣ CWB ␣ block-identifier

Description: The block-identifier refers to a begin block declaration (BLK). The three leading characters of the block identifier must be unique. The identifier is truncated if longer.
The same block-identifier may be used in different terminal classes. If this is done the common work block declaration must always occupy the same position relative to the other work blocks in the same terminal class.
A common work block is only common for those terminal classes in which it is defined.

Example:

Legal:

```
TERM A0
CWB A ◄─┐
TWB B   │
UWB C   │
TERM B0 │
CWB A ◄─┘
TWB D
UWB E
```

Illegal:

```
TERM A0
CWB A ◄─┐
TWB B   │
UWB C   │
TERM B0 │
TWB D   │
UWB E   │
CWB A ◄─┘
```

DBLK *Begin dummy block* DBLK

Syntax: Block-identifier ␣ DBLK

Description: Defines the beginning of a dummy work block. The three leading characters of the block-identifier should be unique since the identifier is truncated if longer. The block-identifier has to correspond with the one defined in the DWB directive. (block-identifier-1).
This begin dummy block declaration must be followed by at least one data item declaration or array declaration. It serves to declare a redefinition of a work block.

DSET — *Data Set* — DSET

Syntax:

```
data-set-identifier ␣ DSET ␣ FC = file-code
                             [, BUFL = decimal-integer]
                             [, DEV = device-type]
                             [, BUFDS = data-set-identifier]
```

Description:

A data set is an I/O device. Particular data sets may only be accessed from a task if the task belongs to a terminal class containing a DSET declaration for that data set.
The same dataset declaration may be used in different terminal classes. If this is done, the dataset declaration must always occupy the same position relative to the other dataset declarations in all terminal classes in which it is used.
The keyword parameters FC, BUFL and DEV may be written in any sequence. They have the following significance:

BUFL	Buffer length. If this parameter is present a fixed buffer is allocated. The parameter value is a decimal integer giving the length in characters.
BUFDS	Buffer data set. If this parameter is present, the preceding buffer length (BUFL) will be shared with the data set buffer indicated by the data-set-idientifier after BUFDS. The buffer size must be smaller than the shared buffer.
DEV	Device type. Parameter value consists of two letters (see following device type list).
FC	TOSS file code. Parameter value consists of two hexadecimal digits (see following file code list).

Any unrecognized keywords are ignored.

Example:

```
PRT      DSET      FC = 40, DEV = LP, BUFL = 120
OWNER    DSET      FC = 32, DEV = TV, BUFL = 100
SHARE    DSET      FC = 31, DEV = TR, BUFL = 36,
                                  BUFDS = OWNER
```

TOSS Device type:	Meaning:
CR	Card Reader
DC	Data Communication
DG	Graphic display
DI	Indicator display
DL	Logical disk file
DN	Numeric display
DY	Alphanumeric display (VDU)
GP	General printer

DSET *Continued* DSET

TOSS Device type:	Meaning:
II	Intertask input
IO	Intertask output
KA	Alphanumeric keyboard
KI	Keyboard indicators
KN	Numeric keyboard
LP	Line Printer
MT	Magnetic Tape (1/2 inch)
SI	System Operator's panel Switches
SO	System Operator's panel lamps
TK	Cassette Tape
TJ	Teller terminal printer : Journal print station
TR	Teller terminal printer : Tally roll print station
TV	Teller terminal printer : Voucher print station
TW	Typewriter

Recommended TOSS File codes : (hexadecimal)	Meaning:
10	System operator panel-in.
11	System operator panel-out.
12	Cassette recorder nr. 1
13	Cassette recorder nr. 2
15	Remote line test
20	Keyboard
25	Reserved for future use.
30	General Printer.
30, 31, 32	Teller printer (TJ, TV, TR)
40	Signal display
41	Numeric display
50	Character display
60	Data communication
70	Magnetic tape
80	Line printer
90	Card reader
A0, B0–B3, B6	Reserved for future use
C0–CF	Data management disk files
D0	Inter task communication-input
D1	Inter task communication-output
F0–FB	Reserved for system
FC–FF	Reserved for user

DWB *Dummy work block* DWB

Syntax: ␣ DWB ␣ block-identifier-1(block-identifier-2)

Description: Block-identifier-1 refers to the begin dummy block declaration (DBLK), where block-identifier-2 refers to a work block declared in the same terminal class, which will be redefined.
The three leading characters of both block identifiers must be unique. The identifiers are truncated if longer. The same block-identifier-1 may be used in different terminal classes. If this is done the dummy work block declaration must always occupy the same position relative to the other work blocks in the same terminal class.

Example 1:

```
        TWB     TB1
        DWB     DB1 (TB1)
         .
         .
TB1     BLK
         .
         .
DB1     DBLK
         .
         .
```

Example 2:

```
        Legal
   TERM     A0
   TWB      TB1
   TWB      TB2
   DWB      DB2 (TB1)  ◄─┐
    .                    │
    .                    │
   TERM     B0           │
   TWB      TB1          │
   TWB      TB4          │
   DWB      DB2 (TB1)  ◄─┘
        Illegal
   TERM     A0
   TWB      TB1
   TWB      TB2
   DWB      DB2 (TB1)
    .
    .
   TERM     B0
   TWB      TB1
   DWB      DB2 (TB1)
   TWB      TB3
```

FMTCTL *Format control I/O* **FMTCTL**

Syntax:

```
␣ FMTCTL␣ { INDS=data-set-identifier,
                   OUTDS=data-set-identifier
            OUTDS=data-set-identifier,
                   INDS=data-set-identifier }
```

Description: A format control I/O declaration, must follow the data set declarations immediately.
It specifies that this terminal class will use the format control I/O feature.
Data-set-identifier after INDS specifies the input device and after OUTDS the output device, used for format controlled I/O, FMTCTL and referenced data sets must be in the same terminal class.

Example:

```
DSKB    DSET      FC = 20, DEV = KB
DSDY    DSET      FC = 50, DEV = DY, BUFL = 90
DSGP    DSET      FC = 30, DEV = GP, BUFL = 90
        FMTCTL    INDS = DSKB, OUTDS = DSDY
```

STACK *Stack* **STACK**

Syntax: ␣ STACK ␣ size

Description: Allocates a user memory stack.
Size is a decimal integer indicating the size of the stack in bytes. If the stack declaration is omitted a default stack size of 128 bytes is allocated. This declaration should occur after the START or REENTER declaration.

STRG — *String data item* — STRG

Syntax: data-item-identifier ␣ STRG ␣ data-item-specification

Description: Strings are of variable length and occupy a whole number of characters. The length varies from 1 to 4095 characters.

Example:

			memory representation:
ALL	STRG	7'HEADING'	X'48454144494E47'
BTA	STRG	8X'42455441'	X'42455441'
BET	STRG	4'BETA'	X'42455441'
F1	STRG	5C'AB'	X'4142424242'
F2	STRG	5C'ABCDEF'	X'4142434445'

STRGI *String array* **STRGI**

Syntax: array-identifier ␣ STRGI ␣ (dimension [,dimension]) ,data-item-specification [,'value'] ....

Description: Within an array all elements have a fixed size. The size for a string array element may vary from 1 to 4095 bytes. The size of an element is derived from the first element in the list. A one dimensional array may contain maximum 32767 elements. A two dimensional array may vary from 1 to 255 elements per row and 1 to 255 elements per column. (Maximum 255*255 elements). When an array is partly initialized, the last element will be copied until all remaining elements are filled.

Example:

```
TAB1        STRGI       (10, 10), 32C' '
TAB2        STRGI       (10).40C
```

SWB *Swappable Workblock* SWB

Syntac: ␣ SWB ␣ block-identifier.

Description: The block-identifier refers to a begin block declaration (BLK). The three leading characters of the block-identifier must be unique. The identifier is truncated if longer.
The same block-identifier may be used in different terminal classes. If this is done the swappable workblock declaration must always occupy the same position relative to the other workblocks in the same terminal class.
A task can only access a swappable work block after the USE instruction has been executed by that task. It can be detached from the task by executing the UNUSE instruction.

Example:

```
Legal:                      Illegal:
TERM  A0                    TERM  A0
CWB   A                     CWB   A
TWB   B                     TWB   B
UWB   C                     UWB   C
SWB   D  ◄──┐               SWB   D  ◄──┐
            │                           │
TERM  B0    │               TERM  B0    │
CWB   E     │               CWB   E     │
TWB   F     │               TWB   F     │
TWB   G     │               SWB   D  ◄──┘
SWB   D  ◄──┘               TWB   H
```

TERM *Terminal class* **TERM**

Syntax: ␣ TERM ␣ task-identifier

Description: This declaration followed by a maximum of 15 work block declarations describes a class of terminal.
The task identifier must consist of a letter followed by a digit or a letter.

Note: Task identifier is the same as the task identifier specified at system loading time. (SYSLOD)

Terminal work block

Syntax: ␣ TWB ␣ block-identifier

Description: The block-identifier refers to a begin block declaration (BLK). The three leading characters of the block identifier must be unique. The identifier is truncated if longer.
The same block-identifier may be used in different terminal classes. If this is done the terminal work block declaration must always occupy the same position relative to the other work blocks in the same terminal class.

Example:

Legal:

```
TERM A0
CWB A
TWB B  ◄─┐
UWB C    │
         │
TERM B0  │
CWB D    │
TWB B  ◄─┘
UWB E
```

Illegal:

```
TERM A0
CWB A
TWB B  ◄─┐
UWB C    │
         │
TERM B0  │
CWB D    │
UWB E    │
TWB B  ◄─┘
```

UWB *User work block* **UWB**

Syntax: ␣ UWB ␣ block-identifier

Description: The block-identifier refers to a begin block declaration (BLK). The three leading characters of the block identifier must be unique. The identifier is truncated if longer.
The same block-identifier may be used in different terminal classes. If this is done the user work block declaration must always occupy the same position relative to the other work blocks in the same terminal class.
A task may only access a user work block after a USE instruction has been executed by that task. It can be detached from the current task by executing the UNUSE instruction.

Example:

```
Legal:                                   Illegal:
TERM A0                                  TERM A0
CWB A                                    CWB A
TWB B                                    TWB B
UWB C  ◄─┐                               UWB C  ◄─┐
         │                                        │
TERM B0  │                               TERM B0  │
CWB D    │                               CWB D    │
TWB E    │                               UWB C  ◄─┘
UWB C  ◄─┘                               TWB E
```