CPG2 Programmer's Manual

Contents 1000

Lattwein information 1010

Address:	**Lattwein GmbH** **Otto Brenner Strasse 25** **52353 Dueren** **Germany**
Telephone:	**+49 2421 81051**
Telefax:	**+49 2421 82127**
Internet:	**http://www.lattwein.de**
E-Mail:	**mailto:service@lattwein.de**

Information:

A central telephone information-service is available to all users during normal working-hours, which is intent on answering any questions, which have not been dealt with properly within this manual.

Working time:	**8.30 - 16.00 Hrs**

Section 2 Service Functions 2000

Data Fields 2100

Field Names and Field Definitions 2110

An area in the Transaction Work Area (TWA) is assigned to each field in a CPG program. A field can be defined explicitly or implicitly.

The explicit definition of fields is made in the data division.

If in the input division is described, from which positions of a data device a field should be read, an implicit field definition takes place at the same time.

All fields stored in the TWA are either alphanumerical or packed numerical. Always if a field is defined, CPG2 requires the following statements:

a) field name
b) field length
c) form (numerical or alphanumerical)
d) case numerical: number of decimal positions.

Field names can be up to 30 positions long. They must not contain empty signs (blanks). The first sign must be a letter from A until Z.

Regulations for field names:

Without using Data Dictionary

It is recommended, to use field names with more than 6 positions only exceptionally, because the compiler generates an internal name with more than 6 positions. Consequently the entry of data dictionary structures and program external QSF masks becomes difficult with the use of long field names.

Using Data Dictionary

With the transaction QDDS (Quick Data Dictionary Standards) it is possible to use field names with a length bigger than 6.

If a field name is described here, it gets an internal reference name with a length up to 6 places.

Once described in the standards, the long field name can be used almost everywhere in the application development with CPG, especially in QSF-screens and List documents.

The $-(dollar-) sign is allowed at every position in the field name.

The sign sequence 'CPG' is reserved for the first three positions of a field name. Its application is only allowed with the CPG-internal fields, indicated in section 2130.

Note for the array processing, that a limitation is also valid for the representation of array elements: The indication consists from the name of the array and the index in brackets, and may be at most 7 positions long.

The maximum length of an alphanumeric field is 256 bytes.
The maximum length of a numerical field is 8 bytes packed, that means 15 digits.

All fields are alphanumerical, if nothing else is indicated. Fields will be assigned as numerical via the indication, that they possess 0 or more digits.

- **`OTTO 5.`** means that the field OTTO is alphanumerical and 5 positions long.
- **`HUGO 3 0.`** means that the field HUGO is numerical, 3 positions long and has zero digits.

Fields and arrays can be combined to groups or records via data structures and overlays in the TWA. If a field, which was not yet defined in the data division, will be specified in the input division, it is implicitly defined. CPG calculates the length of the field on the base of the indicated first and last position, and orders the corresponding storage area in the TWA. If a certain number of decimal places is specified for the field, (i.e.0 or more), it is treated by CPG as packed field. If the input data for a field will be indicated as binary or in the unpacked form, so CPG changes them automatically into the packed form, before it is stored into the corresponding field in the TWA.

In data structures, numerical fields are only allowed in packed form.

If a field has already been defined in the CPG program, with the same name explicitly or implicitly with another field length, so a syntax error will be detected at the compilation time. If the field should be numerical, a value of 0 or bigger must be entered for the number of decimal places. If the attributes defined for the field are not compatible with the operation with which it is used, an error is also indicated.

Note the following instructions for field definitions:

All fields must be defined explicitly or implicitly somewhere in the program, in which they are used. (With exception of the CPG-internal fields and UDATE and UTIME).

A field cannot be used in a manner, that contradicts to its defined attributes.

Regulations for field names:

Field names may be at most 30 positions long.The first sign of a field name must be an alpha (sign) or a dollar sign. Field names can contain any special characters up from the second place. Field names, that contain special characters (beside dollar), and/or arelonger than 6 positions, are changed into names of six places length (CPxxxx).

Examples:

THIS-IS-A-VALID-FIELD-NAME	right
THIS-IS-NOT-A-VALID-FIELD-NAME-FOR-CPG	wrong (too long)
A*27+35=89:IS-NOT-GOOD:BUT	right
7-MILES-BOOTS	wrong (position 1 not alpha)
$$10	right
A-100	right

Arrays 2120

Arrays are groups of fields of same the length and the same type, that can be processed indicately for example via a DO-loop.

Array definition

Arrays can only be defined in the data division in the form: Name, character of elements, *, length, digits.

Alphanumerical arrays will be preformatted with blanks, numerical ones with zeros.
In a CPG2 program a maximum of 80 arrays are supported.

Array Input / Output

Whole arrays or single fields with fixed index can be read or output. Variable indexes are not supported in the input- and output division.

Array and Screen

Arrays will be processed like single fields in the output description. For the output on screen files, the position of the first array element will be indicated. The following elements are indicated automatically in the following screen lines. The screen has to be read analogously.

Array and record files / print output / field editing with EDIT - SELCT

For other files, for the output, always the last position of the last array element will be indicated to read the area from the first up to the last byte of the array.

If whole arrays are output with an edit code, so the single fields are separated by two blanks (plus the length of the edited field).

Regulations for array names:

For arrays processed as indicated, following regulations are valid:

The length of the array name + the length of the index name must be less than 6.

Examples:

A(I)	right
A(ZAHL)	right
FG(88)	right

or:

The index name is one digit, the array name up to 27 positions long (a CP name will be generated for the lengths 7 to 27 as well as for shorter array names, that contain special characters).

Examples:

MYARRAY(A)	right
AR-2(K)	right
MYARRAY(I2)	wrong, index not one digit

Processing arrays

Following operations of the procedure division can be processed as indicated:

all numerical operations

CAB	compare and branch
CAS	compare and branch subroutine
CBS	compare and branch subroutine
COMP	compare
CONVT	convert
DELC	delete characters
DO	loops
EDIT	field edit
ELIM	eliminate characters and replace by blanks
FILL	fill an alpha field
IF	if query
MOVE	transfer right adjusted
MOVEA	array transfer
MOVEL	transfer left adjusted
MOVEN	transfer from an alpha field into a numerical field
REPLC	replace blanks by characters
SCAN	search for a scanner
SELCT	select a field

That means, in a valid operand of one of these operations, the name of an array can be indicated instead of a field name. Following regulations are valid:

Array with fixed index

If an array with the name 'FGR' has been defined, an operation

```
- FGR(7) = X
```

causes that the value 'X' is added in the 7. element of the array 'FGR'. Note, that the whole length of the entry must not be bigger than 7 positions. In this case, the name of the array can have at most 4 positions.

Array with variable index

If an array was defined with the name 'FGR', an operation

```
-  FGR(N) = X
```

causes that the value X is transferred into the N-th. element of the array 'FGR', with which the field with the name 'N' must contain an integer value between 1 and the number of elements of the array. Each other value leads to the abnormal end of the program. The total length of the entry for the element name must not be longer than 7 positions.

Array without index

If an array was defined with the name 'FGR', so an operation

```
-  FGR = 0;
```

causes that all elements of this array are set to 0. If a second array with the name 'FG2' was defined, so the operation

- **`FG2 = FGR`**

causes that all elements of the array 'FGR' will be transferred into the corresponding elements of the array 'FG2', if the character of the elements is equal for both arrays. Is one of the two arrays smaller than the other, then only as much elements are transferred, as have been defined for the smaller array.

Note:

In the described form, only numerical values can be transferred between numerically defined fields and arrays. For the transfer of alphanumeric values, different Move operations are disposable.

CPG internal Fields 2130

CPG internal fields are already defined by the compiler and must not be defined by the programmer. Following fields are disposable:

CPGATR variable attribute (if QSF is not used)

Definition for screen outputs with the entry 'VAR'. The wished attribute will be entered into this one place field.

CPGBZL indicator list

CPGBZL is a 100 place field, that contains the status of the indicators from 00 to 99. It can be used for example to save the indicators before an exit to a subroutine.

CPGCOM Common Area.

This area can be used as temporary storage for program links and is filled and read with EDIT- and SELCT operations.

CPGCSA Common System Area.

This area can be changed or queried via EDIT- and SELCT operations. The data in the CPGCSA is available for all system users independent from the program or terminal. The length of a field corresponds to the length of the Common Work Area, defined in the System Initialisation Table (DFHSIT). The maximum length amounts to 3584 positions.

CPGCUR Variable cursor position (if QSF is not used)

Definition for screen outputs with the entry 'VAR'. The wished cursor position is entered into this 4-digit field.

CPGDID Variable printer name

The name of the printer is entered into this 4-digit field. If a variable printer was declared in the files division.

CPGDRC DL/I return code.

This area contains the DL/I return code after a DL/I call.

CPGEOJ End of job.

This four-digit numerical field enables to transfer a variable return code out of the batch program to VSE or OS/390 that can be queried in the conditional job control.

CPGFIS Offset for file inputs.

This 5-digit numerical field contains the offset, if at the input record description was arranged, that the input positions may be used variably. The content of the field CPGFIS will be added in this case to the indicated input positions. (See example in chapter 8000)

CPGFRC File return code after file operations

For the programming of file operations, no switches are necessary. After the file operations, CPGFRC contains the following codes which can be queried instead of switches:

DK	Duplicate Key after READ on an AIX-file
DR	Duplicate record after adding
EF	End of File after READ, READ-BACK and SETLL
NC	Not closed after CLOSE
NF	Not found after CHAIN or CHECK, OPEN, CLOSE,
NO	Not open after OPEN or CHECK
Blank	if no peculiarity was determined. (No errors on last file operation)

CPGIFC Interface communication area

This 32 places field can store information for the CICS interfaces of the Central Routine Library. CPGIFC can be processed with the operations EDIT and SELECT.

In connection with the operation IFC, the programmer himself can modify operations in the CICS interface.

CPGMCI Array index for cursor position (in a QSF map).

This 3-digit numerical field contains the index of the array, whose name can be found in the field CPGMCU.

CPGMCU Cursor position (in a QSF map)

This 6 digit alpha field contains the name of the field in which the cursor should be set. If this field is empty, the cursor will be set into the field defined in QSF.

CPGMFI Array index for selector pen/screen input.

This 3 digit numerical field contains the index of an array element, that has been chosen with selector pen, or in which the cursor stood during the screen input. (if you use CPG2..QSF).

CPGMFN 1. Field name for selector pen/screen input.

This field contains the name of the chosen field after the selector pen selection, as well as the name of the field, in which the cursor stood during the screen input (if you use CPG2..QSF).

2. Restriction of the map output on particular lines.

$QV' can be set into this field and CPGMLC can be filled before a map output, like described below.

CPGMLC 1. Field position for selector pen/screen input.

This four digit alpha field contains in both first positions the line, and in the positions 3-4 the column of the selected field (if you use CPG2..QSF).

2. From-line/to-line during the output of partial maps.

In order to restrict the map output on particular lines, you enter the from-line/to-line into the field before the output. (CPGMLC = '0205': output line 2 to 5). At the same time, the field CPGMFN must be filled with '$QV'.

CPGMPF This two digit field contains the abbreviation of the last operated program function key.

CPGMRC This two-digit alphanumerical field has following content after a screen input: Either ' ' for 'nothing special', 'NI' for NO INPUT or 'IC' for 'invalid character'. IC will be set, if invalid characters or too much signs on the left or on the right side of the decimal point are identified during the transfer from the screen into a numerically defined field.

CPGNLS National Language Support.

CPGNLS contains a D for German installations or an E for English ones (according to customer configuration).

CPGPGM Phase name from the options, eight digit alphanumerical.

CPGPIW content of the IFC.

CPGSIN System information (see instruction COMRG).

Enter COMRG CPGSIN to make advanced system information disposable for the program.

CPGSMN segment name and key feedback area.

This area contains the key feedback area after a DL/I query.

CPGTCA Task Control Area

The TWA of the program can be processed with EDIT and SELCT via this field. From position 1 to 100 stands the indicator list, the user fields begin from position 117. CPGTCA is at most 4 K big.

CPGTCT Terminal Control Table.

This area may be changed or queried with EDIT- and SELCT operations. The data in the CPGTCT are disposed to the user of a particular terminal independent from the used program. The length of the field corresponds to the 'User-Area-Length' defined during the generation of the terminal control table. The maximum length is 255 positions, from which CPG uses the first 10 places internally. In the CPG 2 program at most 245 positions of the TCT are usable.

CPGTDI Variable Transient Data Name.

The name of the destination will be entered into this 4-digit field, if a variable transient data processing has been arranged in the files division.

CPGTID Terminal Identification.

Into this four digit field, the name of the terminal from the terminal control table will be disposed by the CPG for queries in the program.

CPGTIO Terminal-I/O-Area.

This area can be queried directly via a 'SELCT' operation after a query of a program. It may be used to read data together with the transaction identification. The Trans-Id is set in position 1 to 4 of the CPGTIO or from 4 to 7. The position depends on the way of the terminal input: either in an empty screen or in a formatted field.

CPGTIM Time numerical.

This field contains the time in numerical (packed) form in a 6-digit field (HHMMSS) in hours, minutes and seconds. This field can be used to calculate time intervals in the calculation description. The field becomes actualised at the program start and with the operation TIME. The actualisation can also take place from outside, if the field is in the Central Routine Library. If this is not wished, the field must be saved directly after the operation TIME with help of a Z-ADD-Operation.

CPGTSN Variable Temporary Storage Name

The temporary storage name will be entered into this 8 digit field, if a variable temporary storage processing has been arranged in the files division.

CPGVRL record length of a VSAM file with variable record length.

In this 5 digit numerical field the length of the read record is indicated after the reading operations. The field can be used for the output, if the key word VAR is indicated additionally to ALG and ADD in the record description. In this case, the length of the output record will be taken out of the field CPGVRL.

Before using internal fields, which guarantee a relation to particular areas of the TP CICS Interface, the programmer should test, if the TP CICS Interface used by him disposes these areas.

Date and time 2134

The date format depends on entries in CPGSTH (standard header) and CPGURSIT (customers configuration).

CPG disposes the following internal fields for date and time:

CPGDAI contains the day date in the form '0YYYYMMDDC'. (num).

This date form is practical in view of the millennium change because it is comparable and able to be sorted.

CPGDAT contains the date in the form '0DDMMYYYYC', that means with four places for the year. CPGDAT can be edited with edit code 'Y'.

CPGTIM contains the actual time in the form 'OHHMMSSC'. CPGTIM as six digit numerical field can be edited with the edit code Y. CPGTIM can be actualised in the program with the operation TIME.

UDATE contains the day date in the form: 'DD.MM.YY'. (alpha).

UDATEC contains the century in the form: 'YY' (alpha).

UDATEI contains the day date in the form: 'YYYYMMDD' (alpha) Purpose: See description CPGDAI.

UDAY contains the day of the day date: '0DDC' (numerical).

UMONTH contains the actual month: '0MMC' (numerical).

UTIME contains the time in the form 'HH.MMhrs'. (alpha).

UYEAR contains the actual year: '0YYC' (numerical).

Storage Types 2140

Alphanumerical Fields 2141

Alphanumerical fields can contain all printable and not printable characters. Each character covers a byte in the storage. Alphanumerical fields will be initialised on blanks before the start of the processing.

Numerical Fields 2142

Numerical fields can only contain digits and eventually a minus sign. Numerical fields will always be stored in packed form, that means that always two digits are stored in one byte. A half byte is reserved for the first sign. Numerical fields are initialised on zero before the begin of the processing.

Data fields, with which calculation operations shall be processed, must always be defined numerically.

Of course, the output in unpacked form is also possible. In this case, every digit covers a byte. The first sign will be stored together with the last character in the most right byte of the field. This can cause, that this character will be changed into a letter during the output.

CPG gives to all unpacked numerical fields, if they are positive, automatically a first sign, that makes the last character readable.

Note: If numerical fields are partially overlayed by other fields at the unpacked output, this automatism must be switched off with the entry 'SIGN' in the OPTIONS card.

Binary 2143

Numerical fields can be output and read again binary. The binary output takes place with the entry of the key word 'BIN' in the field description of the output division.

Example: - NUMBER 15 BIN.

Input fields are interpreted as binary, if 'B' or 'BIN' will be coded in the field description of the input division before the positions.

Example: `- B 11 15 0 NUMBER.`

Fields, that are two bytes big, will be installed while the output for numerical fields up to 4 positions, and 4 bytes big binary fields for fields up to 9 positions.

No binary fields must be used for screen and printer inputs and outputs.

Logically packed Fields 2144

Another form of packed storing is the logical packing.

A logical packed field is a field, whose numerical content is packed and stored without first sign.

Example:

The storage of the character 4711 necessitates:

Unpacked	4 bytes	F4 F7 F1 F1
Packed	3 bytes	04 71 1C
Logically packed	2 bytes	47 11

Data fields, that should be packed logically, may contain only positive numerical values. The logical packed form is only valid for the storage in external storage. For the processing in the application program, the data will be retransferred again in the packed form.

Numerical fields will be output logically packed, if the key word 'LOG' is coded in the field output.

```
-    NUMBER 15 LOG
```

Numerical fields are read logically packed, if a 'L' is entered in front of the first input position.

```
-    L 11 15 0 NUMBER
```

Unpacked numerical Field 2145

Numerical fields are always packed in the main memory (TCA). When read from the screen, a numerical value gets therefore the letter 'F' for the zone internally in the last half byte.

If a numerical value from the main memory has to be output unpacked on disk, so the zone will always be set to 'F'. For example the value 123 is then indicated on the disk as F1 F2 F3.

While reading from the disk, the zone does not change.

These regulations for the zone half byte of the packed field can be influenced with the OPTIONS parameter SIGN.

The entry causes, that when reading either from the screen or from the disk, in principle the zone 'C' is taken. At the output on disk, the zone does not change.

In this case, the character 123 is indicated on the disk as F1 F2 C3, so ‘12C‘.

Before	C	F	C	F
after screen input	F	F	C	C
after disk input	C	F	C	C
after disk output	F	F	C	F
OPTIONS	without SIGN	without SIGN	SIGN	SIGN

At numerical operations in the Procedure Division, the result field always gets the zone C, (for positive results) independent from the options parameter.

Screen input Fields 2150

Contrary to input fields of other files, a screen input field will only be read with a MAP operation, if the field has been modified on the screen before the operation. From the hardware this is told to the CICS interface with a 'Modified Data Tag'. The 'Modified Data Tag' can also be set by the programmer with the edit code. In these cases, the input field will also be read, if it has not been modified.

> **A field to be read will be identified with the transferred address via the input message of the 1.byte of this field, that means: only such fields, that were already output on the screen (preformatted), can be read.**

For numerical fields, the decimal adjust of the CPG will be processed as follows: Digits will be adjusted to the defined input field from the first input decimal point or after the last valid digit.

If no digits are entered, so missing decimal places are inserted automatically as zero values. (See example 3 and 5). Too much input digits are ignored, as you see in the example 2. If more digits are entered than a numerical field can take, the field will be adjusted from the (imaginary) decimal point. See also the examples 3 and 6 (next side).

Example: FELDA is defined numerically with 4 positions and 2 decimal places.
FELDB is defined numerically with 5 positions and 2 decimal places.

```
-   110 113 2 FIELDA;
-   209 213 2 FIELDB;
```

The fields 'FIELDA' and 'FIELDB' should be read from the screen; therefore the screen must have been 'initialised' before, that means to be pre formatted for the reception of the data. This takes place via the prior output of a corresponding field on the screen.

This field must be edited with the field characteristic 'unprotected'.

Example	field	pos.mask	input	field content	Number
1		A 115 "....."	"12.34"	12.34	1
2			"1.2345"	1.23	2
3			"123 "	23.00	3
4		A 215 "......."	"0.1234"	0.12	4
5			'123 '	123.00	5
6		" "	'1234567'	567.00	6
7	FELDA	A 116 " , "	" 4.25"	4.25	7
8	FELDA	A 118 " 0 , CR"	"12.34CR"	12.34-	8
9	FELDB	A 215 " , "	" 4.258"	4.25	9
10	FELDB	A 217 " 0 , CR"	" 12.34CR"	12,34-	10

Temporary Storing of Data 2160

The temporary storing can take place differently:

Temporary storing of data in the Transaction Work Area (not recommended!).

The Transaction Work Area (TWA) will be reformatted with each program start. However the programmer can enter the number of bytes of the TWA which have not to be reformatted in the options division behind the key word 'TWA' up from the beginning.

With this form of temporary storage, note the regulations for the program links.

Temporary storing of data in the terminal user area

With help of the EDIT-operation, the programmer can store data in the TCT of the CICS interface which can be queried again at any time with every program, but only from the same terminal. The query takes place with help of a SELCT operation. The result field is called 'CPGTCT' in both cases, and has not to be defined in the program.

Example:

Data store:

```
- -C; EDIT CPGTCT;
-     -O; FIELD CPGTCT;
-     -                NUMF  20  PAC;
```

The numerical field 'NUMF' will be stored in the TCT in packed form. The last position of the field is set on position 20 of the field 'CPGTCT'.

Data selection:

```
-     -I; FIELD CPGTCT;
            PAC  16 20 2  NUMF;

-     -C; SELCT CPGTCT
```

The numerical field, stored in the positions 16-20, will be transferred into the field 'NUMF'. The field was stored packed in the TCT. The field 'NUMF' will be defined with 2 decimal places.

The size of the field 'CPGTCT' will be fixed during the generation of the terminal table (TCT), in fact the parameter, which describes the user area length, must be 10 bytes bigger than the highest output position used in a program. So in the example above, position 20 + 10 bytes gives a TCT user area length of at least 30 bytes.

Temporary storing of data in the CSA. (not recommended!).

With the method described above for the TCT, data can also be stored temporarily in the common system area (CSA). These data can be queried again at any time with every program from every terminal. Therefore the example described above for the TCT is valid, if the field name 'CPGCSA' is set instead of 'CPGTCT'.

The size of the field 'CPGCSA' will be fixed during the generation of the initialisation table of the particular TP monitor, and in fact the CSA size parameter must be at least 16 byte bigger than the highest output position used in a program, because CPG uses the first 16 positions internally.

Temporary storing of data on the screen. (not recommended!).

The screen can also be used as temporary storage. The fields to be read from the following program must be output with an edit code containing the specification 'Modified-Data-Tag-On'.

Temporary storing and temporary storage queuing see chapter 2190.

Temporary storing of data on transient data.

Sequential output data can be stored temporarily on transient data. CPG uses transient data in each case for the temporary storing of printer outputs. The programmer can also use this service for own applications. However he should be able to estimate the effects on the performance. The programming is the same as for record files. The corresponding entry in the file assignment is 'TRANSDT'.

Temporary storing in the common area (for command level programs).

The common area is 4080 bytes big. CPG can communicate via the field CPGCOM and the operations EDIT and SELCT with the common area.

The length of the given common area can be determined with SELECT CPGPIW. It is positioned binary packed in the positions 239-240.

Temporary Storage Usage 2190

For the temporary storage of data, the programmer can use an area in the main memory or in a connected auxiliary storage if it is allowed by the TP monitor. CPG2 generates symbolic names for such storing areas. Under this name, data becomes temporarily stored, looked up and released. The name is composed as follows:

XXXXYYYY

XXXX = terminal identification. With the addition 'I' (for terminal-independent) in the FILE description, this part is equal '****'.
YYYY = storage name from the FILES division.

Therefore, this area must be defined with a 4-places name in a file assignment statement. The device (last entry) is called 'STORAGE'. The length of the reserved storage results from the entry for the record length.

Furthermore, CPG2 offers the possibility to keep the file name variable. At program execution time, the field CPGTSN must then be filled with a valid storage name (of 8 positions). This processing should always be used in case of non terminal tasks.

To avoid difficulties, which can arise through double given names, naming conventions, that the programmer must use, should be introduced.

A temporary data storage should be released at the earliest possible moment, to reduce need of main memory. For the input and output descriptions, the same regulations are valid as for record files.

Temporary Storage today is an area consisting of 1 or more records.

The programmer can determine with entries in the Data-Dictionary (recommended), or (in the record description of the output division or already) in the file description of the files division, if the record should rest in the main memory or should be stored on a disk, and if the data can be read only from the own terminal or from all terminals.

In the output division the following entries behind the storage name are supported:

Blank	The data can only be read by the own terminal and will be stored in the main memory.
“I“	The data can be read by all terminals and will be stored in the main memory (independent storage).

"A" The data can only be read by the own terminal and will be stored in the auxiliary storage. (auxiliary Storage)

In the files division one of the key words AUX for auxiliary storage and IND for independent storage may be indicated in front of the unit STORAGE. These key words have the same effect like A and I in the output division.

Entries in the output division have priority to the corresponding entries in the files division.

For temporary storage queuing, the storage must be deleted with the explicit instruction PURGE.

For older programs using Temporary Storage without queuing:

In the procedure division, temporary stored data can be read with a READ operation. So the programmer can determine with the service function of the READ operation, if the area will be released or is used again for later READ operations. Entry of the service function:

Blank	The area will be released after reading.
'SAVe'	The area will not be released after reading.

The area is released again in each case, if a new output takes place under the same name.

This technique is supported with files, that have an entry 'S' in column 16 for 'simulated queue' only (or entry SIM or corresponding entry in the Data Dictionary).

Temporary Storage Queuing 2195

There are five processing possibilities:

1. **Fill a queue**
2. **Change individual elements**
3. **Read sequentially**
4. **Read directly**
5. **Delete the whole queue**

For temporary storage queuing a 'Q' must be entered into the file description in the Data-Dictionary (recommended!) or into the file description of the files division (see chapter 3400).

For each queue, a CPGQxx field will be defined internally 5 places numerically. XX is the number of the file assignment.

FIX must always be entered as record format, variable record lengths are not supported and are treated like fixed record lengths.

Function 1 – Fill TS-queue:

```
- -I;  FILE STOR DD

- -C;  WRITE STOR
```

This example shows, how a queue must be filled.

In a queue may be stored at most 32.000 records, if it is possible for the TP monitor or the size of the batch partition. The user is responsible for an infringement, there isn’t any error message.

After an output, the record number is available in the field CPGQxx.

Function 2 – Change individual elements:

```
- -D;
-    NUMB  5 0
- -I;
-    FILE STOR
-       1 100 REC;
- -C;
-    NUMB = 4;
-    NUMB READ STOR;
-    IF CPGFRC = ‘  ‘;
-      NUMB UPDATE STOR;
-    ENDIF;
```

This example program shows, how single elements of a queue can be changed. If a selected element is not found, EF (end of file) is set.

In this example, the fourth element will be changed, the output is not processed at 'Not found'.

Note: Contrary to the UPDATE function for files, the output area is generally deleted on blank before the output on temporary storage. If the initial record should be kept for the output, it has to be transferred from the input into the output.

Function 3 – read sequentially (numerical literal possible):

```
-         1 READ STOR;                    * 1.Record
-         DO UNTIL CPGFRC = 'EF';         * following records
-            READ STOR;
-         ENDDO;
```

This example program shows, how to read a queue sequentially.

With a READ operation without factor 1, the elements will be read sequentially one after the other. But if you start to read a certain record (for example record 1), this record must be read with a defined and existing key.

Function 4 – read directly:

```
-     -D;
-         KEY 5 0
-     -C;
-     KEY = 5;
-     KEY READ STOR;
```

This example shows, how to read directly from a queue. The key, indicated in factor 1, has to be used to read the corresponding record. Contrary to file operations, the programmer must ensure for the following

READs, that the wished record number is always contained in the key, because a direct processing is always described with 'KEY READ STOR'. The operations CHAIN, RNDOM, SETLL etc. can not be used here.

```
- -C;
-    KEY = 0;
-    DO UNTIL CPGFRC = 'EF';
-       KEY = KEY + 1;
-       KEY READ STOR;
-    END;
```

This example shows, how you can read a whole queue directly.

Function 5 – delete a queue:

```
-    PURGE STOR;
```

With the operation PURGE, the queue defined in factor 2 will be deleted.

While a storage queue is deleted, the full area will be released and added to the available dynamic storage area.

There is no possibility to release only selected records. All records remain disposable until the area will be deleted with PURGE.

Simulation of the Queuing 2198

With the entry S in the files division or (better) in the data dictionary, you reach, that a TS area can be processed by the programmer according to the single record logic, but that a queue will be generated by the CPG that consists of one record.

Example:

```
-   FILE STOR UPD S FIX 256 STORAGE;
```

Service Functions

Flow Chart 2240

A flow chart can be listed right beside the compilation list of the CPG2 program, which shall simplify the reading of the program also for the non professional.

The flow chart is made up in the following way:

```
Flow Chart              PHASE XXXXXXX           Headline like OPTIONS
SCREEN                                          file assignment L3270
DISK                                            file assignment DISK

DATE   ---------        11   16                    data division, TWA overlay
    DAY                 11   12
    MONTH               13   14
    YEAR                15   16

INPUT  -------------    DISK                    Input file INPUT (disk file)
    X                                        75 field  X is 75 places long (ALPHA)
    Y                                       7,2 field  Y is 7 places, 2 dec.
    FG                  10 *    5               Array  10 * 5 places long

--------------------------                      Beginning of the Procedure division

START -------I                                  START DAY
             I                                  Procedure division without branching
            !F!                                 EDIT or SELCT operation
01 ---       I                                  Combine level for DO- and IF
             I                                  groups (at DO, IF and END )
02 ---       I
 02 --       I                                  moved at ELSE, BREAK, CONTINUE
02 ---       I
01 ---       I
            /// ALL                             EXCPT
             I======UPRO                        EXSR subroutine or EXPR subroutine
             I
             I------OUT                         GOTO OUT or EXITP OUT
  INPUT    ///                                  Read file  INPUT
             I
            < >-----START                       GOTO START if... (e.g. 15 on)
             I
            /// OUTPUT                          UPDATE, ADD ( WRITE ), OUTPUT
             I
             -------START                       GOTO START without condition

--------------------------                      End of the Procedure divisions
OUTPUT  ------------------                      output file  OUTPUT
  X                     1         75            field  X begins on pl. 1, is 75 long
  Y                     76         7,2          field  Y begins at 76 is NUM 7pl/2Dec
  FG                    1   10 *    5           Array begins at 1 is 10*5 long
```

Note, that only the first statement of a line is considered in the flow chart. If the flow chart will be used, this must be considered when coding.

Flow chart for overlays with ORG:

The instruction ORG offers the most flexible possibilities for the redefinition and overlay in the data division. The flow chart indicates the up/to positions per field in the data division. When using the ORG instruction it will be indicated: 1. The next up position, if no ORG would be coded and 2. the new up position as result of the ORG instruction.

Example: Source Code		Flow Chart			
F1	10	F1		1	10
F2	5 * 10	F2		11	60
F3	5	F3		61	65
F4	5	F4		66	70
F3	ORG	ORG	F3	71	61
F5	5	F5		61	65
F5	ORG	ORG	F5	66	61
F6	5	F6		61	65
	ORG	ORG		66	71
F7	10	F7		71	80

List control 2245

Exceptions from the general syntax rules appear for the entries to adapt the compilation list:

/EJECT	for the forms feed at any place
/NOLIST	to suppress the listing
/LIST	to terminate /NOLIST

For these commands there are no entries necessary in column 6 and they always begin on column 7.

Programmer Check List 2246

A programmer check list will be printed in addition to the compilation, that reminds the programmer of certain conditions of the TP control program and shall give information for the most important program data.

The check list has following arrangement:

```
TITEL                                   KD-NR   USER-ID      26.02.98
----------------------------------------------------------------------

PROGRAMMER CHECKLIST

   CONTAINS THE CICS PCT AN ENTRY WITH THE PARAMETER PROGRAM=TSTO26 AND
                                            TWASIZE=00190 OR HIGHER ?
   CONTAINS THE CICS PPT AN ENTRY WITH THE PARAMETER PROGRAM=TSTO26 ?
   CONTAINS THE CICS FCT AN ENTRY WITH THE FILENAME CPGKDN ?

PROGRAMVALUES
   PROGRAMSIZE = 544 BYTES (CIRCA)
   TWA SIZE    = 190 BYTES
   TIOA SIZE   =   4 BYTES
   DEFINED INDICATORS
   T1

 EXTERNAL PROGRAM RELATIONS

   EXIT:  'TRID'
   EXIT:  PHASEX
   EXPR:  PHASEY
   PROG:  QPGMODULE
   EXHM:  MODUL
```

The indication of the program size is a circa indication. The exact program size can be determined via decimal calculation of the indicated address for 'CPGPND'.

The TWA size is the size of the necessary working storage. It will be calculated exactly. However, copy books put in by the user, are not contained in the TWA size. The TWA size is calculated with the address at CPGTND minus the length of the TCA, normally 256 bytes.

The Terminal IO Area (TIOA) may not be bigger than 4080 bytes. A bigger TIOA leads to program errors. If the value exceeds the maximum, this can happen if many little fields are put out on the screen, the terminal output is to devide in several smaller parts. To avoid problems with the TIOA, use QSF!

All indicators used in the program will be listed in the check list in sorted order.All external program relations will be listed in the described form at the end of the check list. The programmer must ensure that no loops emerge while using the EXITP or EXPR operation with the phase name in factor 2, because this could lead to considerable performance losts.

Cross Reference 2247

A cross reference list can be printed in connection to the compilation. Therefore a XREF must be entered into the OPTIONS parameter.

The cross reference list gives information, at which CPG statement numbers the following program elements were used:

file names
indicators
Alphanumerical literals
data fields, arrays and tables
Numerical literals
Tags, internal and external subroutines
operations

Indicators 2260

The flow of a CPG program can be controlled with indicators. The indicators 01 to 99 can be set on or off by the programmer. This processing mode is outdated and will not be described here completely.

Indicators (as described below) should be queried with the operation IF CONDITION.

Following indicators will be set at the processing time per program key by the user and can be queried in the program:

P1, F1 or PF1	Program function key 1
P2, F2 or PF2	Program function key 2
P3, F3 or PF3	Program function key 3
P4, F4 or PF4	Program function key 4
P5, F5 or PF5	Program function key 5
P6, F6 or PF6	Program function key 6
P7, F7 or PF7	Program function key 7
P8, F8 or PF8	Program function key 8
P9, F9 or PF9	Program function key 9
PA, F10 or PF10	Program function key 10

PB, F11 or PF11	Program function key 11
PC, F12 or PF12	Program function key 12
Q1, F13 or PF13	Program function key 13
Q2, F14 or PF14	Program function key 14
Q3, F15 or PF15	Program function key 15
Q4, F16 or PF16	Program function key 16
Q5, F17 or PF17	Program function key 17
Q6, F18 or PF18	Program function key 18
Q7, F19 or PF19	Program function key 19
Q8, F20 or PF20	Program function key 20
Q9, F21 or PF21	Program function key 21
QA, F22 or PF22	Program function key 22
QB, F23 or PF23	Program function key 23
QC, F24 or PF24	Program function key 24

A1 or PA1 PA-key PA1	With the use of PA-keys,
A2 or PA2 PA-key PA2	no data will be read
A3 or PA3 PA-key PA3	from the screen.
DE	data entry
SP	Selector pen query / Cursor Select key (Pos. sel.)
CL o. CLEAR	Delete key

Note: These keys can also be queried in the internal field CPGMPF in the program (2 places, so P1 up to QC etc.).

These indicators as well as 'NI' and 'IC' (see below) remain also up to the next screen input at program relations.

For the processing of disk files the following indicators will be set, if the corresponding condition appears. The indicators can be queried by the programmer just after the input/output operation.

EF or EOF file end at Transient data
EF or EOF file end at sequential disk processing
EF or EOF file end or 'NOT FOUND' at Temporary Storage Queue processing
EF or EOF If there is no place left for the addition at Temporary Storage Queues.
EF or EOF file start at READ BACK (only VSAM)
EF or EOF If the indicated program is not in the PPT at program calls with the operation EXITP with program name or EXPR.
EF or EOF If at program calls with EXITI either the called task or the asked screen are not available in the corresponding CICS table.

DR	Duplicate record for the addition.
DR	Duplicate key for the READ of alternate index files.

At sequential disk processing the indicator 'EF' (End of file) will be set, if the last record of the file was read. This indicator can also only be queried by the programmer.

For the addition of records to a ISAM- or VSAM file, the indicator 'DR' will be set, if a record with the same key is already available. The indicator will be deleted at the next 'ADD' output.

If the error 'DR' appears while the sequential reading of a path of a VSAM file, it means, that several records with the same key are available. If the last record of a group was read, the DR switch will be deleted.
Note: This file return code is also available for the program in the internal field CPGFRC.

After a screen input, the indicator 'IC'(Invalid Character) can be queried, which indicates, if a not numerical sign (for example letter o for zero) has been found in the input for a numerical field.

However, the indicator must be queried just after the READ- or MAP(I) operation.

After a screen input, the indicator 'NI' (No Input) can be queried. If no data is read from the screen, the indicator NI is set. The indicator must be queried just after the reading operation.

After an output with EXCPT, the indicator 'NO' (No output) can be queried. The indicator will be set, if the EXCPT instruction does not lead to an output.

Is channel 12 reached at the list output, the indicator 'OF' for overflow will be set.

Furthermore, the UPSI indicators U1 to U8 are available.

The content of indicator '00' has following meaning:

X'F0'	Sequential reading
X'08'	EF (End of file) condition
X'04'	DR (duplicate record) condition
X'02'	CHAIN U

These values cannot be called in the program, but they can be seen in the Dump.

File Processing 2300

File name 2301

File names are not more than 8 places long. The first sign must be a letter from A to Z or a '$' (Dollar). The following places can contain numbers and letters.

File names have to differ from field names. The check for a CPG error message can only be processed for the first six places of the file name.

Up to 100 files can be defined in a program.

Keys 2304

A key field or key value in factor 1 of the Procedure division must normally be indicated for the operations CHAIN, DELET, READ, READP,UPDAT, SETLL and WRITE.

An alphanumerical key value can be specified as a literal, closed in inverted commas. Such a literal can be up to 8 bytes long. If an indicated literal is shorter than the key length defined for the file in the file assignment, the rest will be formatted with X'00'.

If the key has numerical (that means packed) format, the key value can not be indicated as decimal literal.

If the key is a field defined in the TWA, the length of the field should correspond to the length defined for the file in the file assignment. Is it shorter than the key length, the resting bytes will be formatted with X'00'. Is it longer than the key length, it will be cut off (that means right adjusted bytes are ignored).

If files are processed with key fields in the packed decimal format, they must be processed with attention. For the IBM hardware the hexadecimal values 'C' and 'F' are valid for the processing of numerical fields as a positive sign. This can lead to an apparent not finding of the record, if the sign 'F' is coded and a field with the sign 'C' is specified in the program. The sign of the key field can be rectified with the MLLZO operation.

IF the sign of the record key in the file is a 'C' and in the program field a 'F' is indicated (for example if it was read from a screen into a numerical field), the sign of the program field can be conversed into a 'C' with help of a Z-ADD operation.

Assembled key fields can be structured with help of the EDIT function or per definition of overlay fields or with data structures.

Sequential or random access 2306

A DISK file defined in a CPG program, can either be processed in random access mode (that means that a special key is indicated for every read record) or sequentially. The effect of the CPG2 file operations depends mostly on the file processing mode.

At the beginning of a transaction is assumed ,that all DISK files defined for the program should be processed in random access mode.The file operations supported in this mode are CHAIN, WRITE, UPDAT, DELET, EXCPT and RNDOM.

If a READ-, READP- or READB operation is processed for a DISK file, the sequential processing mode for this file will be set automatically.

In this mode, the file operations READ, READP, READB, SETLL are supported. (CHAIN is also supported, but works as SETLL when used in the sequential mode.)

If a file has been changed into the sequential mode, it rests there, until the processing of a RNDOM operation for this file. This causes, that the file is switched to random access mode again.

File Operations in the sequential access 2307

The sequential processing of a DISK file is started with the processing of a READ-, READB-, or READP operation.

The file rests in the sequential mode, until the random processing mode will be reset for the transaction with help of a RNDOM operation.

These read operations serve for the sequential processing of a whole file or several parts of a file (random-, generic processing or scanning).

With the READB operation, a VSAM-file will be processed backwards in key sequence (that means in inverted key sequence).

The key value serves only to the positioning of the file for the reading of the first record, when the sequential processing is started. At a READ- or READP operation is this the first record in the file , whose key is equal or bigger than the indicated key. At a READB operation, this is the record, whose key is equal to the specified key. If the key value is bigger than the one of the last record in the file (READ, READP) or smaller than the one of the first record in the file (READB), so the file end indicator is 'EF' set. This one must be queried just after the read operation.

If the sequential processing is started, all following read operations read the following record in the file . This happens without consideration for the really indicated key value. However, the entry for the key field must be made(not for Batch programs).

If the file end is determined, the EF indicator is set, and the input description for the file will not be executed. The program will continue with the next instruction in the Procedure Divisions after the READ operation. If further sequential read operations are processed for the file, no records will be read and the EF indicator will be set.

For VSAM, the EF indicator must absolutely be queried. A further READ leads to a system error message.

VSAM Alternate Indices 2315

A VSAM file can be read with an alternate key field. This reading mode with alternate keys needs another entry in the file table (FCT).

If a record is added to the base cluster and if an alternate key (unique) already exists, the indicator DR (duplicate record) is set.

The 'DR' indicator is set, if a file is read with no clear key.

For a following reading with unique key, the DR indicator is set off.

VSAM-ESDS/RRDS 2316

ESDS- and RRDS files will be addressed via the relative byte address (RBA) or the relative record number. For the access, the RBA must be disposable as binary numerical field in the length 9 or as four places alphafield.

If a numerical key field is entered for a file access, the binary edit takes place automatically. If an alpha key is entered, the programmer is responsible for the binary edition (compare example 5-7, chapter 8000)

VSAM Files in entry sequence 2317

Files in entry sequence (also called sequential input files) are supported. If records are added to a file in entry sequence, they will be added at the end of the file. A key for an EXCPT or UPDAT operation will be ignored. If a record is to be read, the relative byte address (RBA)of the record must be indicated as key. The RBA will be calculated as sum of the bytes of all records of the file already read. This is only possible for records with fixed length.

In the file assignment, a 'R' or 'RBA' must be coded to define the relative byte addressing.

To guarantee, that the added records will really be stored on the disk, the processed record should be read with a CHAIN operation after an ADD or WRITE. (By the operation CHAIN, the VSAM CI will be written to the file.) The field CPGKxx will be indicated as key, whereby xx stands for the position of the file in the files division.

Data View Processing 2340

Data View Definition 2341

A data view is a composition of data for a determined problem.

A view can be composed as wished from fields and records from different files. The logic of a view differs from other storage forms because all of its elements can serve as key field.

Realistation of a Data View 2342

In relational data bases a data view is created online and rests in the main storage during the processing.

To simulate this way of processing, following way can be chosen: With a CPG2- or CPG3 Query program, a data view will be produced and stored on the file CPGWKV. The exact steps are described in chapter 7500.

The view will be loaded from the file into the main storage only if it is processed in a program. The loaded view rests in the main storage until the next Shut-Down of the TP monitor.

The produced view has the form of a table with the particularity, that every column of the table can serve as a key.

Processing of Data Views 2343

Note for the processing of data views with CPG:

A file description must be coded for the view. TABLE must be indicated as device, the sum of the fields of the table as record length and the maximum length of the possible key fields as key length.

A data view is processed similar to a file; all fields to be read from the table (especially all key fields), must be declared in the input descriptions under the name of the table.

The view is read with the instruction FIND. Factor 1 contains the key element, factor 2 the name of the view (up to four places).
An indicator for the comparison for equality indicates, if the search argument from factor 1 has been found in the view. If the indicator is omitted, the field CPGFRC contains 'EF' if no matching element has been found for the indicated key.
If during the processing should be returned to the beginning of the table, the sequential processing of the view will be ended with the operation RANDOM. (If an element, searched with FIND was not found, the cursor returns automatically to the beginning of the table.)

Example:

A view is created with data of the article file and the customer file.

Both files have only one key each: The article number and customer number. These keys are called 'primary key(s)' in the following.

In the created view, the data may not only be looked up for the primary keys, but also for all others, in the following indicated as 'secondary keys'. Examples for such secondary keys are postcode, first position of the postcode or agent number.
When creating the view must be considered, how much 'columns' the generated table should have. Note for the use of the view processing, that the view contains the primary keys of the files of whose data the view is composed. So is assured, that all data can be CHAINED after a successful FIND, that are related to the view. Compare also example 17, chapter 8.

Programming Assistance 2400

Decision Tables 2410

CPG2 enables the direct processing of decision tables in the Procedure Division.

With the operation BEGDT, you may switch over to the decision table logic and to the fixed RPG format instead of the CPG2 syntax.

A decision table is not interrupted with the CPG2 statements, and terminated with the operation ENDDT.

A decision table consist in the upper part of conditions, and in the part below of actions. All conditions as well as the actions of a decision table must follow each other directly.

The operation code '-----' from column 28 to 32 indicates, that the conditions end here and the actions begin.

The columns 43 to 74 contain the connection list. Possible entries are in the condition part for each available column:

Y	for 'YES' or condition fulfilled,
N	for 'NO' or condition not fulfilled,
BLANK	for: condition is not decisive.

Possible entries for actions for each available column are:

Blank	Action will not be executed,
Not blank	Action will be executed, if all conditions standing vertically above in the same column are fulfilled or not, depending on the entry (N).

Formular description.

General entries:

Column 6 'C' must be entered.

Column 7-17 rests free

Column 43-74 Connection list (see above).

Conditions.

Column 18-27 contains a valid field name, the name of an array element (FG,I) or a literal.

Column 33-42 contains a valid field name, the name of an array element (FG,I) or a literal.

Column 28-32 Operation key. Valid entries are:

> or '**HIGH**'	bigger than	factor 1 is bigger than factor 2.
< or '**LOW**'	smaller than	factor 1 is smaller than factor 2.
= or '**EQUAL**'	equal.	factor 1 is equal factor 2.

Note: For all comparing operations, factor 1 and 2 must both be defined either numerically or alphanumerically.

Actions.

Column 18-27 rests free.

Column 28-32 Operation key. Valid entries are :

GOTO	Branch to.
EXSR	Execute subroutine
EXCPT	Output

However, no second decision table can be called with EXSR from a decision table.

Example.

A customer file will be processed sequentially: Thereby, all customers whose turn over is more than 10000 DM, should be indicated on a screen and whose debit balance is higher than the turn over or whose debit balance is higher than their overdraft limit. All other customers will not be displayed.

```
- CONTIN.
-   READ SCREEN.
- NREAD.
-   READ CUSFIL.

- LABEL BEGDT
C           REVENU    >    10.000    YY
C           OPENV     >    REVENU    Y
C           OPENV     >    CREDIT     Y
C           OPENV     >    10.000      Y
C                     -----
C                     EXSR CHECK       X
C                     EXCPT          XXX
C                     GOTO CONTIN    XXX
C                     GOTO NREAD        X
- ENDDT
```

Field Edition 2420

CPG offers the possibility, to edit fields via the Output division. With the operation MOVE only up to 8 bytes can be transferred into a field as literal, with the operation '=' 24 places.

With the operation EDIT, every alphanumerical field can be edited in its whole length like an output record. The result field of the operation contains the name of a field, whose edition is described in the Output Division (see operation code EDIT).

Example:

```
DATA DIVISION
  LINE  70
PROCEDURE DIVISION
  EDIT LINE
OUTPUT DIVISION
  FIELD LINE
     CUSNO    7
     NAME    33
     CITY    55
     REVENU  70 ' .   . 0 ,  -';     * edited with pattern
```

Inversed with the operation SELECT, several fields can be taken out of the edited field. In the SELCT operation the name of the field is described in the Input Division. In the Input Division is described which parts of the field are to be moved into other fields.

Example:

```
INPUT DIVISION;
   FIELD LINE
      1  7 CUSNO
      8 33 NAME
      38 55 CITY
PROCEDURE DIVISION;
   SELECT LINE
```

After the execution of the operation, the field 'CUSNO' gets the bytes 1 to 7, the field 'NAME' the bytes 8 to 33 and the field 'CITY' the bytes 38 to 55 of the field 'LINE'.

Field Edition with TYPE 2425

To select different structures from the same field, a select type can be indicated with the key word TYPE.

Example for the Input Division (output analogous):

```
      - -I.  FIELD CPGCOM  TYPE  PROG1
      -         PACKED  1  4 0  CUSNO;
      -                 5 34    CISTMR
      -      FIELD CPGCOM  TYPE  PROG2
      -                 1 20    PARAM

      - -C.
      -      SELECT CPGCOM TYPE PROG2
```

In the example, a CPG program works with program relations together with several other programs. The data exchange is made via the Common Area. The SELECT instruction from the example above applies to the Common Area of the program 2, which has another construction than for example the communication area of program 1.

Note:

In the program code of the input and output division, the different types of a field edition must be coded without interruption by descriptions for other fields or files.

Structured Programming 2450

CPG2 offers the following operations for the structured programming :

BREAK	(terminate a DO loop or all DO loops)
CAS	(compare and branch into a subroutine)
CONTINUE	(interrupt a specified loop processing)
DO	(execute)
DO UNTIL	(execute..up to)
DO WHILE	(execute..during)
ELSE	(other..execute)
END	(end)
ENDDO	(DO end)
END-EVALUATE	
ENDIF	(IF end)
EVALUATE	(execute just one of several alternatives)
IF	(if..then)
WHEN	(if..then in an EVALUATE statement group)

The structurizing operations DO, IF, EVALUATE and CAS always indicate the beginning of a statement, which will be terminated with an END instruction.

As comparing operators in DO-, IF- and WHEN queries can be set:

Operator	Meaning
> GT	factor 1 is higher than factor 2
< LT	factor 1 is less than factor 2
= EQ	factor 1 is equal factor 2
>< <> NE	factor 1 is not equal factor 2
>= GE	factor 1 is higher or equal factor 2
<= LE	factor 1 is smaller or equal factor 2

Example:

```
      DO WHILE ERROR = '  '
            IF WERT1 > WERT2
                  :
            END
      END
```

If a DO group contains another complete DO- or IF group, it is called a nested DO group. In the example, the nesting depth is 2, at most 40 nested groups are supported by the CPG.

A CPG program can contain at most 999 DO- and IF instructions.

AND linking 2453

With the logical linking, several conditions should be checked in one program step. The logical linking with AND is supported for the operations IF, DO and WHEN.

(See chapter operations - IF instruction and example 26)

BREAK Operation 2455

The operation BREAK terminates the actual DO-, DO UNTIL- or DO WHILE-loop and branches behind the affilated END statement. BREAK ALL branches behind the END of the outermost DO loop.

CONTINUE Operation 2457

The operation CONTINUE interrupts the processing of the actual DO-, DO UNTIL- or DO WHILE loop. CONTINUE leaves the loop before the END statement and branches back to the loop condition. The statements between CONTINUE and END are not processed, but the loop will continue according to the loop condition.

DO Operation 2460

Example: `DO FROM X TO Y WITH I`

The DO operation works as follows:

For DO, the starting value is stored into the index at first.

The index is compared to the limit value. If the index is higher than the limit value, the program control branches to the statement that follows the corresponding END.

If the index is lower or equal to the limit value, the operations between the DO and the corresponding END will be processed.

In the END instruction of the DO group, the increment of the loop will be added to the index field. (the increment is normally equal 1, but can also be indicated as positive number after the ENDDO instruction.) Afterwards, the program control branches back to the corresponding DO and compares at new the index value and the limit value.

To terminate a DO loop, the limit value can be changed within a loop. An index can not be changed. The loop interruption is also possible with the operations BREAK and CONTINUE.

At the end of the loop, the index value is bigger than the limit value.For a loop, that runs from 1 to 12 (with the increment of 1 per loop processing), the index after the processing is equal 13. A DO group ends with the instruction END or ENDDO. If an increment value is indicated for the loop index, the operation ENDDO must be chosen.

DO UNTIL Operation 2461

The operation DO UNTIL works as follows:

The statements of the DO group will be executed as long as the indicated condition is fulfilled. The condition will be checked after every processing of the DO group. If the condition is fulfilled, the loop will not be processed, but it is branched to the statement, which follows after the corresponding END operation.

The condition for the processing of the loop can consist of several logically linked conditions. Therefore the operators AND and OR are available.

If the condition should not be checked at the first execution of the loop, the service function '1' is appended at the DO UNTIL statement.

DO WHILE Operation 2462

The operation DO WHILE works as follows:

The statements of the DO group will be executed as long as the indicated condition is fulfilled. After each processing, the condition will be checked at new. If the condition is fulfilled, the loop will be processed as specified, otherwise is branched to the statement that follows after the corresponding END operation.

The condition for the processing of the loop can consist of several logically linked conditions. Therefore the operators AND and OR are available.

ELSE Operation 2463

An ELSE operation indicates the beginning of the operations, which are executed, if the check of the corresponding IF operation does not apply.

The operation ELSE is always a separate statement in the CPG.

END Operation 2464

The operation END must finish every DO- and IF group.

If in a DO loop the increment value should be unequal 1, it can be indicated as factor 2 of the operation ENDDO as a numeric literal witha value greater equal 1.

For the documentation, also ENDIF or ENDDO can be programmed instead of END.

EVALUATE Operation 2465

The EVALUATE operation is chosen, if (at most) one of several alternatives should be executed.

'EVALUATE' must be indicated as operation term. The alternatives will be distinguished with 'WHEN' in the following lines. The factors for the conditions can contain numerical or alphanumerical fields, field names or array elements. If a condition is fulfilled, the following instructions will be processed. Afterwards, the evaluate instruction ends and the program will continue after the 'END-EVALUATE'.

The condition 'WHEN OTHER' is fulfilled, if none of the WHEN instructions before applied. In this case, the accompanying instructions will be executed and the EVALUATE is finished.

IF operation 2466

The IF operation works as follows:

Both factors of the comparison must be of the same type. They contain either an alphanumerical or numerical literal, a field or an array element. If the relation between factor 1 and factor 2 is not fulfilled,the program control branches to the statement, that follows after the corresponding END or ELSE instruction.

An END description must be entered to terminate an IF operation.

If an ELSE description follows an IF description, the END instruction must be entered after the ELSE description and not after the IF description.

IF CONDITION 2467

The IF CONDITION works as follows:

Only indicators and other switches can be queried. The queried condition can exist from up to three indicators that are logically linked with AND.

If the condition is fulfilled, those statements are executed, that are coded between the IF and the next END or ELSE. If the condition is not fulfilled, the statements behind the corresponding END or ELSE are executed.

OR Connection 2468

With the boolean connection, several conditions can be checked in one program step. The logical OR connection is supported for the operations IF, DO and WHEN.

(See also chapter operations – IF instruction and example 26)

WHEN 2470

WHEN works as follows:

The WHEN operation indicates a condition. If it is fulfilled, all following instructions will be executed up to the next WHEN or up to the END-EVALUATE.

WHEN OTHER 2471

WHEN OTHER is the 'ELSE' branch of the multiple alternative (EVALUATE):

If there was none of the WHEN conditions before fulfilled, the instructions of the WHEN OTHER group (up to the END-EVALUATE) will be executed.

WHEN OTHER is (optionally) the last condition query in every EVALUATE group.

Boolean Connection of IF, DO and WHEN Operations 2475

DO, IF and WHEN operations can be logically linked with the boolean operators OR and AND.

1. Or connection with OR at the IF operation

The statements up to the next END or ELSE will be executed, if at least one of the IF conditions is fulfilled. If none of the conditions is fulfilled, it is branched behind the next END or ELSE.

2. Or connection with OR for the WHEN operation

The statements up to the next WHEN instruction will be processed, if one or both WHEN conditions are fulfilled. If no condition is fulfilled, the program will continue at the next WHEN operation or behind the END-EVALUATE.

3. Or connection with OR for the DO UNTIL / DO WHILE operation

The statements up to the next END will be executed, if at least one of the DO conditions is fulfilled. If none of the conditions is fulfilled, it is branched behind the next END.

4. And connection with AND for the IF operation

The statements up to the next END or ELSE will be executed, if all IF conditions are fulfilled. If only one of the conditions is not fulfilled, it is branched behind the next END or ELSE.

5. And connection with AND for the WHEN operation

The statements up to the next WHEN instruction will be processed, if both WHEN conditions are fulfilled. If one of them or both are not fulfilled, the program will be continued at the next WHEN operation or behind the END-EVALUATE.

6. And connection with AND for the DO UNTIL / DO WHILE operation

The statements up to the next END will be executed, if all DO conditions are fulfilled. If only one of the conditions is not filled, it is branched behind the next END.

OR and AND can be used mixed. Thereby applies, that AND bounds more than OR.

Always only one END (IF) belongs to the linked IF conditions, always only one END (DO) to the linked DO conditions. The operation WHEN is not terminated with END.

Data Structures 2477

CPG allowes to determine an area in the storage as well as the installation of fields - called subfields – within this areas. This area in the storage is called data structure. A data structure can be used to

describe the internal area several times with the use of different data forms,

calculate with a field and change its content,

divide a field into subfields, without using MOVE or MOVEL instructions,

describe a data structure and its subfields in the same way as a record is defined,

group not connected data into connected internal storage areas.

Data structure instructions are described in the Input Division as follows:

```
- FILE  name  DS  (length)
```

Following rules have to be considered for the specification of the data structure instructions

the data structure name must be a valid symbolic name, at most 30 places long. It can be used everywhere, where an alphanumerical field is allowed.

All entries for a data structure and its subfields must appear together; they cannot be mixed with entries for other data structures.

Subfields are defining for the data structure; therefore subfields must not be part of different data structures at the same time.

The length of a data structure can be as follows:

The length, indicated in the input field descriptions, if the data structure name is an input field name.
The highest to-place of a subfield within a data structure, if the data structure name is no input field.
The length that can optionally be indicated in the instruction described above in the Input Division
The length of the data structure is determined with the first instruction in the program, that defines a length of the just described types. Following different length indications are not valid.

A data structure and a subfield of a data structure cannot have the same name.

If a SELCT operation is used for a data structure, the SELCT input descriptions must be coded before the data structure descriptions.

If a field is defined in a data structure, it must not be subdefined in the Data Division.

Data Structure Subfield Descriptions 2480

The subfields of a program described in the data structure must follow just after the data structure instruction, to which they belong. The syntax is the same as for other field inputs.

Use of a data structure, to define subfields within a field:

```
- FILE INPUT
-   3 18 PARTNO
-  19 29 NAME
-  30 40 PATNO
-  41 61 DR
-  FILE PARTNO DS; 1  4   MFG
-                 5 10   DRUG
-                11 13   STRNTH
-           PAC  14 16 0 COUNT
```

The data structure subfields can be addressed with the name PARTNO or with the subfield names MFG, DRUG, STRNTH or COUNT.

Use of a data structure to group fields:

```
- FILE INPUT
-    3 10  PARTNO
-   11 16 0 QTY
-   17 20  TYPE
-   21 21  CODE
-   22 25  LOCATN
- FILE PRTKEY DS
-    1  4 LOCATN
-    5 12 PARTNO
-   13 16 TYPE
```

If a data structure is used to group fields, fields from places on the input record that are not nearby can be grouped toghether. With the data structure name and/or the individual subfield names, this area can be addressed.

For the moment, at most 500 entries can be made in the data structure table. An extension is possible. See Copy CPG*CDTB.   (* = Release suffix).

TWA Overlay 2485

Overlay fields can also be defined. An overlay field is a field, which is separated in further fields. It is defined in the Data Division as an array, but '0' has to be indicated as number of the elements.

Overlay fields can be numerical or alphanumerical.

The definitions of the fields, by which the area should be structured, must follow directly after the specification of an overlay field. These are defined as ordinary fields or arrays.

An overlay field can also contain further overlays.

Note, that enough fields must be defined to fill the whole defined area for an overlay field. The block diagram can be used as control. To be able to use the block diagram, only one field definition should be coded per line.

Example:

```
- ANSCHR   0 * 60;   NAME  20;  ORT  20;  STR  20;
```

(In this example, there will be no usable block diagram, because the block diagram can only be printed for the first statement in a line.)

The three fields NAME, ORT, STR can can be called with the common name 'ANSCHR' caused by the overlay; while reading the field ANSCHR, also the fields NAME, ORT and STR will be filled.

For numerical fields is to note for the overlay, that these are stored internally in packed form.

Selector Pen Selection 2490

When reading into the screen, it is possible to choose only a determined selected field. This selection happens with the tap of the selector pen or if you set the cursor onto the chosen field and push the button 'pos sel' for the selection (for positions selection).

After this so called selector pen selection, in the program

the switch SP is set
the 6 places alphanumerical field CPGMFN is filled with the name of the chosen field (when using QSF)
the 3 places numerical field CPGMFI is filled with the index of the chosen element, if the chosen field was an array (when using QSF)

Note the following for the selector pen selection :

Every selector pen selectable field must have a blank in the first byte.

Selector pen selectable fields and fields, for which the attribute 'Modified data Tag' is set, may not be at the same time in a map.

This would lead to errors, because the 'Modified Data Tag' works in the same way as the selector pen selection internally.

Cursor Selection 2495

The selector pen selection has lost its importance. Nowadays, such selections can be reached with a simple cursor positioning.

The internal fields CPGMFN and CPGMFI are filled at every screen input.

CPGMFN contains the field name, in which the cursor stands.

CPGMFI contains the index of the array elements, if the cursor stands in an array.

Optimizing of the TWA Size 2500

The TWA size will be minimized by the CPG. Fields, that are defined in the Input Division (implicitly), will not be defined in the TWA, if they are processed neither in the Procedure Division, nor in the Output Division.

The optimizing function will not be activated for fields, which were defined explicitly in the Data Division.

The optimizing function will also not apply for the data structure subfields. The not optimized fields will be marked internally as 'used'; data structure subfields are also always 'used'.

The optimizing can be canceled with the Options Parameter DEF for define.

Rules for the pseudo conversational Programming 2550

The clear key or another program function for the program end must be queried by the programmer.

For the transaction oriented program call, the operation EXITT is the appropriate EXIT operation.

The screen data will be transferred into the program with the operation MAP.

The data of the TWA (Transaction Work Area) are not available after the task end. The relevant data for the program must be stored temporarily (for example into a Temporary Storage Queue) and read into the program at the start of the next task.

Consider, if these temporary storage areas should be deleted at the end of the application.

The VSAM strings are released at the end of the task. This is to note for example, if a VSAM file is to be read sequentially.

Data Dictionary in the Files Division 2605

Data Dictionary is always attracted in the file description, if files are described incompletely in the Source Code. The description of the record type ' ' (2 blanks) is adopted in the program.

Like in the other divisions, the key words DD (for Data Dictionary) and TYPE (for the record type) can here be indicated here to choose different file descriptions.

Application examples:

- Choice between input, output and update file in the Batch
- Choice between physical file and HL1 dataset

Coding:

```
    FILE KUNDEN  DD  TYPE  HD
    FILE KUNDEN  DD  TYPE  HD INPUT
```

Data Dictionary in the Data Division 2610

1. DEFINE structure

Structures, that are described in the Data Dictionary, can be adopted in the data division with DEFINE 'structure name'.

If a record type of the structure should be attracted, it has to be appended as a two places literal to the DEFINE instruction (see example).

The key word TYPE can be indicated optionally for the record type like in the other divisions.

In these cases the field descriptions of the Data Dictionary will be included in the data division at compilation time.

Examples:

```
- -D;  DEFINE KUNDEN
-      DEFINE CPGWRK 80
-      DEFINE CPGWRK TYPE 80
```

Note: The key word DD is contrary to the other divisions not necessary for the data dictionary processing in this division.

The entry DD behind a DEFINE FILE Definition is interpreted as record type DD.

About the internal solution: At DEFINE DD a filler will be generated, if the structure is incomplete. (For files with Directory Field Check).

Example:

```
      DEFINE DEMO
        F1              10   * field  1
          CPGFIL        10   * CPGFILLER
        F2              20   * field  2
        F3              15   * field  3
          CPGFIL        65   * CPGFILLER
        F4              20   * field 5
```

2. DEFINE Multiple

If the same field is defined several times in the Data Division, the error message '...double defined' appears. If you work in the data division with DEFINE structure, this case applies if a field is part of multiple structures. With DEFINE Multiple it is possible to define a field several times in the Data Division.

Note:

These multiple defined fields will be set internally on commentary and must not be part of a redefinition (overlay).

Example:

```
      DEFINE FILE  M;
      DEFINE FILE  TYPE xx M;
      DEFINE FILE  TYPE XX MULtiple;
      DEFINE HQTFC MULtiple;
```

Both structures contain the eight places alphafield DOKUM. Without MUL the compilation would abend with the message 'field name double defined'.

3. Standard values for missing field definition

If only field names are entered, the field definition is taken from the QDDS. Long field names are supported if DDL is a parameter in the options division.

Data Dictionary in the Input Division 2620

In the input division Data Dictionary structures are attracted with the addition of the key word DD to the record description.

Data Dictionary is supported for FILEs and FIELDs.

If a record type of a structure should be attracted, it has to be appended at the instruction with the key word TYPE.

Example:

```
- -I;  FILE  KUNDEN DD;
-      FILE  CPGWRK DD  TYPE  80
-      FILE  CPGKSD DD  TYPE  AA  LIST
```

With the key words 'List' or 'TEXt' you reach, that a commentary line will be generated to each field, that lists the describing text from the Data Dictionary.

HL1 data channels are also described in the Data Dictionary.Here is to differ, if the data channel shall be optimized or not. If no optimizing is wished, you enter the key word HS and afterwards the entries for the Data Dictionary. To optimize the channel, you omit the entry HS. CPG guarantees, that the control field CPGHIC remains in each case in the program.

For data structures, the key word DS is indicated before the entries for the Data Dictionary.

For field editions the key word TYPE can have different meanings. TYPE can stand for the record type of the Data Dictionary, but also for the selection of the field selection. For reasons of the clearness, the key word SELECT-type can be indicated for the field selection.

Examples:

```
-      FILE  KANAL  HS  DD  TYPE 02
-      FILE  STRUKT DS  DD
-      FIELD CPGCOM DD TYPE XS  TYPE PROGRAMM-5
```

Data Dictionary in the Input Division with Field Queries 2625

When using the Data Dictionary, the input can depend on the check of two signs of the input record.

Example:

```
- -I;  FILE KUNDEN  DD             01  1 C    0  2 C    1
-      FILE KUNDEN  DD TYPE 99     02  1 CHAR 9  2 CHAR 9
```

The syntax rule:
1. the complete Data Dictionary entries
2. the indicator or the character #
3. the query of one or two signs

Data Dictionary in the Output Division 2630

Structures of the Data Dictionary can also be attracted in the Output Division. The entries for Data Dictionary must follow directly after a FILE 'structure name'. A DD for Data Dictionary and eventually additionally a TYPE 'record type' for the record type is indicated like in the Input Division.

The other possible entries for the record description in the Output Division follow in the known sequence, that is described above.

For field editions, the key word TYPE can have different meanings. TYPE can stand for *the record type* of the Data Dictionary, but also for the *selection* of the field edition. From reasons of the clearness, the key word EDIT type can also be indicated for the field selection.

Examples:

```
- -O; FILE KUNDEN DD
-     FILE KUNDEN DD ADD NAME-OF-THE-EXCPT
-     FILE KUNDEN DD ADD ON 99
-     FILE KUNDEN DD     ON 01 AND NOT 02
-     FILE CPGWRK DD TYPE 80
-     FILE CPGWRK DD TYPE 80 ADD
-     FILE CPGWRK DD TYPE 80 ADD ADD
-     FILE CPGWRK DD TYPE 80     ON 01 AND NOT 02 UPDATE-CPGWRK

-     FIELD A  DD
```

Key fields marked with Y or T will be generated as commentary statements at DISK and KSDS files at the output for Update. For the addition the key fields are also generated as output fields.

Data Dictionary can be used for field editions. In this case the Edit Code, that is indicated in the Data Dictionary, is also considered.

Reference Structures at Data Dictionary Processing 2640

Analogousely to the reference file, as it can be indicated in the Data Dictionary (see manual CPG2 service programs), you can also work with reference structures in the input and output division.

With this processing mode the structure, that was defined with FILE 'file', will not be attracted but the structure with the name, that was additionally indicated with REF 'file 2' will be attracted.

Example:

```
- -I; FILE KUNDEN DD REF TEST01
-     FILE        DD TYPE 09 REF TEST01
-   :
- -O; FILE KUNDEN DD REF TEST01             KDUPD
-     FILE KUNDEN DD REF TEST01 TYPE 01 ADD KDADD
-     FILE        DD REF TEST01 TYPE 02 ADD KDADD
```

For the processing of the file KUNDEN, a structure will be read, that is made up from the structures TEST01 and TEST01, record type 09. The structure TEST01 is put out for the Update; for the addition, a structure will be put out, that is made up from the record type 01 and record type 02 of the structure TEST01.

The key word REF identifies the following named structure as reference structure. REF is always placed behind the key word DD or behind the statement of a record type with TYPE. Independent from the position in the instruction, a record type is always related to the reference structure.

The combinations with other entries will not be limited by the parameter REF.

When you omit the file name, you reach, that several structures can be composed under one file name. Without this possibility a complete record description would always be generated: When changing the record description, the read or output processing ends during the execution.

Data Dictionary and Optimizing Function of the CPG 2670

Normally the optimizing function is active in the CPG. That means, that all fields defined in the Input Division, which are not used in the further program, will be ignored by the compiler and so not be displayed in the compilation list.

This optimizing function can be canceled with the options parameter DEF. The parameter DEF defines all fields described in the Input Division explicitly.

Example:

The Options parameter DEF must be indicated, if data are read from a file and are put out on the screen with a QSF Map without processing.

In this case the fields are described in the Input Division, but not used again in the Procedure Division and in the Output Division. The fields would be ignored from the internal optimizing function and would so not be known by the QSF for the output. (With QSF only fields can be output that are defined in the program.)

Data Dictionary Lay Out in the Program List 2680

The listing of the Data Dictionary entries in the program is controled by the following Options parameters:

CPG The entry CPG for the indication in RPG like fixed notation implicises the listing of all attracted fields.

RPG See CPG.

DIC All coded statements are listed. The attracted fields out of the Data Dictionary are listed with a '-' in column 6 under their record description (only in the RPG like format).

ENT Entire Input. The whole input is listed. The Default Suppress is canceled, thereby input structures are listed completely; the fields used by the program, as well as the not used fields. ENT can be combined with DIC or MIX.

FRE Only the source program in the CPG2 format is listed. FREE is default and has not to be coded.

GEN works like CPG. Contrairy to the parameter CPG, not used fields will not be listed in the Input Division.

MIX Additionally to the source format, all generated statements will be listed in the RPG format, especially the attracted field descriptions from the Data Dictionary.

Example: Pecularity for the Output Division

```
- OPTIONS DIC;
:
- -O; FILE CPGWRK DD TYPE F2
-*                        KEY       14
-                         RECORD     100
-     FILE CPGWRK DD TYPE F2 ADD
-                         KEY       14
-                         RECORD     100
```

If an Options Parameter is chosen, where the attracted fields out of the Data Dictionary are indicated, the key field of the file will be listed as commentary statement at an Update in the Output Division, because it must not be updated for the file; For the adding, the key field or the (partial) key fields will be generated normally as field description. The attracted fields from the Data Dictionary are indicated in alphabetical sequence. With an entry in the standard header card, the system programmer can replace this sequence by the line convention rising sorting.

Test debugging 2800

QDF Quick Debugging Facility 2810

The interactive test debugging QDF is contained in the CPG2. It is described detailed in the manual 'CPG2..service programs'.

QDF replaces the test debugging operations DEBUG and SDUMP. The functions of this operations are implemented in the QDF.

Special Terminal Dump 2830

The operation 'SDUMP' contains the operation 'DEBUG' and can be called from every place of the program to be tested. In the second information line, the program address, the SDUMP code and the last used function key will be indicated contrairy to the operation DEBUG.

At first, following mask appears:

```
**********************************************************************
                            T E S T - A I D
**********************************************************************
INDICATORS ON
70 80


**********************************************************************
PROG.-ADDRESS 00000604            CODE 0152          FUNCTION KEY DE
**********************************************************************
```

After pushing the data entry key, a terminal dump with the following format appears:

```
Program = TST005   TCA                            05.03.93  15.28UHR
E         F         0         1           3         4         5
002DBAA0  0023F180  0000001C  00       0A  502DA80A  502DA80A  502DB80A
502DC80A  502DD80A  802DBBE6  00       00  002D8B08  0023F080  001F3E40
6         7         8         9           B         C         D

00000000  0023F000  00000000         000  0023F080  ..0.....  ........
00000010  00000000  002082C0         000  0023F090  ........  ....d...
00000020  701F6A46  0023F18         BAA0  0023F0A0  ......1.  ........
00000030  502D9B1E  502D980         6952  0023F0B0  ........  ........
00000040  502DC80A  502DD8         2F720  0023F0C0  ..H...0.  ...U..7.
00000050  011EBCA8  001EA2         23300  0023F0D0  ........  ........
00000060  402D9D72  0023F         2DBAA0  0023F0E0  ......1.  ........
00000070  402D9B1E  502D        02DB80A  0023F0F0  ........  ........
00000080  00220095  00         00000000  0023F100  ........  DV......
00000090  00000000  0          0023F739  0023F110  ........  ......7.
000000A0  502D9D00  0          002DBAA0  0023F210  ......1.  ........
000000B0  502D9B1E         0  502DB80A  0023F130  ........  ........
000000C0  502DC80A        E4  4022F720  0023F140  ..H...Q.  ...U..7.
000000D0  0022EF40       000  00000000  0023F150  ..?.....  ........
000000E0  0000000        000  00000000  0023F160  ........  ........
000000F0  000000        0000  00000000  0023F170  ........  ........

SDUMP = ...  +          PRINTER = ....
```

In the 1. line the phase name of the program (here TST005) and the area will be indicated, which the Terminal Dump accesses at first. For the first call is this always the Task Control Area (TCA).

In the four following lines the content of the register E,F,0,1 etc. to B,C,D are indicated, whereby the register number will be indicated on the left above or below the register content. In the example above means:

F the content of the register 15 (F) was before the entry 0023F180 in the SDUMP = 0023F180.

The following 16 lines each describe 256 bytes of the main storage in hexadecimal and character spelling. The columns 2,3,4 and 5 of this section indicate the storage content hexadecimally.

The columnn 7 and 8 describe the content in clear text, whereby all not printable signs with a hexadecimal value less than 'C1' are replaced by a point.

Column 1 of this section describes the relative address for each starting point, in the example above for the TCA address. Column 6 of this section describes the absolute main storage address to each relative address in column 1.

With the program function key PF8 or data entry, the next 256 bytes can be requested. Thereby can be paged forward/backward as wished in the Dump.

With the program function key PF7 can be paged backward.

The last line allowes to display the Dump from every place of the CPU. Thereby the programmer can go directly to the following areas:

ADR	address XXXXXX
CSA	Common System Area
CIO	CPG Input output Area
CWA	area Common Work Area
END	End Sdump
HLB	HL1 Library
IFC	Interface Com. Area
MBK	Central Routine Library
PRG	Application program

The positioning takes place with the entry of one of the literals beside at: SDUMP = ... and pressing of the data entry key.

```
PWA   Private Work Area
TCA   Task Control Area
TCT   TCT  User    Area
```

The programmer can indicate an offset for file inputs at + ... , for example TCA + 000100 is the Transaction Work Area.

If the literal 'ADR' is entered at SDUMP = ... , the following field (+ ......) must contain a valid address. This address, which can also lie outside the TP partition, will be the start address for the Terminal Dump after pushing the DE key.

For Printer = ... the destination Id of the online printers is indicated for the print with PF4.

Program function key:	DE	Page forward by 256 bytes.
	PF2	Reposition.
	PF4	Print.
	PF7	Page backward by 256 bytes.
	PF8	Page forward by 256 bytes.
	other	end SDUMP

Restrictions for the OPTIONS Parameter BIG and 12K 2920

The operations WRITE, DELET and UPDAT are not supported for VSAM files.

Restrictions for modules without Dataset Logic 2930

If the OPTIONS description of a HL1 module does not contain the parameter DAT or PWA (for dataset logic), then the sequential reading without key is not supported. Then the operations READ, READ-BACK READPAGE and READB-PAGE can only be coded with the indication of a keyfield.

See also the table 'maximum values in CPG programs', page 7040.

Section 3 Program Design 3000

Syntax 3010

The source code is entered in a 8O digit 'coding form'. It may be structured in two types:

Coding begins up from position 1

OPTIONS must stand in the first line of the program up from column 1 as key word. The entire program code will be read afterwards from position 1 up to position 71.

All further rules are described under 2.; note here, that only 71 positions are at the disposal for the code. That means, that all maximum values indicated below, must be reduced by 8; so:

Last read position:	**Column 71!**
Last position, from which a statement may begin:	**Column 64!**

Program code begins up from column 8

Position 6 remains free or contains a minus sign.

Position 7 remains always free.

The source code consists of single words, which are joined to several records or statements, separated by one or several blanks.

A line can contain one or several CPG2 statements. The end of a statement is indicated by a semicolon and a blank afterwards.

A record end sign is only necessary, if several statements are described in a line. An exception is the OPTIONS statement, for which a record end sign or the key word 'END' is requested.

The words within a statement are separated by at least one blank. The space between two words can be as long as the statement can be accommodated in a line. The end of a statement is indicated with a record end sign directly after a word (without blank), followed by a blank. If the following statement begins with a star, the rest of the line is interpreted as commentary.

A record or statement may not be extended over the line end, that means that the record end sign must stand in the same line as the first word of the record.

The last statement of a line may not begin behind position 72.

Mixed using of RPG like format and CPG2 format 3020

It is possibile to use 'free' and RPG like format mixed in CPG2 programs.

Note only, that a division must start either with a division indicator in the CPG2 format or with a card in the RPG like format.

Program Structure 3030

Dependent from the type of the instructions to be executed, the program will be structured into different areas (divisions). For each division an own grammar applies. A program can contain the following divisions:

OPTIONS	This area contains instructions for the compiler for particular tasks or environments.
FILES	This area describes the files used in the program. (abbreviation: -F) (Division indicator is optional)
DATA DIVISION or WORKING STORAGE SECTION	In this area the data fields (variables) used in the program are defined. (abbreviation: -D)
FORMS	This area describes the form control of a printer.
INPUT DIVISION	In this area is described, from where the data which are necessary for the program are taken. (abbreviation: -I)
PROCEDURE DIVISION	

This area describes, how the data are processed. (abbreviation: -C) At the end of the procedure division, the subroutines, which may be separated optionally from the main program with the key word SUBROUTINES, are coded. (abbreviation: -SR). Between the main program and the subroutines, may still be total computing regulations (see chapter 2270). These will be marked with the division indicators -L0 up to -L9.

OUTPUT DIVISION

In this area is described, where the data processed in the program shall be output. (abbreviation:-O)

The stated sequence is requested.

OPTIONS 3300

The key word 'OPTIONS' indicates, that the following key words are interpreted for the control of the compiler. The following key words can be lined up in any sequence, the single words are separated with one or several blanks. The end of the OPTIONS is indicated by a record end sign. Several lines can be used for the options.

Note:

If the phase name is not given by a Job Control statement, and should the Options parameter list consist several lines, then 'PHAse xxxxxxx' must be coded in the first line of the options.

Following key words are possible:

ADD x or **ADR** X. Particular addressing routine.

The registers are assigned dynamically. The programmer can enter a proper adressing routine into the program for particular applications, that were cataloged before under the name 'CPG*CADx' into the Source Library. The sign '*' stands here for the release suffix (for example 2 for 2.5).

The 'x' in the last position will be determined by the entry ADD x in the OPTIONS statement. If ADD 0 is entered, the copy book CPG*CAD0' will be inserted into the program.

So the programmer can extend the TWA from 4 to 8K, or take particular registers for example used in a subroutine out of the addressing.
Standard copy books are delivered for the following entries, that means that these values are valid for 'x' :

'**B**'	CPG internal.
'**C**'	CPG internal.
'**D**'	DL/I application programs.
'**G**'	CPG internal.
'**I**'	CPG internal.
'**J**'	CPG internal.
'**K**'	CPG internal.
'**L**'	CPG internal.
'**O**'	OS/390 assemblers.
'**P**'	CPG internal.
'**R**'	CPG internal.
'**U**'	PLT program with PWA using.
'**V**'	Com. -Level and DL/I-Dataset.
'**W**'	see 'V' and TWA size 8K.
'**X**'	CPG internal.
'**Y**'	CPG internal.
'**#**'	Command Level application programs (#=X'7B').
'**0**'	Com. Level maximum program size 20K and TWA SIZE 8K

'**1**'	Macro Level maximum program size 20K and TWA Size 8K
'**2**'	CPG internal.
'**3**'	12K TWA in Batch Programs
'**4**'	CPG internal.
'**8**'	PLT program with PWA using (Command Level).
'**9**'	DL/I using program with maximum prog. 20K and TWA size 8K.

Example: `'ADD D' = 'ADDRESS D' = 'ADDRESS DL/1'`

ASM x If the CPG source code will not be translated with //EXEC ASSEMBLY into machines language, but with another assembler procedure, so the suffix of the procedure name is indicated here instead of x.

It must be guaranteed, that an assembler procedure with the name CPGUASSx exists, which is generated as follows: // EXEC PROC=CPGUASSx.

Example: OPTIONS ASM-SUFFIX H for the procedure CPGUASSH

ASS x or ASSEMBLER LIST x is the type of the assembler list, where 'x' can accept the following values:

'**A**' assembler list without macro-dissolution.

The part of the assembler program, important for the processing is printed. TP dummy sections, macros and CPG subroutines are not printed.

'**C**' complete assembler program.

The assembler program generated by the CPG will be completely listed.

'**D**' assembler list with all Dummy sections.

As 'A', however all Dummy sections will also be listed.

'**M**' assembler list with macro dissolution.

As 'A', however the generated statements will be listed for every macro.

'**N**' no assembler list

The assembler program generated by the CPG is not displayed. Only the CPG compilation list including CPG- and assembler diagnostics will be printed.

'**S**' complete assembler program with short cross reference.

Like 'X' but with display of the assembler short cross reference (only VSE).

'**T**' Transaction Work Area.

From the assembler list, only the Transaction Work Area and the assembler diagnostics will be displayed.

'**X**' complete assembler program with cross reference.

As 'C', however additionally with listing of the assembler cross reference.
X can also be the first letter of a word.

Example: `ASSEMBLER LIST TWA`

ATT C or **ATTRIBUTES C**.

A 'C' causes, that the entries behind the key word ATT are interpreted in the output division as hardware attribute. So the entire hardware attribute set of the screen is available to the programmer, if for any reason the CPG attributes are not sufficent.

C can also be the first letter of a word.

Example: `ATT CICS`

AUT At program links an automatic RNDOM*ALL shall be processed.

BAT The generated program should be activated in a batch partition.

BIG The program is bigger than 24 K.

This parameter causes, that an own CSECT will be generated for each input data, for each subroutine, for the procedure- and output divisions and for the field processing.

<u>Following restrictions apply to this CSECTS:</u>

Per program at most 200 CSECTS are supported.
The CSECT for the field processing may include at most 4K; at most 400 fields can be transferred per field processing.
The main program (procedure division without subroutines) cannot be bigger than 8 K.
All other CSECTS can be at most 12 K.
The TWA size is restricted to 4 K, but can be increased to 8K with the parameter ADD (see above). see also 12K.

CAT or **CATAL**. OPTION CATAL- and phase card will be given with job control statements.

CIC or **CICS/CICSE**. The program works without the Central Routine Library of the CPG (CICS-version). The key word CICS generates a macro level program, CICSEsa a command level program.

COL or **COLUMN**. For any column of the generated H-card every sign can be set in the form **`COL 47 -`** or **`COLUMN 47 = '-'.`**

The **minus sign** in the example is set in order to 'clear' an entry of the standard header for the application.

COM or **COMMAND LEVEL**. The generated program shall be executed under CICS-command-level.

DAT for dataset logic. For HL1 modules, this entry can be intended, to obtain the PWA of the module within a task. This way of processing offers (beside performance advantages) the service, that the files division of such a module might not be completely contained in the associated main program.

DDL Output of the long field names from the data dictionary in the compilation list.

DDS Output of the short field names from the data dictionary in the compilation list.

DEB for **DEBUG**. If the Debug facility QDF should be used, the program must be compiled with the options parameter DEBug.

Note, that the program code with the parameter DEBug increases 8 bytes per statement of the procedure division.

DEC for **DECK**. An object deck is punched.

DEF for define. All fields described in data division and input division are defined for the program, independently if they are used in the further program flow. The optimization function of the CPG is switched off with DEF.

DIC The programmed statements including the data dictionary entries will be displayed in 'free' form (columns 6: '-').

END end of the OPTIONS.

ENT for entire input. A complete list of the input division will be indicated, not only the fields really used from CPG in the program. ENT can be combined with the options DIC or MIX.

ESA ESA mode able programs. See chapter 2970.

GEN lists the commands generated by the compiler and suppresses the listing of fields not used in the input division.

HL1 The index of the private HL1 Library can be entered for X. (=LIB)

INT for interruption. The parameter causes that a program break takes place in a batch program if there is any program error (cancel).

LAN X or **LANGUAGE XXXXX**. X can assume the following values: **A 'D', 'E', 'I', 'J'** causes, that all text output is in english. A **' '** or a **'G'** causes, that all text output is in german.

Each other sign causes, that a text phase cataloged by the programmer under CPGSX * applies, whereby the * is replaced. The entry has no effects on the presentation of the decimal signs.

LIB X For X, the index of the private HL1 Library is entered. (=HL1)

LIS X list processing. X can assume the following values: **E, F, L, N, O, P**.

E	error messages will be printed right beside the text.
F	a block diagram is printed
L	the line number or the left side is suppressed
N	the programmer check list is suppressed
O	the error messages are printed between the lines
P	to every generated statement the number of the required bytes will be printed.

LON long array names. LONG (or standard header column 100) offers the possibility, to work with array names of all lengths. To note is, that the name of the index field must be only one position. Also 5- and 6 digit array names will be moved internally into CPxx names.

LOW small letters are not translated automatically at screen input.

MAC a macro level program is generated (replaces an entry in the standard header, see also chapter 2970).

MAI or **MAIN** (only for HL1) A HL1 main program will be generated.

MAP The QSF maps are displayed in the compilation listing.

MIX The programmed as well as the generated statements will be listed.

MVS the program should work under OS/390 or MVS.

NON- NON-ESA command level programs (to overwrite the standard header, so that programs are executed with the CPGCLI). See also chapter 2970.

NOS NOSYSIN, enables the CPG compilation via the punch queue, so that IJSYS04 is not necessary. For this type of compilation, the job CPGZPUN, which is described in chapter 7000 is necessary.

OPT optimization for numerical operations. This entry refers to the numerical operations '+', '-' and '='. Optimize means, that for these operations the direct assembler source code is generated. The central routine library has not to be executed. This causes a considerable improvement of the running time especially in CPG batch programs

Note:

The parameters OPT and DEBug exclude themselves mutually. Only one of the two parameters is meaningful. CPG2 takes the last made entry.

PHA XXXXXX or **PHASE XXXXXX**. XXXXXX = Phase. Phases might be at most eight positions long.

PUN or **PUNCH**. An assembler deck is punched.

PWA to preserve the PWA. This entry for HL1 modules means, that the PWA is not released within a task, but will be reinitialized with each call (without initialisation: DAT). This processing type offers (beside performance advantages) the service, that the files division of such a module has not to be included completely in the corresponding main program.

QSF The QSF maps and the fields used by these maps will be listed in the refe-rence list.

QLF The LIST documents and the fields used by these documents will be listed in the compiler list.

QTF See QLF.

RDR or **REAder** to indicate the readers address (usually an entry in the standard H card CPGSTH.)

ROO or **ROOT** (only for HL1). A HL1 main program (root phase) shall be generated.

RPG lists only the generated CPG statements in RPG-like syntax), not the '-' cards coded by the programmer.

RUN assembly is also executed at CPG errors.

SEM or **SEMICOLON**. The end of a statement is indicated by a semicolon.
(Default in english version, where decimal places in numerical fields are separated by a full stop (controlled by the standard header, colum 93).

Note:

SEM must not be the last key word of the options.

SHA shared data. The entry for a HL1 module is taken, to exchange data with the calling program automatically. (that means, without using a data channel)

The data exchange takes place between fields, that have equal names and field definitions in both programs. The data is exchanged like in QPG, only up to the field CPGEDS, if it is defined in the module.

SHO as opposite to LONG (s.o.). With SHOrt, the compiler does not translate 5- and 6 places array names internally.

SIG or SIGN. Zone C for numerical fields (see chapter 2145)

SIZ XXX or **SIZE XXX**. XXX is the size entry for the compilation and must be 3 positions long.

SUB for subroutine. This parameter causes for batch programs, that the control is given back to a calling program at the program end (for example DL1).

TIT XXXXXXX or **TITLE XXXXXXX** sets the following title (XXXXXXX), that may be at most 23 positions long, as a headline onto all pages of the compiler list.
Blanks within the title have to be replaced by '#'. (#) = Number sign.

Example: `TITLE THIS#IS#A#TITLE`

TRA for trace. If the quick debugging facility QDF shall be used, the parameter TRAce may be indicated instead of the parameter DEBug. TRA causes, that only the numerical operations in the interactive test can be debugged. Advantage of TRA is, that the program code does not increase contrairy to DEBug.

TWA XXXX. XXXX = TWA size.
Should be branched from another program into this program, so the 4-places TWA size of the calling program is entered. If the program can be called from several programs, the TWA of the biggest calling program, must be entered. See also program links.

If only a part of the TWA has to be taken, so the decimal value from the TWA list, which is positioned behind the corresponding field, must be entered. (Minimum = 116 bytes, maximum = 7836 bytes). Numerical fields, which start behind the taken TWA area, will be initialized to X'0C'.

USE No automatic RNDOM*ALL shall be made at program links.

XRE or **XREF** prints a CPG cross reference list.

12K this entry causes, that a program, that is bigger than 24 K and has a TWA with a size of up to 12 k, can be generated. The **restrictions** of this parameter are equivalent to those of the parameter BIG.

Note, that 'TWA' must not be set as Dummy word, because the 'TWA' itself is a key word.

Connective words in the text are allowed, if they are not identical with key words. We do not recommend the related text, because it is possible, that in further releases these filler words get a meaning.

```
OPTIONS TITLE EXAMPLE THIS IS A COMMAMD LEVEL PROGRAM;
```

is identical to: `OPTIONS TIT EXAMPLE COM;`

Further examples for OPTIONS:

```
- OPTIONS    BIG RUN LIS E MIX TIT TEST PHA EXAMPLE2;

- OPTIONS    PHASE EXAMPLE3
-            COMMAND LEVEL
-            SEMICOLON
-            TITEL BUCHUNGEN
-            QSF
-            END

- OPTIONS    PHASE EXAMPLE3
-            ENT DIC                 * complete data dictionary
-            12K                     * 12 K TWA, no restrictions to pgm.-size.
-            TITLE BOOKINGS
-            QSF;                    * maps and fields are listed
```

Note:

The phase must be coded in the first line of the options!

Options and Standard Header CPGSTH 3350

A standard header card, which contains all standard informations for the environment, may be installed by the system programmer. These standards can be replaced by the options instruction if necessary.

Two peculiarities:

Exception: The entry for ESA compilations (CPGSTH column 47) has priority to the corresponding OPTIONS entry!

2.If there is no options key word to replace an entry of the standard header, so a minus sign must be set into this column to cancel the functions.

The minus sign with the parameter COL may be set like every other sign:
`OPTIONS COLUMN 46 = '-'.`

The following table indicates the relationship between the options key words and the columns of the standard header CPGSTH.

OPTIONS		Columns	Entry
Default	list in the CPG2 format	42	F
Default	list without display of the optimized fields	16	S
Default	HL1 module	51	H
ADD x	Addressing	48	x
ADR x	Addressing	48	x
ASM x	to use High Level assemblers	8-9	Ax
ASS x	extension of the assembler list	11	x
ATT x	attribute	49	x
AUT	automatic RANDOM *ALL	33	Y
BAT	Batchprogram	47	B
+	Batchprogram (is always a main program)	51	C
BIG	no restriction to program size	32	S
CAT	phase is defined by // OPTION CATAL	31	O
CIC	program works without Central Routine Library (Macro Level)	47	C
CICSESA	program works without Central Routine Library (ESA)	47	D

COM	Command Level program	47	L
DAT	Dataset logic in the HL1 module	34	*
DDL	long names from the data dictionary in the list	102	L
DDS	short names from the data dictionary in the list	102	-
DEB	Debugging with QDF possible	46	S
DEC	Punch object deck	10	C
DEF	define all input fields	16	*
DIC	Dictionary display	42	D
ENT	Display dictionary completely (ENTire DIC)	16	*
ESA	ESA Command Level Programm	47	E
GEN	generated statements	42	C
HLI x	Private HL1 Library	22	x
HL1 x	Private HL1 Library	22	x
INT	Interrupt in Batch Programs	46-47	IB
LAN x	Language Parameter	21	x
LIB x	Private HL1 Library	22	x
LIS x	processing of the listing	15	x
LON	long array names (also for 5-/6-digits)	100	A
LOW	upper-lower letters in taskoriented programs	39	L
MAC	macro Level program	47	M
MAI	main program (Main)	51	C
MAP	maps and QLF lists in the list	46	M
MIX	mixed listings of generated and coded statements	42	M
MVS	MVS-/OS/390 compilation	8-9	OS
NON-	non ESA mix mode	47	O
NOS	NO Sysin, compilation via puncher instead of IJSYS04	10	P
OPT	optimization of numerical operations	46	O
PUN	punching	10	D
PWA	Dataset logic in the HL1 module, PWA initialized	34	*
QLF	maps and QLF lists with documentation in the listing	46	N
QSF	maps and QLF lists with documentation in the listing	46	N
QTF	maps and QLF lists with documentation in the listing	46	N
RDR xxx	reader adress	12-14	xxx
REA xxx	reader adress	12-14	xxx
ROO	main program (Root Phase)	51	C
RPG	list in the CPG1 format (RPG syntax)	16,42	*
RUN	execute compilation even in the error case	50	A
SHA	Shared data (EXHM without data channel)	34	*
SHO	5- and 6-digit array names remain unchanged	100	-
SIG	Sign	40	S
SIZ xxx	Size	7-9	xxx
SUB	Batch subroutine	51	S
TRA	trace with QDF only at numerical operations	46	T
TWA xxxx	TWA length of the calling program	27-31	xxxx
USE	no automatic RANDOM *ALL (User does it)	33	N
XRE	cross reference in relation to the list	16	*
12K	no restriction to program size with up to 12K TWA	32	T
Default	For information:	31	O
Default	For information:	49	E

Files 3400

Before the first application, a file should be described at first in the data dictionary of the CPG2.

In this case the statements

```
     - FILE PLATTE
     - FILE PLATTE DD TYPE01
     - FILE PLATTE DD TYPE01 INPUT
```

are sufficient in order to define the file PLATTE.

If the file is not described in the data dictionary, it must be described manually by the programmer. The following parameters must be indicated in the requested order. Connective dummy words are not allowed on this occasion.

```
-------------------------------------------------------------------------------
               KW DN (IO) (MO) (FO) (BL) (SL) (KL) (FT) (FO) (SV) UN (BA)
-------------------------------------------------------------------------------
```

KW	= key word FILE
DN	= file name
IO	= in-/output mode
MO	= mode: queueing for Temporary Storage, buffer mode
FO	= record form: fixed or variable
BL	= block length
SL	= record length
KL	= key length
FT	= filetyp (KEY, RBA)
FO	= file organization
SV	= service: for example variable processing
UN	= unit
BA	= Batch extensions

For the different units, very different syntax rules are valid. Each file can be described according to the shown system, in that a '#' (number sign) must be set for the entries which are not used.

A shortened form should be used, that is described in section 3450.

Key word 'FILE'

FILE must be entered.

File name Name of the file defined in the TP file table. The name may be 7 positions long for VSE and 8 positions long for OS/390.

The first sign of the name must be a letter or a $ sign, for the further signs, letters, digits and $ signs are supported.

If a disk file is used for the first time, it must be described in the TP file table.

File type
Possible entries are:

'I' input file (or INPut)

'O'	output file (or OUTput)
'U'	update file (or UPDate)
'C'	combined file (or COMbined)/screen units for dialogue mode

These entries are even then supported, if a file is already completely described in the data dictionary. In this case you replace the entry predetermined from the data dictionary.

Temporary Storage processing mode

'**Q**' or '**QUEue**' is a peculiarity for the Temporary Storage processing. The entry causes, that the Temporary Storage can be processed 'record wise'.

The processing of queues and TS single records is different in the syntax. TS single records are not supported in every environment. If a queue has to be processed or provided with the single record logic, so the entry 'S' may be set.

Printer type (alternatively) (outdated)

The entry '**B**' or '**BUFfer**' causes, that the printer works in the buffer mode. The printer output is then described like a screen output.

Record format
Possible entries are:

'**F'** or '**FIX'**	for fixed record length
'**V**' or '**VAR**'	for variable record length (only for screens and VSAM files)

If the entry is missing, 'F' is taken for disk files and 'V' for screen files.

Block length
Disk file:

The block length for the blocked records must be entered into this field.
For a VSAM file, the double record length or a blank will be entered (in the CICS-FCT Recform=Blocked).

HL1 Dataset:

For datasets, this field remains free. (in this case, '#' must be entered.)

Screen:

The number of the screen lines is entered into this field. (for example 12, 24, 27, 32, 43, 50 or 62). The field record length must contain the length of a screen line.

Record length
For the disk file the logical record length must be entered.

For files with variable record length, the length, that corresponds to the longest record to be processed, must be entered.

For datasets, the length of the datasets must be entered.

On the screen, the length of a screen line will be entered. (outdated). The CPG compiler checks the first and last position in the input and output division for the file on the base of this length statement.

Key length

For KSDS files and datasets, the length of the (packed or unpacked) key must be entered in bytes.

For VSAM ESDS or RRDS files, 4 (bytes) must be entered.

Record addressing

For ISAM and VSAM files:

' ' KEY	ISAM and VSAM KSDS and RRDS
'K' KEY	ISAM and VSAM KSDS and RRDS
'R' RBA	VSAM ESDS

'K' will be entered for missing entries.

File organization

Possible entries are:

' '	ISAM -, VSAM - or DA-file
'I'	ISAM -, VSAM - or DA-file
'V'	VSAM file
'L'	VSAM file locate mode
'R'	Reuse. VSAM file is loaded again (only batch)
'AUX'	Auxiliary Storage (stored externally on disk)
'IND'	Independent Storage (terminal independent)

Processing mode

The entry 'V' or 'VARiabel' describes a variable file name, indeed only for

- Printers
- Transient Data
- Temporary Storage

If a 'VAR' is entered here, so the file name can be changed during the program execution. The standard value is the name indicated behind the key word FILE.

CPG internal fields, which are filled in the program with the alternative name, may be taken to change the file name

CPGDID 4-places alphanumerical for unit PRINTER
CPGTDI 4-places alphanumerical for unit TRANSDT
CPGTSN 8-places alphanumerical for unit STORAGE

Input/output device

Possible entries are:

DISK	disk unit (for all disk types and file organizations)
DISPLAY	screen (outdated)
DLI	DL/I data base
ESDS	VSAM ESDS file
HL1	HL1 Dataset (for users of the CPG3)
HL1DS	HL1 Dataset (for users of the CPG3)
KSDS	VSAM KSDS file
PRINTER	printer (only Batch)
PUNCHER	puncher (only Batch)
READER	reader (only Batch)
RRDS	VSAM RRDS file
STORAGE	Temporary Storage
TABLE	table (see operation FIND)
TAPE	record (only Batch)
TRANSDT	tranaction storage (transient data)
VBOMP	VBOMP (EDN) data base

Following units are not admitted in the batch processing

DISPLAY, TRANSDT

Extensions for the batch programming: (See also 'short form')

For the batch programming, advaced functions are supported, that can be indicated in any order behind the unit.

The meaning of the key words:

NO OPEN files with this entry are not opened automatically, they must be opened with the instruction OPEN explicitly for the processing in the program.

The entry is not supported for the units DL1, PRINTER, PUNCHER, READER, STORAGE and TABLE.

NO REWind tapes will be rewound usually after the processing. With NO Rewind, the tape is not rewound.

STAndardlabel This entry must be taken, if sequentially record files with file label are processed; otherwise the check of the TLBL parameter can be omitted.

SYSnumber	A six places SYS number may be entered for records. For printers, also a SYS number or SYSLST can be entered; with this entry, different print outputs may be produced at the same time in a program.
UNLoad	For the tape processing, UNLoad causes that a tape is rewound at program start to the tape start and is unloaded at the program end from the the record station.
UNOrdered	a VSAM KSDS file can be loaded unsorted with this entry or an UNSorted addition is possible. Note indeed, that this processing mode may interfere the performance considerably. Bigger files should be sorted and played into a KSDS file.

Examples for file descriptions:

```
- FILE KUNDEN
- FILE PSBNAM DL1
- FILE DR01 OUT FIX 132 PRINTER
- FILE DR02 OUTPUT BUFFER FIX 120 PRINTER
- FILE TS01 O F 500 STORAGE
- FILE TS02 UPD FIX 56 VAR STORAGE
- FILE TS03 UPD FIX 22 VARIABLE AUXILIARY STORAGE
- FILE TS04 UPD QUEUE FIX INDI STORAGE
- FILE STOR INPUT QUEUE FIX 1000 STORAGE
- FILE BILD O V 24 80 DISPLAY
- FILE TAB1 INPS FIX 19 9 TABLE
- FILE TD01 I F 560 VARIABLER TRANSDT
- FILE TEST01 OUT FIX 1000 500 05 KEY REUSE KSDS UNORDERED UPDATE
- FILE TAPEIN INP FIX 300 TAPE STANDARDLABEL SYS010
- FILE TAPEOUT O F 300 TAPE NO REWIND NO OPEN
- FILE LISTE2 OUT FIX 132 PRINTER SYS012
- FILE HQTFC UPD FIX 500 4 HL1
```

Peculiarities for DISK and the VSAM file types:

With the units ESDS, KSDS and RRDS, two numerical values are expected: the first value is the record length, the second the key length.

DISK is intended for sequential batch files. The first numerical value is interpreted here as block length, the second as record length. A number sign must be entered if a block length is missing.

If all entries are made like it is described above, you can also use the unit DISK for online processing:

```
- FILE PLATTE I F 1000 500 05 K I DISK;
- FILE PLATTE INPUT PRONTO 1000 500 5 KEY INDEX DISK;

- FILE SEQBAT OUT FIX 500 100 DISK;    * sequential batch file

- FILE DATEINS UPD VAR 500 05 KSDS
- FILE DATZWEI OUT VAR 256 04 RBA ESDS
```

Shortened Syntax 3450

Only the parameters, that are necessary for the definition of the corresponding unit, can be indicated in the following shortened syntax:

1. Only file name and unit must be indicated for DLI / DL1.

2. File name, input/output mode, fix or variable, the record length and the unit are sufficient for the units:

DISK	only for unblocked sequential batch files (!)
ESDS	
LU61, LU62	
L3286	**B** is also supported for buffer mode.
PRINTER	
READER	
RRDS	
STORAGE	The file definition for temporary storages can be extended with the entries for variable processing, auxiliary or independent processing and queueing.
TAPE	For unblocked tapes.
TRANSDT	

File name, input/output mode fix or variable, two length statements and the unit are sufficent for the following units. For which entries the two length entries are taken, is described in brackets:

DISK	Block and record length (for sequential batch datas)
DISPLAY	Block and record length
HLI,HL1(DS)	Record and key length
KSDS	Record and key length
RRDS	Record and key length
TABLE	Record and key length
TAPE	Block and record length (for blocked tapes)

Extensions for the Batch Processing 3455

Extensions for the batch processing, like NOREWIND, NO OPEN, etc, can be indicated directly behind the file name, if the file is described in the data dictionary.

```
- FILE TAPEOUT NOREWIND
- FILE TAPEOUT NO OPEN
```

The combination of the key words INPUT/OUTPUT and the batch extensions is supported, if the key word DD is indicated.

Example: `FILE CPGWRK DD OUTPUT NO OPEN`

Data Division (Working Storage Section) 3500

Fields and arrays are defined in the Data Division. The most simple form is the definition of an alpha field. It consists of the name of the field and the field length in bytes. The instruction...

```
- ALPHA 7;
```

... defines a field with the name 'ALPHA' in the length of 7 bytes as alphanumeric field. Several fields may be defined in a line.

```
- ANTON 5; BERTA 10; CHARLY 25; DORA 7;
```

For names, longer than six positions, see chapter 2100.

For the definition of numerical fields, the number of the decimal places separated by a blank, is additioned to the field length.

```
     - VALUE 7 2;
```

defines a numerical field named 'VALUE' with 7 positions, out of them two decimal places.

Arrays are defined in the form name, number of elements, in the form '*', length, or decimal places.

```
     - FG 10 * 5;
```

defines an alphanumeric array with 10 elements, all 5 bytes long, with the name 'FG'.

```
     - FN 10 * 5 2;
```

defines a numerical array FN with 10 elements. Each element has 5 positions and two decimal places.

TWA Overlay (Field redefinition) 3510

Overlay field (redefinitions) may also be defined. An overlay field is a field which is separated into further fields. It is defined like an array, 0 (zero) must be indicated as number of the elements.

Overlay fields may be numerical or alphanumerical.

The definitions of the fields, in which the area should be subdivided, must follow on every specification of an overlay field.

An overlay field can contain further overlays.

Take care that there are enough fields defined to fill the defined area for an overlay field. The block diagram can be taken as control.

Example:

```
     - ANSCHR 0 * 60
     - NAME       20
     - ORT        20
     - STR        20
```

The three fields NAME, ORT, STR, can also be processed, as a result of the overlay, under the common name 'ANSCHR'; while reading the field ANSCHR, the fields NAME, ORT and STR are filled.

Note at the overlay with numerical fields, that they are stored internally in packed form. Their length is not the length of the definition, but the length of the packed field in bytes. Example: VALUE 7 2 is four bytes long.

Overlay with ORG 3520

The key word ORG offers another possibility of the TWA overlay. In relation with a field name, ORG positions in the TWA onto the position, on which the stated field begins. If ORG is set without the field name, it is positioned behind the highest byte of the TWA, described up to now, that means, on the next 'free' position.

Example:

```
-  -D. F1 10;      * Byte  1 - 10
-      F2 10;      * Byte  11 – 20
-  ORG F1;         * Positioning on byte 1
-      F3 5;       * Byte  1 - 5
-      F4 5;       * Byte  6 – 10
-  ORG;            * Positioning on byte 21
-      F5 3;       * Byte  21 - 23
```

To make redefinitions more flexible, the end of a redefinition may be explicitly indicated with ORG-END. Behind the key word ORG-END must stand a field name already defined!

Example:

```
DEFINE STRUKT1
            ORG RECORD
              FELD1 10
              FELD2 20
              ORG FELD2
                FELD3 2
                ORG-END FELD2
              FELD4  15
```

Data Dictionary in the Data Division 3530

Structures, that are described in the data dictionary, may be defined in the data division with DEFINE 'structure name'.

If a record type of the structure should be chosen, it is attached to the DEFINE instruction (see example). The record type can be two bytes long.

Optionally the key word TYPE may be indicated for the record type like in the remaining divisions.

In this cases, the field descriptions from the data dictionary will be inserted into the data division during the compilation.

Examples:

```
- -D; DEFINE KUNDEN
-     DEFINE CPGWRK 80
-     DEFINE ARTSTA TYPE 01
```

If the same field in the data division is defined several times, the error message '...previously defined' appears. If you work with define dictionary structure in the data division, then this case appears, if a field lies in several structures. With define multiple it is possible to define a field several times.

Example:

```
DEFINE DEMO1
     F1         10    * FELD 1
     F2         20    * FELD 2
     F3         15    * FELD 3
 DEFINE DEMO2  MULTIPLE
     F4          8    * FELD 4
     F2         20    * FELD 2
     F5         12    * FELD 5
```

Note:

These fields defined several times will be set internally on commentary , that means, they must not be a part of a redefinition (overlay).

If only field names are entered, the field definition is taken from the QDDS. Long field names are supported in the options instruction in relation with DDL.

Furthermore, a FILLER will be generated in the data division (Define DD). (Only for files with Directory Field Check).

Example:

```
DEFINE DEMO
 F1             10   * FELD1
  CPGFIL        10   * CPGFILLER
 F2             20   * FELD2
 F3             15   * FELD3
  CPGFIL        65   * CPGFILLER
 F4             20   * FELD4
```

External Fields (key word EXT) 3540

The key word EXTern behind the field description identifies a field as external.That means, that the field is not taken over to the TWA of the program. So the programmer can access for example external fields in the CSA, with which he must guarantee that the stated field exists also under the same name and with the same length and type outside the TWA.

Adjustment on Word Borders 3550

Alpha fields can be adjusted at the definition in the TWA; the following keywords are available:

Double	for the adjustment on double word border,
Full	for the adjustment on full word border and
Half	for the adjustment on half word border.

The key word is attached to the definition.

Only one key word in relation to a definition is possible; the combination of the adjustment on word borders with the key word EXT for external fields is not supported.

Forms Division (printer control) 3600

For online printers, that work in the Line Mode (with line transportation and skip to channel), a FORMS description can be given after the Data Division.

The FORMS division starts with the parameter FORMS. Afterwards, the printer name of the files division follows.

The default form length is 72 lines (12 inches).If another form length is wished, the key word LENGTH has to be set in front of the wished line number.

The default for channel 1 is line 6, and line 69 for channel 12. For other skips to channel, all channel skips must be indicated in pairs in the sequence 'Line', 'Channel', with or without key words. The FORMS description must not exeed a line. The key words here have only descriptive meanings and can be chosen by the programmer.

Examples:

```
-  FORMS DR01;  * Printers DR01, 72 lines long, line 6 channel 1, 69 12
-  FORMS DR02 LENGTH 36; * Printer DR02 36 lines, Z 6 channel 1, Z 69 KS 12
-  FORMS DR03 LINE 3 CHAN 01 LINE 12 CHAN 02 LINE 66 CHAN 12; *Length 72
-  FORMS DR04 LENGTH 72 LINE 1 CHANNEL 1 LINE 69 CHANNEL 12;
-  FORMS DR05 LENGTH 36 003 01 066 12;
-  FORMS DR06 LENGTH 36 ZEILE 3 KANAL 1 ZEILE 66 KANAL 12;
-  FORMS DR07 LENGTH 36 LS 3 CS 01 LS 66 CS 12;
```

If no FORMS description is indicated, the default values are taken internally.

So the FORMS Division is optional.

Input Division (Syntax) 3700

In the input division, the input structures are described for the required data in the program. Here we distinguish two types of input descriptions: The record description and the field description. For each file, from which data are read, an input description is necessary. An input description consists of a record description, that is demanded necessarily as first statement of an input description, and field descriptions, that are normally inserted into the program during the compilation from the data dictionary.

Record Description 3710

It is distinguished between file- and field descriptions:

```
-----------------------------------------------------------------------------
     Files          KW DN (VAR) (BA)
     Fields         KW DN (TP)
-----------------------------------------------------------------------------
```

KW	= key word
DN	= file-, segment- or array name
VAR	= variable input positions
BA	= condition query
TP	= type, name of a certain field selection,

Key word (KW).

FILE	means, that the data are read from a file. A file is each type of external storage, from which data can be read, for example disk, screen, storage etc. With FILE, data can also be read from logical files, for example datasets, HL1 module, data bases etc.
FIELD	means, that the data are read from a field or an array in the main memory, which must be defined previously in the data division.
SEGM	segments are structures, that belong to VSAM files or HL1 datasets and are read with READI. The same syntax rules are valid as for FILE. Segments are always coded just behind the FILE instruction, to which they belong.

File name (DN)

The name of the file or the field or the array must be coded directly behind the key word. According to the key word, different rules are valid for the names.

FILE the name of the file may be at most eight positions long. For the

SEGM An exception exists for the names of the data structures, whose length is restricted to six places.

FIELD field names may be up to thirty positions long. Note the restrictions, listed in section 2 (fields, arrays) in particular for arrays which shall be processed indicatedly.

Specifying of field selections

If different possibilities exist for the field selection (key word FIELD), so a name in the record description may be given to select a certain input description. This name is added to the statement in relation with the key word TYPE.

The field selection can also be taken from the data dictionary. So the key word TYPE can have different meanings: It can stand for the record type of the data dictionary, but also for the name of the field selection. For reasons of the definiteness, the key word SELECT type can be indicated for the name of the field selection.

In the input record description, DEFine may be indicated in addition to the data dictionary key words. So is guaranteed, that all fields of the structure are defined for the program.

Example:

```
      -I;
            FILE DATEI DD DEF;

  or: -I;
            FILE DATEI KF 01 1 C A;
            FILE       DD REF DATEI TYPE XX DEF;
```

Variable input positions (VAR)

A special type of processing is the use of variable input positions. If a file should be processed in this way, the key word VAR must be entered directly after the file name.

At present, this processing type is an exception; it is used for example to process inputs for record lengths of more than 8000 bytes.

The moving factor is indicated in the internal field CPGFIS (see example in chapter 8000).

Condition query (BA).

The condition query is different for the different file types. The general form depends on the file type:

Disk, tape, reader	**(TP) (BD) (CA) (AND) (CA) (AND) (CA) with CA=PS (NOT) CD CH**
Dataset	**(TP) (BD) (CA) (AND) (CA) (AND) (CA)**
Screen	**(TP) (BD)**
Data dictionary	**TP (KW SA) (KW REF)**
Data structure	**TP (LG)**
HL1 data channel	**TP**
Field	**(KW) TP**

Meaning of the abbreviations:

TP	=	filetype / field processing type
BD	=	condition (switches 01 to 99)
SA	=	record type (with key word 'TYPE')
REF	=	reference structure (with key word 'REF')
CA	=	character query
PS	=	position in the record
CD	=	character digit zone
CH	=	character
LG	=	length of the data structure
AND	=	linking of several queries
NOT	=	inversion (condition is not fulfilled)

Filetype / field processing type (TP)

Usually, the filetype is optional.

Files any entry (for example NS = no sequence)

The indication of the type is requested in following special cases:

Data Dictionary	DD
Data structure	DS
DL1 data bases	DS (input over data structure)
HL1 data channel	HS
Field processing	any name, up to 30 positions

Condition (BD)

Here, a switch between 01 and 99 is set. For a condition query, the switch will be set if all queries are fulfilled. A switch is absolutely required, if a character query (CA) follows.

Example:

```
      - FILE PLATTE KF 99 1 C 0;  100 101 FELD2A
      - FILE PLATTE    98 2 C 0;  100 101 FELD2B
```

If the first character of the file is equal 0 in this case, data will be read from the file PLATTE into the field FELD 2A, but not into the field FELD2B. If a condition is fulfilled and a switch is set, the following record definitions of the same file are not processed.

Remedy:

```
      - FILE PLATTE KF   # 1 C 0;  100 101 FELD2A
      - FILE PLATTE KF   # 2 C 0;  100 101 FELD
```

In this case, the input switch is not necessary. A number sign must be entered instead. The input record definitions are now independent from each other, so that FELD2A as well as FELD2B can be read.

Character query (CA).

A character query always refers to a storage position in a record, that is queried on a certain value. That query can be repeated up to three times for different characters. It will be queried, if a certain character (CH) from the type (CD) is placed on a certain position (HP) or not.

PS is the up to four digit value of the position, relatively to the beginning of the record. PS = 1 for the first position in the record.

CD indicates, how should be tested:

C or Character	the whole character is tested.
D or DIGIT	only the digit part is tested.
Z or ZONE	only the zone part is tested.

CH indicates, on which value should be tested. For example 'A': does the position to be tested contain the character 'A'? (# for blank?)

NOT inverts the query, that means, the position to be tested contains any other character than 'CH'.

For a sign query, the parameters TP and BD are absolutely necessary.

Examples:

`2 C A`	position 2 contains the character 'A'
`1234 D 7`	position 1234 contains the digit seven
`1 NOT C 0`	position one does not contain the character zero

Length of the data structure (LG).

For a data structure, the full length of the data structure can be entered here, if the structure is only described partially in the input definition.

Examples for record descriptions:

```
- FILE BILD
- FILE PLATTE VAR
- FILE KUNDEN DD
- FILE KUNDEN VARIABEL DD 99 11 CHAR *;
- FILE PLATTE KF 11 1 NOT CHAR A 2 CHAR B 3 CHAR #;
- FILE ARTIKEL AA 01 1 C A AND 2 C R;
- FILE HUGO DS 200;
- FILE XKANAL HS;
```

Input record description in relation with data dictionary is described more extensively in chapter 2600.

```
- FIELD HUGO;
- FIELD AF Type SATZ1
- FIELD ALPHANUMERIC STRING;
```

Not allowed is:

```
- FILE LANGER-HEINRICH;            * File name bigger than 8 positions
- FILE ARTIKEL 1 C A AND 2 C R;    * Filetype and condition is missing
- FIELD HUGO AA 01;                * Filetype and condition are not allowed
```

Field description 3750

The general representation form of a field description is:

(SF) VP BP (DP) FN (SP) (BD) (BD)(BD)

SF	=	storage form (packed, binary, logical)
VP	=	from position in the record
BP	=	until position in the record
DP	=	number of digits
FN	=	field name
SP	=	key word selector pen indicator
BD	=	switches 01 - 99 or L0 - L9

Storage form (SF)

This entry is omitted for alphanumeric data fields. For numerical fields, an Entry is taken here, if the data are stored in one of the space saving packed forms.

P or **PACked**	the field is stored in packed form.
B or **BINary**	the field is stored in binary form.
L or **LOGical**	the field is stored logically packed.
O or **OPTimize**	the field is stored in packed form with even digit number.

Up from position (VP)

The position, at which the field begins, is entered here. The entry is numerical, up to four positions long and absolutely requested.

Until position (BP)

The position, with which the field finishes, is entered here. The entry is numerical, up to four positions long and absolutely requested.

Decimal Places (DP)

An entry in this position marks the field as numerical. The number of the decimal positions for numerical fields must be entered here. The entry is numerical and must not be bigger than a position.

Name of the field or the array (FN).

In this position, a valid field name must be entered (see also section 2, fields).

Key word selector pen indicator

SP is set for the selector PEN and marks a selector pen indicator (only for screen files).

Example:

```
- 0205 0279 PAGE SP 20
```

Indicators (BD)

Three input indicators can be indicated, which are set during the input, if the input field is greater (first BD), less (second BD) or equal zero or blank (third BD). The positions, that contain no switch, must be filled with number signs.

A group switch L0 - L9 may be set.

The three switch types (group switches, input switches and selector pen indicator) exclude themselves mutually.

Examples for field descriptions:

```
- 1 2 SA;                     * from 1 to 2 stands the field SA;
- 3 9 2 VALUE                 * from 3 to 9 stands the VALUE with 2 decimals
- P 10 11 0 RNR;              * from 10 to 11 stands the running No packed;
- 1 2 SA; 3 9 2 WERT; P 10 11 0 LNR.      * ;
- 0202 0279 PAGE SP 01;       * selector pen indicator (up from) 01
- 11 12 WGRP L2;              * Group changes L2 at the field WGRP
- PAC 1 3 2 NUM # 10;         * Switch 10 is set, if NUM is less than 0
```

Examples for Input Descriptions: 3790

```
      FILE KUNDEN DD;
```

All field descriptions are taken from the data dictionary. The input description for the file KUNDEN is so complete.

```
     - FILE KONTEN;                                    * manual description
     -                  1 5        KONTO
     -                  6 30       TEXT
     -               P 31 35 0     UMS;       * Array 12 fields from 31 to 90
     -                 91 92 0     MONAT

     - FILE KONTEN
     -      1 5 KONTO
     -      6 30 TEXT
     -      PACKED 31 35 0 UMS
     -      91 92 0 MONAT

     - FILE KONTEN; 1 5 KONTO; 6 30 TEXT; P 31 35 0 UMS; 91 92 0 MONAT;
```

Operations in the Procedure Division 3800

ADD	adding
AFOOT	average value of an array
BEGAS	switching to assembler coded program modules
BEGSR	subroutine start
BITON	set bits on
BITOF	set bits off
BREAK	finish a loop
CAB	compare and branch
CALL	call subroutine (other programming language)
CALLM	call routine from Central Routine Library
CAS	compare and branch subroutine (multiple alternative)
CBS	compare and branch subroutine
CHAIN	read record directly
CHECK	check file
CHANG	exchange fields
CLEAR	delete all fields to blank or zero
CLOSE	close file
CLRIN	clear switches dependent from an index
COMP	compare
COMRG	call communication region
CONT	loop interruption
CONVT	converting of a field
COPY	input of an assembler Copy Book
DEBUG	debugging aid or enable/suppress QDF
DELC	delete character from a field
DELET	delet a record
DEQ	release queue (from a locked program part)
DIV	divide
DLI	DL1 call
DO	process a program block
DSPLY	output of a console display
DUMP	print a dump
EDIT	alpha field output
ELIM	eliminate character
ELSE	otherwise
END	end of a program block
ENDAS	end of an assembler encoded program part
ENDDO	end of a DO block
END-EVALUATE	end of an EVALUATE block
ENDIF	end of an IF block
ENDSR	end of a subroutine
ENQ	lock a program part
EREAD	input/output of the screen
EVALUATE	multiple alternative
EXCPT	output
EXHM	execute a HL1 module
EXIT	exit to another program
EXITD	exit to another program with data transfer
EXITI	exit to another program with interval control
EXITP	exit to another program (like EXIT)
EXITS	send program (later) to another terminal
EXITT	exit to another program with screen data
EXPR	execute another program
EXSR	execute a subroutine
FILL	fill an alpha field with a character
FIND	searching in a data view

GETHS	retransferation of a HL1 datachannel
GETIN	get switches from the higher level (only CPG3)
GETMI	get switches from the highest level (only CPG3)
GOTO	branching
IF	IF block
INDOF	delete the switches from/to
INDON	set the switches from/to
JLB	move alpha field left adjusted, blanks behind
JRB	move alpha field right adjusted, blanks in front
JRC	move alpha field right adjusted, character in front
JRZ	move alpha field right adjusted, zeros in front
LIST	output of lists described program externally
LOADT	load back a saved terminal
LOKUP	look up in an array
MACRO	statement of an assembler command
MAP	read a QSF Map
MAPD	output of a QSF Map and read in the dialogue
MAPI	read a QSF Map in the dialogue
MAPO	output of a QSF MAP on the screen
MAPP	output of a QSF Map on a printer
MLLZO	transfer of first sign
MOVE	field content transfer (move right)
MOVEA	array content transfer (move array)
MOVEL	transfer field content left adjusted
MOVEN	transfer alphanumerical into numerical field
MOVEV	variable MOVE Operation
MULT	multiply
MVR	transfer rest of a division (move remainder)
OPEN	open file
QSSA	qualified SSA (DL1)
PARM	parameter transfer for the instruction CALL
PROGRAM	execute a QPG program
PROT	give a protection code (for CPG3 users)
PURGE	delete a Temporary Storage Queue
PUTIN	transfer switches to the higher level (CPG3)
PUTMI	transfer switches to the highest level (CPG3)
READ	read from a file
READB	read backwards from a file
READI	read file work area / screen data transfers
READP	read page
REPLC	replace signs
RNDOM	set file to random processing mode
ROLL	roll array
ROLLB	roll array backwards
SAVET	save screen content
SCAN	search for a character string in an alpha field
SDUMP	screen dump
SELCTS	select field
SETOF	set switches off
SETIN	set switches dependent from an index
SETIX	set index dependent from a switch
SETON	set switches on
SETLL	set lower limit (for file processing)
SORT	sort an array
SORTA	sort an array
SQRT	square root
SUB	subtract
SYNCP	set syncpoint
TAG	set a label
TESTB	test a bit
TESTF	test field for numerical signs

TESTN	test field for numerical signs
TESTT	test screen name
TESTZ	test zone
TIME	set time
TWALD	load saved TWA from Temporary Storage
TWASV	save TWA on Temporary Storage
UCTRN	upper case translation
UPDAT	change record
USSA	unqualified SSA (DL1)
WAIT	waiting
WHEN	characterisation of a condition (with EVALUATE)
WRITE	new record
XFOOT	calculate the sum of an array
Z-ADD	delete and adding
Z-SUB	delete and subtracting

Syntax 3810

An operation in the procedure division has four basic elements:

Condition query	**ON**
Execution of the operation	**OP**
Service query	**SV**
Set conditions	**BD**

To simplify, these elements are represented in short form in the following text. All other elements can be used optionally with exception of the basic operation. If an element is supported optionally, so it is represented in the text in brackets (ON). If an element is absolutely required, so it is represented without brackets, if it is not supported, so the abbreviation is absent in the description.

An operation can be generally represented in the following form:

(ON) OP (SV) (BD)

That means: The elements of the operation are described in the order, condition query (ON), basic operation (OP), service query (SV) and conditions (BD),whereby OP is requested while ON, SV and BD are optional.

The condition query (ON).

For almost all operations, the processing may depend from one or several conditions. If the condition query will be renounced, so the operation is executed in each case. If any conditions are queried, so the operation will only be processed, if all conditions are fulfilled.

Conditions can be numerical switches from 01 to 99, that are set by other operations, or program function keys of screens or other switches (for example EF for end of file, see chapter 2260, switches).

The condition query can be generally represented in the following form:

KW (NOT) BD (AND) (NOT) BD (AND) (NOT) BD

The indication is the same for all operations and will so be indicated for the description of the operations in the abbreviation(ON).

A condition query is always coded with the key word 'ON'. If this key word is missing and nevertheless conditions are indicated, so misinterpretations of the compiler occur, that lead to an abend of the compilation.

The key word 'ON' can be followed by up to three conditions, that are logically linked with 'AND'. To clarify this in the text, the programmer can insert the word 'AND'. BD stands for the two digit condition switch. If the operation should be executed, if a condition is not fulfilled, so the key word 'NOT' is to be set before the condition.

Examples for condition queries:

```
ON 11                                      at switch 11
ON P1                                      with program function switch 1
ON NOT 12                                  not at switch 12
ON 11 12                                   at switch 11 and 12
ON 11 21 31                                at switch 11, 21 and 31
ON 11 12 and 13                            at switch 11, 12 and 13
ON 11 12 and not 13                        at switch 11, 12 and not 13
ON NOT PB and NOT EF and NOT 10            not at PF11, not EF, not 10
```

The operation (OP).

For the operations, according to the type of the operation, different syntax rules apply, that are explained during the description of the several operations. This section restricts itself to the general elements of the operation.

An operation consists of the following basic elements:

Operations key word	OC	(Op. -Code)
extended OP.-Code	OE	
factor 1	F1	
factor 2	F2	
result	EG	
dummy words	DY	
operators	OK	

The general representation of the basic operation reads as follows:

OC (OE) (DY) (F1) (DY) (NOT) (OK) (F2) (DY) (EG)
or shortly: OC (F1) (F2) (EG)

Operands (F1 F2 EG)

The elements F1, F2 and EG are also named operands. An operation needs in each case the operation code, but not always all operands. Also the sequence of the operands can be different according to the operation. So EG or F1 can stand in front of OC for different operations.

```
  A MULT B C                             F1 OP F2 EC
  C = A * B                              EG OP F1 OK F2
```

Operators (OK)

At comparation operations, an operator, that is eventually supplemented by connective textwords, can stand between F1 and F2. **For example:**

```
      IF A = B                          OP F1 OK F2
      IF A EQ B                         OP F1 OK F2
      IF A IS EQUAL TO B                 OP F1 DY O K DY F2
      IF A IS NOT GREATER THAN B         OP F1 DY O K O K DY F2
      DO UNTIL A IS GREATER THAN B       OP OE F1 DY O K DY F2
      DO WHILE A = B                     OP OE F1 OK F2
```

Valid representation forms for operators are:

>	or **GT**	or (IS) **GREATER (THAN)**	greater than
<	or **LT**	or (IS) **LESS (THAN)**	less than
=	or **EQ**	or (IS) **EQUAL (TO)**	equal
>=	or **GE**	or (IS) **NOT LESS (THAN)**	greater equal
<=	or **LE**	or (IS) **NOT GREATER (THAN)**	less equal
><	or **NE**	or (IS) **NOT EQUAL (TO)**	unequal
<>			unequal

Examples for operations:

```
  EXCPT                         OP
  DO 10 TIMES                   OP F2 DY
  DO FROM X TO Y                OP DY F1 DY F2
  MOVE TEXT TO LINE             OP F1 DY F2
  X = 0                         EG OP F2
  X = A + B                     EG OP F1 OK F2
  READ OTTO                     OP F2
  MAP ARTIKEL                   OP F2
  START                         F1
  START TAG                     F1 OP
```

Service query.

The service query consists of a single key word. No limitations exist for service key words, that means, that they can also be used as names for variables. Valid key words are:

A	**AFTer**	**ARRay**	**BEFore**	**BLAnk**	**C**	**Sign**
CHEck	**CLEar**	**D**	**E**	**Extern**	**F**	**H**
HEX	**I**	**INDex**	**INPput**	**K**	**L**	**LEFt**
LOOP	**LOW**	**N**	**O**	**P**	**R**	**RIGht**
ROLl	**ROUnded**	**RUNden**	**S**	**SAVe**	**SEConds**	**T**
TIMe	**U**	**UPDate**	**V**	**Variable**	**1**	

It is enough to enter the capital letters (one or three positions). In case of doubt, the first position will be accepted as key word. Consequently, the CPG compiler detects possible errors.

Output Division 3900

In the output division is described, how and where the data fields will be output. We distinguish here 'record descriptions' and 'field descriptions', depending from eventually given units- or field specific descriptions.

Data Dictionary in the Output Division 3905

With the entry of the data dictionary in the record description, you reach, that no field descriptions must be coded. Data Dictionary in the output division is also described in chapter 2600.

Usually applies as syntax rule, that the complete data dictionary parameter will be indicated directly after the file name. The other key words are attached according to the rules indicated below.

Record Description for EXCPT Outputs (files of all type) 3910

The first statement of an output description is always the record description, that means, the indication of the file, to which the data fields will be transferred.

The key word 'FILE' always starts such an output description. The name of the file follows directly after the key word.

According to the type of the input/output unit, different key words can follow after the file name. In each case, the first three letters are sufficient and the described sequence must be kept.

Disk

The key word 'ADD' indicates in disk files, that a record has to be added, 'DEL' indicates, that a record is to be deleted. 'ALG' means, that the length is changed for files with variable record length.

```
  - FILE PLATTE DD          ADD
  - FILE PLATTE DD TYPE LX  ALG
  - FILE PLATTE             DEL;      * No field description at DEL
```

For files with variable record length, it is possible to determine via the record description of the output division, how long an output record has to be. With the key words ADD-VAR, ALG-VAR or with a VAR in relation to ADD or ALG, you reach, that the actual length of the output record is determined by the CPG internal field CPGVRL.

```
  - FILE PLATTE    ADD VAR  FALL1
  - FILE PLATTE    DD ADD-VAR FALL2
```

Temporary Storage

The entries 'I' and 'A', described as follows, should only be coded exceptionally. The two processing modes 'Independent' and 'Auxiliary' are normally already given in the file description (FILES DIVISION and data dictionary).

The key 'I' indicates, that the data can be read from the storage area by all terminals, and will be stored in the main store. The entry 'A' causes, that the output data can only be read again by the same screen and is stored in the auxiliary storage.

If none of these entries is made, the output data is stored in the main storage and can only be read by the output terminal.

Tapes

At the output on tape files, no additions like ADD, DEL and ALG are supported.

For tapes with variable record length, it is possibile to decide via the record description of the output division, how long a record should be. With the key word VAR you reach, that the actual length of the output record is determined by the CPG internal field CPGVRL

```
     - FILE TAPOUT VARIABEL;
```

Printer

For printer files, line transportations are indicated with SPAce. Both entries (for line transportation before and after printing) must follow after the key word.

```
- FILE DRUC SPACE 2 1
- FILE PRNT SPACE # 3
```

Channel skips are indicated with the key word 'SKIP', that is followed by a 2-digit literal for the channel.

```
- FILE DRUC SKIP 01 ON 11
```

The channel (in the example 01) must be described in the forms division, if the program is executed online, or if the program is executed in the batch and the printer contains no SYS number.

Screen. - (old processing type!) (not necessary when using QSF)

1. **UNF**ormatted output	see below.
2. **ERASE**	clear screen before the output
3. **U** or **UNP**rotected	clear all unprotected fields
H or **BEEP**	horn
M or **MOD**ified	do not release modified fields
K or **L, N, O, S** =	combinations according to the table 7030

Unformatted outputs define the output position relatively to the screen start. So line 2 column 40 becomes position 120 for instance.

Unformatted outputs are read in with SELECT CPGTIO.

In the files division, a record length must be indicated for an unformatted screen, that is as big as the biggest output position. For the screen, that has 24 lines and 80 columns, for example the record length must be 1920.

A peculiarity is furthermore, that the programmer himself is responsible for the construction of this output during an unformatted output without Erase (on an unformatted screen). Blanks have to be set for all not described positions.

Conditions

If the output only has to be output under one or several conditions, the instruction 'ON' followed by up to three condition switches must be coded.

```
- FILE BILD ERASE ON 01.
- FILE PLATTE ON 01 AND 02 AND NOT 03.
```

An up to 6-digit name may follow after the conditions, which can also be indicated instead of the switches. This name works exactly like the indicators.

Record Description for Field Processings 3915

Field processings are described in the output division with record description and field description(s).

Data fields can be edited in the main storage. Therefore, the key word FIELD is always used for the field. Field edits are not processed with the instruction EXCPT, but with the instruction EDIT.

The syntax is different to the file output: If you want to describe different output possibilities for a field, you work in the procedure division as well as in the output division with an EDIT name, that is indicated with the key word TYPE.

The field edit can be taken out of the data dictionary. The key word TYPE can have different meanings: It can be set for the record type of the data dictionary, but also for the name of the field processing. For reasons of the clearness, the key word SELECT-type may be entered for the name of the field processing.

Examples for record descriptions.

```
- FILE PLATTE
- FILE PLATTE DD ADD NEWRECORD
- FILE PLATTE DEL TAKEAWAY
- FILE PLATTE DD TYPE XY REWRITE

- FIELD DATUM
- FIELD CPGCOM TYPE PROG3
```

Field Descriptions 3920

If data dictionary is not used, one or several field descriptions to every record description must be coded. On the belonging record description follows a field output description for each field to be output. If it is a variable, it first contains the name of the field to be edited. Then follows the output position. For screen files, the position will be indicated in the form 'LLCC', where LL indicates the output line and CC the column within the line.

-	**FELD 25;**	the last byte of the variable 'FELD' is set into position 25
-	**X 1240;**	the last byte of the variable 'X' is set into line 12, position 40, if the file is a screen file.

Literals are locked up in inverted commas and are set behind the position statement.

```
- 542  'TEST'
- 1247 'THIS IS A TEXT'
```

If the field should only be output under one or several conditions, so the instruction 'ON', followed by up to three condition switches is set in front.

```
- ON 15 FELD 25
- ON 11 AND NOT 12 AND 13 1242 'TEST'
```

Note:
If a literal smaller than 99 shall be output under a condition into an output position, for example on 15 in position 25 the literal 'Test', the combination

```
- ON 15 25 'Test'
```

is not clear for the compiler, because 25 may be a condition as well as a position. In this case, a blank (#) has to be entered to separate conditions and position.

```
- ON 15 # 25 'Test'
```

For field descriptions, the following sequence is to keep absolutely:

1. Conditions introduced by 'ON' (alternatively)
2. Field name (alternatively)
3. Output position (necessarily)
4. Literal, pattern (alternatively)
5. Key word (alternatively)

Key words for attributes, edit codes, colors, EH-values, cursor etc, can be attached in any sequence. To the key words listed as follows, always the 3 first letters, for example ATT for ATTRIBUTES, are enough. Following key words are valid:

ATTribut	followed by a letter (see table, p.7020)
EDItcode	followed by a character (see table, p.7070)
Colors:	**WHIte, RED, BLUe, GREen, YELlow, PINk, TURqoise**,
EH values	**BLInk, REVerse, UNDerscore**,
Formats	**PACked, BINary, HEXadecimal, LOGical**,
Other	**CURsor, BLAnk, VARiable cursor**

The key word 'PAC' indicates, that the field is packed at the output. 'BIN' is set for binary packed output, and 'HEX' for hexadecimal output.

The key word 'ATT' behind the position statement indicates, that the following letter is a field attribute. Furthermore colour indications (BLUE, GREEN, PINK, RED, TURQOISE, WHITE, YELLOW) or extended highlight codes (BLInkend, REVersiv, UNDerlined) follow.

```
  - FELD 520 ATTRIBUTE A WHITE BLINK;
  - X    630 ATTR N;
```

The key word 'EDIT' indicates, that the following letter is an edit code.

The key word 'BLAnk' causes, that the affiliated field after the output will be deleted.

The key word 'CURsor' sets the position indicator at screen files into the firstposition of the field. 'VAR' or 'VARIABLER-CURSOR' will be entered as key word for the variable cursor.

The key word 'ARRay' enables, to describe not indicated arrays, (in that the compiler error message will be suppressed).

The key words LAST, CONFIRM and NOWAIT for LU6.2-connections are supported. For distributed transaction processing with CPG, a separate manual exists.

Note: principally applies 'inverted commas before key words'

For example for a hexadecimal output

```
          - 1017 '0000' HEX
```

For example for literals

```
          - 0211 'KDNR'
          - 0217 '.....' CURSOR ATTRIBUT N
```

<u>For example for patterns</u>

```
          - NUM 2366 '. 0, -' ATTR P BLUE
```

Examples:

```
       - FELD 110;
       - FELD 110 PACKED;
       - 2 'TEXT'
       - 5 '5C5C00001C' HEXADEZIMAL
       - FELD 110 EDIT2
- FELD 110 '*'  EDIT2
```

Data formatting Attribute Bytes for 3270 3950

A field on a screen of the type 3270 is defined as the area between two attribute bytes.

Before and behind each field and each literal, which are output on the screen, an attribute byte is set.

The output field as well as the literal always finishes on the indicated position. The closing attribute byte of the field is on the position behind the indicated position. The closing attribute byte has the characteristic 'protected, skip'. Through the indication of a following field as well as a literal, this attribute byte may be replaced (overwritten) with the initial attribute of another field.

The attribute standing before a field defines the characteristics of the field. If no attribute code is indicated, the field gets the standard attribute 'S' (protected, skip).

If no name of an output field is indicated, nor a literal, then only the attribute will be output on the screen.

The possibility to output several attribute bytes on the screen can be used for the following purposes:

a) define fields on the screen without the output of field data.

b) change the present attribute for a field on the screen.

c) set the cursor on another screen position (here an attribute byte in relation with the key word 'CURsor' is indicated).

If the attribute 'V' is indicated for a field or a literal, so the initial attribute byte for the field is taken out of the one digit field CPGATR. This must be defined in the program and be set on the corresponding hardware attribute code for the field, before the output operation takes place.

If 'V' is indicated and the output description does not contain any field name and no literal, so the content of CPGATR is output as attribute byte on the indicated position.

The CPG attribute code 'F' means, that the first byte of the field to be output, as well as the literals, should be valid as attribute bytes. It must be a valid bit combination for attributes for 3270.

Alphanumeric Data in the TWA 3952

Alphanumeric data can be output on the screen as literals or as variable fields. The length of the output data corresponds to the length of the field in the TWA as well as the length of the indicated literal.

IF in a literal '&'-signs as well as inverted commas are output, so they must be double indicated.

Literals and fields, that need several lines, can be output of the screen.

The maximum length of an alphanumeric field is 256 bytes.

Numerical Data 3954

Fields stored in the TWA are put out on the screen in character format. Before the output, the fields become unpacked.

The length of the output data on the screen corresponds to the TWA-length of the field (packed) multipled with 2 minus 1, that means, CPG2 reserves place for all digits, that the field can contain.

If a field is defined with a straight number of digits,an additional digit must be previewed.

CPG does not suppress the zeros in numerical fields, which are output in this way.

The suppressing of zeros is possible by the specification of a pattern for the field or by the entry of an edit code.

Patterns 3956

A pattern for the description of the output of a numerical field can be indicated just behind the output position in inverted commas. Patterns are used for the following purposes:

a) suppression of leading zeros up to the indicated position in a field.

b) the output of one or several characters right from a field, if its content is negative.

c) insertion of blanks left from the field.

The length of the data edited on the screen, corresponds to the length of the pattern (i.e. the number of characters between the inverted commas) plus 1. The additional character precedes the data and will be shown on the screen as leading blank.

Edited fields are processed from the left to the right. The digits of the field from the TWA will be set on the positions indicated in the pattern as blank or zero. If the field in the TWA contains more digits, than the pattern can take, data will be cut off on the right. Unvalid results are possible, if the pattern can take more digits than the field contains.

Suppression of zeros 3957

Through zeros in the pattern, the suppression of leading zeros is controled. All appearing leading zeros will be suppressed and replaced by blanks during the processing of the field, up to the position, on which is placed the zero in the pattern.

If there is a significant digit in the field in front of the zero, so the suppression of zeros finishes automatically.

Beside the suppression of leading zeros, this function also suppresses all insertion signs, which are specified as part of the pattern in front of the zero. If a significant digit is detected, so the suppression of the filling signs finishes at this position.

If there is no zero indicated in the pattern, so the suppression of zeros is valid up to the end of the pattern.

Insertion Signs 3958

/ Slash
, Comma
. Point are typical examples for insertion signs.

They are output like specified in the pattern, if they are not suppressed in the input data because of leading zeros.

A **'&'** in the pattern defines a space (blank) for the output. Within a pattern, the **'&'** must be set only once per blank.

Characterisation of negative Amounts 3959

The signs, which are indicated right in the pattern before the closing inverted comma, are only output, if the content of the field is negative.

If the content of the field is positive, one or several blanks will be output instead.

Typical signs for negative amounts are '-' and 'CR'.

Insertion of leading Blanks 3960

All input signs indicated left in the pattern are suppressed with the edit and are replaced by blanks.

With this function, blanks can be inserted during the output on the screen on the left side of a field.

Length of the processed Data 3961

The number of the digits from the field in the TWA, that are taken on into the pattern, corresponds to the necessary number of bytes for the field multiplied with 2 minus 1.

At fields, that are defined with a straight digit number, an additional left adjusted digit position must be intended, if a pattern is therefore defined.

Examples for Patterns 3963

A field, that is defined 9-digit with two decimals, should be processed per pattern.

Pattern	**values:**	**0**	**13.85**	**- 1234.56**
no one		"000000000"	"000001385"	"000123456"
' '			"13,85"	"1234.56"
' 0. '		"0,00"	"13,85"	"1234.56"
' 0. - '		"0,00 "	"13,85 "	"1234.56-"
' 0. - &!'		"0,00 "	"13,85 "	"1234.56- !"

Arrays 3965

From an array, alphanumerical as well as numerical data may be output to a screen.

If the whole array should be output, only its name must be indicated and the last position of the first element as output position.

The following elements will be set on the lower line on the same sign position, until all elements are on the screen.

If an output position is indicated, so that there are not enough lines on the screen for all array elements, that causes errors during the processing.

If a fixed index is indicated to the name of the array, for example FG(10), the output of single array elements is possible of course. In this case, each element can and must be output with an own output description.

For numerical arrays, a pattern can be indicated behind the position. The array is processed element par element according to this pattern. An element with the value zero causes, that the entire field contains only blanks at the output.

The place occupied by an element is just as big as at each other alphanumerical or numerical field with the same type and the same length.

Overlapped fields 3967

The output descriptions for a screen file will be processed sequentially from above downwards according to their sequence in the program.

So data can be output, so that they are overwritten by a following field or a literal. This does not lead to processing errors and can be meaningful, if complex screen forms are processed.

Field editing 3969

The field edit through the call of an EDIT operation in the procedure division is explained in chapter 2420.

The specifications for the field processing will be treated as own subroutines within the output division. They will be ignored while the processing of the output descriptions by an EXCPT instruction.

If the processing of an output description for the field edit is started, it will be branched to the output descriptions for the relevant field. These will be processed sequentially from above downwards, until the next record description is reached or the end of the source code. At this point, will be rebranched to the instruction in the procedure division that follows directly after the EDIT instruction.

Protection Star writing 3970

Protection star writing is used for example to write checks, to make an additional altering of the printed amount impossible. Therefore, the processed numerical amount field will be filled up left adjusted with stars (*).

In the CPG2, the protection star writing is supported additionally to the edit code. Only the literal '*' must be indicated instead of a pattern.

Example:

The field SUMME with the content 98765.55, is put down with the editcode K (suppression of zeros, 11-places, thousand-point, minus sign) and protection star writing. SUMME is agreed as comma numerical field, therefrom two decimal places.

```
Output:   * * * * * 98,765
```

Syntax: `- SUMME 1020 '*' EDITCODE K`

Peculiarity: The protection star in combination with the edit code Y for date-processing causes, that blanks are indicated at empty date fields, and that leading zeros are suppressed.

Flowing Monetary Signs 3972

Another possibility of the protection offer the flowing monetary signs. With it, a dollar-sign ($) is set directly before an amount so that a later possible manipulation of the amount is prevented.

The flowing monetary sign is supported in the CPG2 additionally to the edit codes. The literal '$' must be entered instead of a pattern.

Note: For the flowing monetary sign, the output field will be extended internally by one byte (for the dollar sign).

Example: The field SUMME with the content 98765,55 is put down with the editcode K (suppression of zeros, thousand point, comma, minus signs) and a flowing monetary sign. SUMME is defined as 11-places numerical field, therefrom two decimal places.

Output: `$98,765.55`

Syntax: `- SUMME 1020 '$' EDI K`

Operations

For the operations the following expressions are used:

DY	Dummy words to simplify the readability of the programm, the words: ALL BY FROM INTO IS OFF ON THAN THEN TIMES TO and WITH are reserverd.
EG	Result as field name or array with or without index,according to the operation.
FN	File name in file processing operations. The file must be defined in the FILES DIVISION.
F1	Factor 1 as field, array with or without index or as literal according to the operation.
F2	Factor 2 as field, array with or without index or as literal according to the operation.
OC	Operator for comparisons at DO WHILE, DO UNTIL and IF operations: Possible entries: > < = >= => <= =< >< or <>.
OP	**Operation code.**
SV	Service as complement to the operation, e.g. H or ROU for rounding.

Entries in brackets are optional and will only be entered as the need arises, e.g:

`EG = F1 * F2 (SV)` here the service (ROUnded) is optional.

Fields

for description of variable data, e.g.: KDNR, KEY, BETRAG, WERT. A fieldname may be max. 6 characters long.

Arrays

Arrays are supported in a part of the operations. Either only the name of the array may be given, if the whole array (element for element) shall be processed (Full Array support) -or- fixed or variable index may be given in brackets (indicated operations), e.g: FGR(10) or PAGE(X).

Literals

for alphanumeric values:	`'*', 'EF' or 'wrong switch'`
as hexadezimal value:	`X'00', X'1DF0' or X'FFFF`
for numerical values	`-10 123,45 or 10`

= assigning values

EG OP F2 (SV)

EG	Field name of the result field
OP	'='
F2	Field or literal
SV	'H' or 'ROUnded' for rounding with num. fields

Examples:

```
      X = 1
      A = B ROUNDED
      STARS = '*****'
```

Purpose:

Into the result field a literal or contents of a field can be transferred. If EG and F2 both are numeric, then EG is deleted before the transfer. If EG and F2 both are alphameric, then a transfer takes place as with MOVE-LEFT (see below). IF EG and F2 have different types, i.e. an operand is alphanumeric and the other one numeric, then a transfer takes place as with MOVE-RIGHT (see below).

+ addition

EG = F1 OP F2 (SV)

EG	Field name of the result field
OP	'+'
F1	Field or literal
F2	Field or literal
SV	'H' or 'ROUnded' for rounding

Examples:

```
      C = A + B
      X = Y + 5 ROUNDED
```

Purpose:

Into the result field the sum of the contents of F1 and F2 will be tranferred. The result can be rounded.

- Subtraction

EG = F1 OP F2 (SV)

EG	Field name of the result field
OP	'-'
F1	Field or literal
F2	Field or literal
SV	'H' or 'ROUnded' for rounding

Examples:

```
      C = A - B
      X = Y - 5 ROUNDED
```

Purpose:

Into the result field the difference of F1 to F2 will be transferred. The result can be rounded.

* Multiplication

EG = F1 OP F2 (SV)

EG	Field name of the result field
OP	'*'
F1	Field or literal
F2	Field or literal
SV	'H' or 'ROUnded' for rounding

Examples:

```
    C = A * B
    X = Y * 5 ROUNDED
```

Purpose:

Into the result field the product of F1 and F2 is transferred.
The result can be rounded.

/ Division

EG = F1 OP F2 (SV)

EG	Field name of the result field
OP	'/'
F1	Field or literal
F2	Field or literal
SV	'H' or 'ROUnded' for rounding

Examples:

```
    C = A / B
    X = Y / 5 ROUNDED
```

Purpose:

Into the result field the quotient from F1 and F2 will be transferred. F2 may not be 0. The result can be rounded.

With a division by 0 an error message (DEBUG) is displayed during the execution at the display.

ACCEPT Data record read directly

identical to CHAIN (see below)

AFOOT Calculate the average of an array

OP F2 EG (SV)

OP	AFOOT or AVERAGE must be entered
F2	name of a numeric array
EG	result field (name of a numeric field)
SV	service: 'ROUnded', 'RUNden', 'H' for rounding

Examples:

```
     AFOOT   FG1 MW
     AVERAGE FG1 MW ROUNDED
```

Purpose:

The average value of all fields not equal zero of a numeric array is to be calculated.

Description:

Factor 2 (F2) contains the name of the array. The result field (EG) contains the name of the field, into which the average value shall be stored. Fields with the contents of 0 are not included into the average value calculation.

```
        AFOOT  FG1  MW
```

Contents of the array:

FG1(1)	**125,00**
FG1(2)	**75,00**
FG1(3)	**85,00**
FG1(4)	**0,00**
FG1(5)	**0,00**
FG1(6)	**0,00**

Contents of the field MW
after execution of the instruction: **95,00**

AVERAGE Calculate the average of an array

identical to AFOOT (s.o.)

BREAK Terminate a loop

OP (SV)

OP BREAK must be entered SV **All**, to leave interlocked loops

Purpose:

DO -, DO UNTIL or DO WHILE loop is to be terminated.

Example:

```
     DO FROM 1 TO WERT WITH I
        :
        IF REST <= 0
           BREAK
        ENDIF
        :
```

```
     ENDDO
```

By the instruction BREAK a DO-LOOP is terminated immediately independently of the loop condition. BREAK branches out behind the ENDDO. The statements between BREAK and ENDDO are not executed any longer. BREAK ALL terminates the DO processing in interlocked loops, by branching behind the END of the outermost loop.

BEGSR Start Subroutine (not for QPG)

F1 OP

F1 name of the subroutine

OP BEGSR must be entered

Examples:

```
     - UPRO  BEGSR;               *subroutine UPRO
```

Purpose:

Beginning of an subroutine.

Description:

Every subroutine must start with this operation. F1 contains the name of the subroutine. At the end of the procedure division, all subroutines have to be placed in a subroutine section which may be started with a division indicator 'SUBROUTINES' or '-S.'

CALL Call of any subroutines

OP F2 (EG)

OP	CALL must be entered
F2	name of the subroutine in inverted commas
EG	numerical field without decimales for the return code

Examples:

```
     - CALL  'PHASE01'
     - CALL  'PHASE07'   RETURNCODE07
     - PARAMETER DATEI
     - PARAMETER FUNKTION
```

Purpose:

Any subroutine which is not written in CPG shall be called. So the communication with all software products which have a CALL-interface is possible.

Description:

The program to be called will be indicated in inverted commas. It will be linked during the compilation to the CPG program. A numerical field with zero decimal characters may be indicated optionally. The retun code will be transferred into this field while the return from the subroutine.

CHAIN Read a record in random processing mode

F1 OP F2 (SV)

OP	CHAIN or ACCEPT must be entered
F1	Field name of the key
F2	File name
SV	Service: 'U', 'Update', 'C', 'Check', 'P'

Examples:

```
KEY CHAIN DATEI
KDNR ACCEPT KUNDEN UPDATE
```

Purpose:

A record of a file is to be read in random processing mode.

Description:

With this operation a record of the random file specified in F2 is read with the key in F1. The length of the code must corespond with the key length indicated in the FILES division.

If no record is found, then 'NF' is transferred for NOT found into the internal field CPGFRC.

'C' or 'Check' in SV means, that the record is checked only for availability. In this case no data will be read.

'Update' or 'U' in SV causes, that the record will be blocked for a further CHAIN for updates, until an update is executed or the record is again released by a RNDOM. It will only be blocked on record level. With Share option 4 or Journaling the entire VSAM-CI is blocked.

A 'P' in SV includes Check and Update. The explicit specification of both services is likewise supported.

Note: If the file is in the sequential access mode, then CHAIN operates like the operation SETLL. It will be positioned only at the indicated key in the file, no data will be read.

CHANG(E) Exchange the contents of two fields

F1 OP F2

OP	CHANG or CHANGE must be entered
F1	Field name
F2	Field name

Examples:

```
A CHANG B;
OTTO CHANGE HUGO;
```

Purpose:

Contents of two alpha fields are to be exchanged.

Description:

With this operation the contents of two data fields can be exchanged.

```
     FELD1 CHANG FELD2
```

After execution FELD1 contains the value of FELD2 and FELD2 contains the value of FELD1. If FELD1 and FELD2 are redefined, then the result is unforeseeable.

Special case: FELD CHANGE FELD   * after the execution FELD has * the value x'00'.

CHECK   Check file status

CHECK-VAR   Check file status variable

OP FN

OP	CHECK or CHECK-VAR must be entered
FN	File name with CHECK or alpha field with CHECK-VAR

Example:

```
     CHECK DATEI;
     CHECK-VAR FELD;
```

Purpose:

The status of a file (open, closed) is to be checked, or whether the file is defined in the FCT.

Description:

In FN CHECK the name of the file to be checked will be entered and with CHECK-VAR the name of an alpha field, which contains the file name.

As result of the CHECK operation, the following informations will be transferred to the file status CPGFRC:

' '	the file is open.
'NO'	the file is not open.
'NF'	the file was not found.

In the CICS the file is not in the FCT and in the batch not in the main program and not yet opened by a module.

CLEAR   Clear data to zero or blank

OP

OP	CLEAR must be entered

Example:

```
CLEAR
```

Purpose:

CLEAR deletes the contents of all data fields of the program.

Description:

All alpha fields and arrays are filled with blanks and all numeric fields and arrays are set to 0.

CLOSE Close file

OP FN

OP	CLOSE must be entered
FN	File name

Example:

```
CLOSE FILE;
```

Purpose:

A file is to be closed.

Description:

The CLOSE operation will explicitly close a file. FN describes the file which shall be closed. The file status can be queried in the internal field CPGFRC. 'NC' means 'not closed' and 'NF' means 'not found in the FCT'.

COM-REG Communication region

COMRG Communication region

OP EG

OP	COMRG or COM-REG must be entered
EG	Name of a field that is 32 characters long

Examples:

```
COM-REG FELD;
COMRG FELD;
```

Purpose:

System information is put to the program .

Description:

The operation COMRG characters a communication region at the disposal, which enables the programmer the access to internal data of the TP-monitor. The communication region is an alphanumeric field that is 32 characters long.

From Column	To	Description
1	3	Operator identification (Sign On Table)
4	6	Cursor position numerically packed (170 = line 3, position 11)
7	10	Trans-id
11	12	Number of screen lines default (packed)
13	14	Number of screen columns default (packed)
15	16	Number of screen lines alternate (packed)
17	18	Number of screen columns alternate (packed)
19	21	Task number (packed)
		U' = UCTRAN Upper-case translation
22	22	'N' = NOUCTRAN (yes/no)
23	26	Reserved
		UCTRAN informations. 'N' = NOUCTRAN: no translation
		'T' = UCTRAN Transaction (starting at CICS 2.2)
27	27	'U' = UCTRAN on: translate into uppercase letters
28	32	Reserved

The result field must contain the name of a field that is 32 characters long, which takes up the communication area. The individual fields can be selected with a 'SELCT' operation or by redefining in the data division.

Example:

```
-D
       FELD 0 * 32;             *  All
                OPID   3;       *  Operator-Id
                CURSOR 5 0;     *  Cursor position
                TRANID 4;       *  Transid
                DZEILE 3 0;     *  Number os screen lines def.
                DSPALT 3 0;     *  Number of screen columns def.
                AZEILE 3 0;     *  Number of screen lines altern.
                ASPALT 3 0;     *  Number of screen columns alt.
                TSKNO  5 0;     *  Task number
                UCTRNB 1;       *  UCTRAN byte
                FILLER 4;       *  Reserved
                UCTRNE 1;       *  UCTRAN Byte extended
                REST   5;       *  Reserved
 -C
                COMRG FELD
```

COMPUTE   Calculation of formulas (QPG only)

With the instruction COMPUTE the formula translator is called. The formula translator permits the calculation of formulas and mathematical functions.

Examples:

```
COMPUTE   Y = ( A + B ) * ( C - D )
COMPUTE   A = D _ 2 * 3, 14159 / 4
COMPUTE   Y = SIN(X)
COMPUTE   WINKEL = ARCTAN( HOEHE / BREITE )
```

```
     COMPUTE   POTENZ = X _ Y
     COMPUTE   Z = LOG(X) * SQRT(Y)
```

With the calculations the floating point description is used in double accuracy (15 valid digits) internally. Apart from the basic operations (+,*,/) functions are available in a mathematical library. This contains at present the following functions:

Power (x high y)

ARCCOS	**Arcus Cosinus**
ARCSIN	**Arcus Sinus**
ARCTAN	**Arcus Tangens**
COS	**Cosinus**
COT	**Cotangens**
EXP	**Exponential function (e hoch x)**
LN	**Logarithmus naturalis**
LOG	**Logarithmus**
SIN	**Sinus**
SQRT	**Square root**
TAN	**Tangens**

With the trigonometric functions the arguments are given in degrees. With the inverse functions ARC... the result is returned in degrees. If values are indicated in the arc measure, the result has to be found out in the arc measure, then the conversion takes place easily with the following COMPUTE statements:

```
     COMPUTE   BOGEN = GRAD  * PI / 180
     COMPUTE   GRAD  = BOGEN * 180 / PI
```

The functions may not be used as variable names. With the operands no indicated variables are supported at the moment.

The calculation of the mathematical functions takes place internally with floating point operations of double accuracy (ca. 15 valid digits). Thereby a max. accuracy is achieved by suitable selection of the size of the result field.

Floating point operations are also used with multiplication and division to get more place with the result. Additionally the result becomes internally rounded.Thereby with COMPUTE the result can be more exact with conventional operation sequence, **<u>for example:</u>**

```
     COMPUTE X = ( 1 / 3 ) * 3;    * Result X = 1
     X = 1 / 3;                    * and
     X = X * 3;                    * X = 0,999999 (if X  has 6 decimals)
```

With an overflow (result > 15 integer digits) the processing will be interrupted. With call at the display the debugging aid appears (DEBUG).

<u>CONT(INUE)   Continuing a loop</u>

<u>OP</u>

OP   CONTINUE or CONT must be entered

Purpose:

A DO -, DO UNTIL or DO WHILE loop is to be continued with the next loop run.

<u>Example:</u>

```
     DO FROM 1 TO WERT
```

```
        :
        IF FELD = 0
           CONTINUE
        ENDIF
                             EDIT PAGE
     ENDDO
```

The loop will run through according to the loop condition. In the case FELD = 0 the operation EDIT will not be executed. CONTINUE branches back to the appropriate DO instruction.

CONVERT Converting an alphanumeric field

CONVT Converting an alphanumeric field

OP (F2 INTO) EG (SV)

OP	CONVERT or CONVT must be entered
F2	Name of the field, which shall be converted
EG	Result field (INTO must be coded)
SV	CHAracter, DATe, HEX, LOW, SEConds, TIMe, U, Year

Examples:

```
     CONVERT A1
     CONVERT A1 INTO A2  HEXADECIMAL
```

Purpose:

The contents of an alphanumeric field is to be converted. For the conversion of time and date informations also numeric operands are permitted.

Description:

CONVERT A1 translates in A1 lowercase letters into uppercase letters

The service functions have the following effect:

LOW (Lower Case translation) translates uppercase letters into lowercase letters. HEX translates characters into its half bytes (EBCDIC description), e.g. the letter A into c1. CHA combines two hexadecimal values to a character, e.g. C1 to
A.

Date-conversion (DATe, U and Year):

With DATE and Year the two operands must be of the same type and of the same length. The following lengths are supported:

```
numeric          6,0 and 7,0      Contents:    (0)TTMMJJ
numeric          8,0 and 9,0      Contents:  (0)TTMMJJJJ
alphanumeric     6                Contents:       TTMMJJ
alphanumeric     8                Contents:     TT.MM.JJ
                                      or  :     TTMMJJJJ
alphanumeric    10                Contents:   TT.MM.JJJJ
```

DAT exchanges year and day, so that the year is in the first characters of the field after the operation.

U converts a date of the ISO format jjjjmmtt into the standard format tt.mm.jjjj. The result field must be 8 or 10 characters alpha and factor 2 numerically (8.0 characters).

Y for Year is needed, if the year is located in the first four characters of the field. YEAR is the inverse function of DATE. (with two characters years the service function DATe is sufficient for both conversion directions).

Conversion: Time <-> Seconds (SECs, TIME of conversion).

The time is located in a field, which is defined numerically from 5,0 to max. 9,0. The two last characters are interpreted as seconds, the last but one as minutes and all further as hours. The max. number of seconds, which is supported during the conversion is 359,999,999 in a numeric field defined with 9,0.

SECs translates a time-of-day or a value (in hours, minutes and seconds) into the number of seconds. If the number of seconds or minutes is greater 59, then the result field is set to zero.

TIME translates a number of seconds into a current value.

The result field is optional.It must be available however with the services **HEX, CHA, SECS** and **TIME**.For the indication of the result field the keyword INTO must be indicated. During the processing with EG the origin field remains unchanged in F2. The compilation is executed left justified in the length of the shorter operand.

For the entries in F2 and EG applies:

- Literals are not permitted
- The operation is indicatable in both entries
- If both entries are to be made and one of them is a not indicated array, the other must also be a not indicated array.

DEBUG Program testing aid

OP (F2)

OP	DEBUG must be entered
F2	ON or OFF for switching DEBUG on or off

Examples:

```
DEBUG
:
DEBUG ON
:
DEBUG OFF
```

Purpose:

1. A test mode is displayed at the screen. (QPG only)

2. The possibility of testing with DEBUG is switched on or offfor a program section.

DELC Delete a character

OP F2 EG (SV)

OP	DELC must be entered
F2	the character which shall be removed as literal

EG	name of an alpha field or element of an array or the name of an alphanumeric array.
SV	A for alternative array processing

Examples:

```
      DELC '0' FIELD
      DELC X'FF' INPUT
      DELC '*' FG A
```

Purpose:

Delete a character from an alphanumeric field.

Description:

With this operation a character can be deleted from an alpha numeric field. All following characters are shifted one place to the left, into the last place a blank is inserted.

Behind the operation code DELC the character which can be removed is entered in inverted commas or hexadecimal in the form X'00'. Behind it the name of the field which shall be processed is entered.

Example : `DELC '*' FIELD`

Field beforehand: `'A*B**C'`
Field afterwards: `'ABC   '`

With arrays the entire array is regarded as a connected area, i.e. removing a character, then all following characters will also be moved together separately for each element. However if the service 'A' is indicated,then the array will be processed by element i.e. the shifting will be achieved only within one element of an array.

DELET(E) Delete a record

(F1) OP FN

F1	The key which shall be deleted
OP	DELET or DELETE must be entered
FN	Name of the file, whose record is to be deleted

Examples:

```
     KEY  DELET  CPGWRK
     DELETE KUNDEN
```

Purpose:

Delete a record of a file.

Description:

With DELET a record of a file is deleted, so that it cannot be found again.

Before the DELET a key field can be indicated, whose content indicates the record which has to be deleted. If such a field is not indicated then DELET deletes the record, which is in the moment in operating.

Behind the operation code is the name of the file from which the record is to be deleted.

DEQ(UEUE) Dequeue a program (not for QPG)

F1 OP (SV)

F1	sign, the same name how at enqueue (ENQ)
OP	DEQ or DEQUEUE must be entered
SV	possible entry: EXTern

Examples:

```
      -A100  DEQUEUE
      -X DEQ EXTERN
      -* see also: ENQ
```

Purpose:

Dequeue a closed part of a program.

Description:

A part of a program closed with ENQ, may be dequeued with DEQ. For dequeuing of closed external storage adresses the service funktion 'EXTern' is used.

Further description with examples for enqueuing see ENQ.

DISPLAY Console message

identical to DSPLY (s. u.)

DO Perform calculation specifications within a DO loop

OP (DY) (F1) (OC) (DY) (F2) (DY) (EG) (SV)

OP	DO, DO UNTIL or DO WHILE must be entered
F1	first matching field
OC	Operator (see below, table of the operators)
F2	second matching field
EG	Index. Name of a numeric field with 0 decimal characters
SV	Service: 'LOOP'
SV	Boolean operators AND and OR with DO WHILE or DO UNTIL
DY	Dummy word: 'FROM', 'TO', 'TIMES', 'WITH'

Examples:

```
     DO;                                * execute sequence of instructions
     DO 10;                             * execute 10 times
```

```
DO 10 TIMES;                    * 10 times
DO 5 TIMES WITH I;              * 5 times with index I
DO FROM X TO Y WITH I;          * initial value X, final value Y
DO LOOP;                        * continuous loop
DO LOOP WITH I;                 * continuous loop with index I
DO WHILE A = B OR               * as long as A = B or
DO WHILE C = D AND              * as long as C = D and
   WHILE E = F;                 * as long as E = F
DO UNTIL X = 0 AND              * until X = 0 and
         Y = 0;                 * until Y = 0
DO WHILE A EQ FGA(I)
DO WHILE A EQUAL B;
DO WHILE FGN(3) IS EQUAL TO ZAHL;
DO WHILE A IS NOT GREATER THAN B;
```

Purpose:

A sequence of statements shall be executed due to a comparison once or several times.

Description:

The operation DO makes possible to execute a group of statements (a DO group) once or several times.

F1 can be a numeric field without decimal characters, which contains the initial value, or with the extension 'UNTIL' or 'WHILE' a numeric or alphanumeric field or an element of an array.

F2 can be a numeric field or a numeric literal without decimal characters, which contains the delimitation value or with the extension 'UNTIL' or 'WHILE' any numeric or alphanumeric field or an element of an array.

Note: The field type of the two factors must be alike.

In EG, if 'UNTIL' or 'WHILE' were not indicated, a numeric field without decimal characters can be entered, into which the current index is stored during the execution of the operation.

A result field does not have to be specified. If the result field is missing, then the index is located in the field CPGDxx, whereby xx indicates the nesting level, which is indicated in the block-diagram of the compilation list.

If F2 is not specified, then the delimitation value is 1. In SV the keyword 'LOOP' can be entered,if the loop should be endless.

The sequence of instructions which can be processed must be locked by an 'END' or 'ENDDO'-statement.

Table of operators:

Operator			**Meaning**
GT	**>**		greater than F1
LT	**<**		less than F1
EQ	**=**		equal F1
NE	**><**	**<>**	not equal F1
GE	**>=**	**=>**	greater equal F1
LE	**<=**	**=<**	less equal F1

DO und IF can be nested up to 20 levels.

With boolean operators AND and OR with DO WHILE or DO UNTIL the continuation line can be continued with DO WHILE, DO UNTIL or only with WHILE, UNTIL or only with the comparison.

DO UNTIL-DATe, DO UNTIL-DATI, DO WHILE-DATe and DO WHILE-DATI permit the query of date fields in the standard- (-DAT) or in the ISO-format (- DATI). With AND/OR date queries can be combined with the 'normal' DO UnTIL or DO WHILE queries. See also IF-DAT and IF-Dati.

DSPLY Output to the operator console

F1 OP ONLINE and BATCH

(F1) OP (EG) only for BATCH

F1	Name of the field, which contains the message
OP	DSPLY or DISPLAY must be entered
EG	Name of the field, which contains a message/response

Examples :

```
MSG  DSPLY
MSG1 DISPLAY RESP
```

Purpose:

Output of messages on the operator console. With batch also receipt of responses of the console.

Description:

With this operation a message can be send to the operator-console.

The message is located in an alphanumeric field, which may be max. 255 characters long (in batch 64 bytes). In batch this operation is more flexible.

The field name for the message can be before and/or behind the operation code. If two message fields are indicated, then both fields are displayed on the console.

With batch operating DSPLY waits for the input of the operator, if for EG a field was indicated. The 'response' of the console is transferred into the result field to the program.

EDIT Edit alphanumeric field

OP EG (KW FN)

OP	EDIT must be entered
EG	Name of the field which shall be edited
KW	Dummy word: TYPE
FN	FIELD name in Output Division

Examples:

```
EDIT  FELD;
EDIT  FG1(J);
EDIT  ZEILE TYPE POSTEN
EDIT  CPGCOM;         * Output Common Area (only Command Level)
EDIT  CPGTCT;         * Output TCT User Area (only ESA Mode)
```

Purpose:

Edit a field using the output division.

Description:

With this operation alphanumeric fields can be edited.Edit branches to the output division where the editing of the field is described. If TYPE is indicated, then the field edit with the appropriate edit 'name' is executed in the output division.

With EDIT CPGTCT during execution in the CICS the length of the TCT user area is checked. If this area is missing or too small, the program abords.

ELIM(INATE) Replace a selected character with a blank

OP F2 (DY) EG

OP	ELIM or ELIMINATE must be entered
F2	character which can be removed as literal/hex-literal
DY	Dummy word FROM
EG	Alphafield / -array / -element of an array

Examples:

```
ELIM '_'  FELD
ELIMINATE  X'00' FROM INPUT
```

Purpose:

Delete a character from an alphanumeric field and replace it by a blank.

Description:

With this operation a character in an alphanumeric field, in an array or in an element of an array can be replaced by a blank. The character which can be replaced is indicated either as one-digit literal in inverted commas or as hexadecimal literal in the form X '00'.

Example : `ELIM '*' FELD`

Field beforehand:	**'A*B**C'**
Field afterwards:	**'A B C'**

ELSE Indication of a program block in the IF instruction

OP

ELSE

The operation ELSE indicates the program branch, which will pass through within a IF group, if the condition is not fulfilled in the IF-operation.

END End of a program block

The syntax rules of END depend on the 'Beginning' operation.

See DO and IF for detailled description.

END-EVALUATE   End of an EVALUATE block

identical to ENDEV (see below)

ENDDO   End of a DO-block

OP (F2)

OP ENDDO must be entered
F2 increase value as a numeric literal or field.

The operation ENDDO can be used in place of an END as conclusion of a DO block (for example for the better documentation with nested DO and IF blocks). In F2 a numeric value can be indicated as a numeric literal or field, by which the DO-LOOP is increased after each run. The default value is 1.

ENDIF   End of an IF block

The operation ENDIF can be used in place of an END as conclusion of an IF block (for example for the better documentation with nested DO and IF blocks).Differences to END do not exist.

ENDEV   End of an EVALUATE block (QPG only)

The operation ENDEV or END-EVALUATE terminates an EVALUATE group. Example, see EVALUATE.

ENDPR   Terminate a program (QPG only)

ENDPR

ENDPR is used in order to terminate a program prematurely. Then the control is returned to the calling program.

ENDSR   End internal program subroutine code (not for QPG)

(F1) OP

F1	label
OP	ENDSR must be entered

Purpose:

Ending an internal program subroutine code

Description:

In this operation the program branches back to the calling instruction EXSR or PERFORM.

In front of the ENDSR a label may be putted e.g. to branch direct by to a program subroutine end with a GOTO.

ENQ(UEUE)   Enqueue a part of a program (not for QPG)

OP F1 (SV)

OP	ENQ or ENQUEUE must be entered
F1	name of the label up to which shall be enqueued
SV	'EXTern' (See below, example)

Examples:

```
      - ENQ A100
      - ENQ C1 EXTERN
```

Purpose:

Enqueuing a part of a program

Description:

CPG programs are reentrant, that means that a program may be used at the same time from different terminals. That means that a disk record may be changed in the same time from different places.

The operation ENQ allows the programmer to enqueue the using of a part of a program between reading or backwriting of a sentence, until the update process is finished.

The enqueuing takes place with the operation DEQ.(See above) Per the instruction   -ENQ X the program will be enqueued for other users as long as the current user has reached the place X (that means the instruction –X DEQ).

The service function 'EXTern' causes that on a particular place in the memory capacity, the information 'in the moment enqueued' in the CWA accessible from all programs may be filed or called.

Example:

If in the user copy book GPGUCCUA the fields

```
C1    DS    C
C2    DS    C
C3    DS    C
```

will be defined, a system wide protection may be reached in the program; on a 1-byte place C1 in the CSA (common system area) may be filed the information per ENQ C1 EXTERN that C1-parts of programs able to be closed are for the moment enqueued.

Program 1:

```
      -    ENQ  C1  EXT
            :
      - C1 DEQ      EXT
```

program 2:

```
-    ENQ  C1  EXT
      :
- C1 DEQ      EXT
```

EVALUATE Multiple alternative

OP EVALUATE must be entered

Purpose:

The EVALUATE operation is used, if (at the most) one of several alternatives shall be executed.

Description:

As operation code 'EVALUATE' must be entered. The alternatives are marked by 'WHEN' continuation lines.The factors for the conditions can contain numeric or alphanumeric fields, field names or elements of arrays. If a condition is fulfilled, the following statement will be processed. Afterwards the EVALUATE instruction is finished and the program is continued behind 'END-EVALUATE'.

The condition 'WHEN OTHER' is fulfilled, if none of the beforehand WHEN-instructions were applicable. The appropriate instructions in this case are executed and the EVALUATE is finished.

Example:

```
-C
:
EVALUATE
   WHEN X = 1
      :               * 1. Case
   WHEN X = 2
      :               * 2. Case
   WHEN X = 3 OR
   X = 4 AND
   A = 'ABC'
      :               * 3. Case
   WHEN-DAT VDAT < UDATE OR
   WHEN-DATI XDAT > CPGDAI AND
   WHEN A = 'XYZ'
      :               * 4. Case
   WHEN OTHER
      :               * otherwise
END-EVALUATE
```

EXCPT Execute output specifications

OP FN

OP	EXCPT must be entered
FN	up to 8-digit file name

Examples:

```
EXCPT STOR;          * Storage
EXCPT KUNDEN;        * Update Kunden
```

Purpose for CPG: The output shall be executed.

Description: for CPG

This operation corresponds to a branch into the output-division. In the easiest case (-EXCPT.), all outputs for data will be executed.

If the EXCPT-operation contains a name or indicators, all outputs which are locked with this names or indicators will be executed.

Note:

All outputs which are not locked will also be executed.

Examples:

Procedure division: `- EXCPT ASATZ`
:
Output division: `- -0; FILE ARTIKEL ON 25`

Purpose: for QPG

The output definitions for the file shall be executed.Hereby a record in a VSAM file can be modified or an output on Temporary Storage can be executed.In OUTPUT files with EXCPT a record is added, with UPDATE files a record will be modified.

Description: for QPG

The output is executed, which is described to the file in the output division.

```
     OUTPUT DIVISION
            FILE KUNDEN
                          100 'X';         * Update 'X' position 100
```

Note: EXCPT is supported only for compatibility to the CPG and should be replaced if possible by WRITE and UPDATE. The parameters ADD, ALG and DEL in the OUTPUT division are not supported. EXCPT is not permitted with HL1 Dataset.

EXHM Execute HL1-Module

OP F2 (EG) (SV)

OP	EXHM must be entered
F2	Name of a HL1-Module
EG	Name of a HL1-Data channel
SV	I : initialize module, T : I + PF-keys

Examples:

```
     EXHM  HB0001
     EXHM  HB0002  Channel
     EXHM  HB0003  T
     EXHM  HB0004  Channel I
```

Purpose:

A HL1-Module is to be executed.

Description:

In F2 the name of a module, which must be contained in the HL1-table, is indicated.

EG can contain the name of a data channel, which must be described in the input division. With the call of the module all fields described under the data channel are transferred into the private TWA (Transaction Work area) of the module; after its execution the changed contents of these fields are transferred back into the appropriate fields of the calling program.

The service 'I' or INIT causes that the flow is continued, if the Clear key was pressed in a diolog-oriented module.

Service 'T' contains service 'I'.Additionally program function keys can be queried.

EXHM-VAR HL1-module processing (only for users of CPG3)

(ON) OP F2

OP	EXHM-VAR must be entered
F2	name of a 8-digit alpha field

Example:

```
      - EXHM-VAR XHMFLD
```

Purpose:

A HL1-module shall be executed

Description:

In F2 the name of a 8-digit field will be indicated.It contains the name of the processing module in the first six characters and, should the occasion arise, the private HL1-library in place 7. Place 8 rests free.

If at variable EXHM datas has to be changed between the calling and to be called module, so the called module must take its data from the calling module.

Therefore the entry EHA for 'Shared data' must be put into the options of the called program. Then the data between the fields with the same name and the same field distinctives will be exchanged.

EXITD Start other transaction with data transfer

OP EG (SV)

OP	EXITD must be entered
EG	name of a data structure
SV	'T' for time (instead of time interval)

Example:

```
      EXITD  STRUKT
```

Purpose:

Start another transaction with data transfer. For example EXITD will be used to activate printer tasks.

Description:

The operation is an extension of the operation EXITI.

The basic difference is the fact that with EXITD the program will not be leaved.

In EG the name of a data structure is entered, which must be described in the Data Division as follows:

```
-D
  STRUKT 0 * 88;      * DS: can be up to 256 characters long.
      TRANID  4;      * 1-4 Trans-Id of the following task.
      TERMID  4;      * 5-8 Terminal-Id, at which the following
                      * task is started.Default: same terminal
                      *
      ZEIT    7 0;    * 9-12 Time or interval at the point of
                      * time of the instruction.Indicates, when
                      * the following task is to be started.
      TSNAME  8;      * 13-20 Temp.storage name. If blank, the
                      * name will be assigned by the TP-monitor
      INTERN  4;      * 21-24 will be filled internally by CPG.
      DATEN  64;      * 25-88 data which can be transferred.  -
                      * The user dates can be max.232 byte long.
```

The following task can read the transferred data with a READ operation on '$CPG', an CPG-internal Temporary Storage queue. If no data are available, 'EF' is set. If no READ takes place on $CPG, then these data are deleted automatically with the end of the task.

The service function 'T' indicates that in the characters 9 to 12 a fixed time of the data structure in the form 0HHMMSSC will be transferred. The alternative is a time interval of the EXITD instruction, e.g.20 for a start of the following task after 20 seconds or 230 for an interval of 2 minutes and 30 seconds.

The internal field CPGFRC will be filled with 'EF', if either the required task or the addressed terminal is not defined in the appropriate CICS tables, otherwise CPGFRC will be filled with ' '.

The following task can read the transferred data as follows:

```
-I;
   FILE $CPG;
                  1  54 DATEN
-C;
   READ $CPG
   .
```

Under $CPG a pseudo TS queue is read; if it is missing, EF is set.

<u>EXITI   Call another program per interval control</u>

<u>(F1) OP F2 (SV)</u>

OP	EXITI must be entered
F1	Terminal-Id at that the task is to be started
F2	Transaction in inverted commas or as variable
SV	N if the current task is not to be left

Examples:

```
     EXITI 'QTF'
     EXITI TRID
     'DR01' EXITI TRID N
```

Purpose:

Call another Transid per interval control.

Description:

F2 contains the transaction identification of the program which shall be executed either as four-place literal in inverted commas or as variable alpha field. In the second case the programmer is responsable for the fact that the four places defined field contains a valid transaction at execution time of EXITI.

EXITI branches immediately to the following program.The called program does not read data at the start from the screen. If the service function 'N' is set, the branch is not immediate but at the end of the running task.

EXITP Calling another program (not for QPG)

OP F2 (SV)

OP	EXITP must be entered
F2	transaction in inverted commas or program name
SV	'SAVe' TWA to temporary storage

Examples:

```
     -EXITP TSTO24
     -EXITP 'PA36' SAVE
```

Purpose:

Calling another program.

Description:

This operation calls up another program, defined in the program list of the TP-monitor.

1.possibility

In F2 the name of the program in which shall be branched stands as literal (inverted commas). In this case the task rests preserved, only the program will be exchanged.

If datas have to be adopted, they are to describe identically in the data divisions of both programs beginning at the first defined field.

Remark that no loop arises because that might cause considerable performance losts.

This operation is identical to the 'EXEC CICS XCTL'.

The service funktion 'SAVe'causes that the TWA is saved in a CPG internal temporary storage queue and adopted in the defined length in the following program.(compare therefore the parameter TWA xxxx in the OPTIONS statement). This save is important when multiple programs are brunched with EXPR or EXIPT (with program name) in a Task cycle.

Look up the description in EXPR.

The operation EXITP with program name contains a RNDOM*ALL operation, except if the parameter USE is declared in the options.

EXITP-VAR Call up an external program variable (not for QPG)

OP EG (SV)

OP	EXITP-VAR must be entered
EG	name of the 4- or 8 digit alphafield
SV	'SAVe' to temporary storage

Examples:

```
- EXITP-VAR
- EXITP-VAR TRID SAVE
```

Purpose:

Call up an external program (see EXITP)

EXITS Send a program (later) to another terminal (not for QPG)

(F1) OP F2 EG (SV)

F1	terminal in inverted commas or as variable
OP	EXITS or EXIT-SEND must be entered
F2	transaction in inverted commas or as variable
EG	time delay as literal or as variable
SV	'T' : EG contains clock time, 'N' does not leave program

Examles:

```
-        EXITS       TRID    #
- 'NONE' EXID-SEND   'TRO1'  '0005'  N
- 'DV14' EXIDS       TRID    ZEIT    T
- TERM   EXID-SEND   'TRO2'  #       N
```

Purpose:

Call an external program on another screen and/or in a time delay.

Description:

F2 contains the transaction of the program to be called either as 4 characters literal in inverted commas, or as variable alpha field. In the second case, the programmer is responsible that the 4 characters defined field gets a valid transaction at time of the EXITS.

F1 contains the terminal-ID of a screen where the transaction indicated in F2 shall be started, either in inverted commas or as 4-digit alpha-field. In the second case the programmer must guarantee that the field is filled with a valid Trans-ID at the time of the EXITS. To start a Non Terminal Task, 'NONE' must be entered in F1.

In EG the time delay will be indicated either as 4-digit literal in inverted commas, or as 7-digit numerical field with the contents 'OHHMMSS'. A literal is defined as 'HHMM'.

Note: if no time delay is wished, a # must be entered.

EXIT-TRANS   Call of the next transaction

EXITT   Call of the next transaction

OP F2

OP	EXITT or EXIT-TRANS must be entered
F2	transaction in inverted commas

Example:

```
     EXITT  'TST1'
     EXITT  TRID
     EXITT  ' ';        * Return to CICS (only in QPG-modules)
```

Purpose:

Call next transaction

Description:

The operation EXITT starts the next CICS transaction with the indicated Transid. Modified screen fields can be transferred afterwards with the operation MAP.

If ' ' is indicated as Transid, then will be branched to CICS from a QPG module.

EXITT-VAR   Variable EXITT (not for QPG)

OP EG

OP	EXITT-VAR must be entered
EG	transaction in the 4-place alpha field

Example:

```
     - MOVE 'TST1' TRID
     - EXITT-VAR   TRID
```

Purpose:

Call an external program (pseudo conversational)

EXIT-SEND Transfer of the program (later) to another terminal (not for QPG)

identical to EXITS (see above)

EXPR Execution of an external program (not for QPG)

(F1) OP F2 (SV)

F1	numerical field for length of the common area
OP	EXPR or EXECUTE must be entered
F2	Phase as literal (without inverted commas)
SV	'IND', 'SAVe'

Examples:

```
- EXPR  TST001  SAVE
- EXPR  PROG44
```

Purpose:

Call an external program

Description:

With this instruction an external CPG- (OR CICS-) program may be executed at this place of the program. After the execution the processing will be continued with the next instruction.

The condition is that the TWA's of the called and to be called program correspond.F2 contains the name of the program which has to be executed.

Attention that no loops arises which might cause performance-losts. This operation corresponds to the 'EXEC CICS LINK'.

The service function 'SAVe' saves the TWA of the calling program on temporary storage in a CPG internal temporary storage queue („TERM$CPG). This should not be used by the programmer.

Example:

indicators at service 'SAVE'

```
- EXPR  PROG2  SAVE;          * instruction in the program PROG1
```

In PROG1 the switches 01 and 02 are set.They are also set in PROG2 after the EXPR. Switch 99 will be set in PROG2. After returning into PROG1, there is the same condition as before the EXPR: 01 and 02 are set, 99 is not set.If the parameter TWA and the number of the characters to be adopted will be indicated in the options division of PROG2, so all switches set in PROG2 will be transferred to PROG1 while the backjump; that means that 01, 02 and 99 would be set in the example.

In the OPTIONS specification will be determined via the parameter TWA how much characters are adopted from the Transaction Work Area.

If the control returns into the calling program, the same number is transferred back into the TWA of the program to be called, that has been adopted before.

The indicators will also be adopted out of the called program. The rest of the TWA keeps its old contents, just like before the call of the subroutine.

The service function 'INDicator' corresponds to the 'SAVe' with the difference that the indicators are not token out of the subroutine, but from the temporary storage.

EF is set when the called program is not defined in the PPT. (not at Macro Level!).

Optionally a numerical field without decimal characters may be indicated in front of the EXPR. The contents of this field determines the lenght during the execution, in which the Common Area is processed. (max. 4080 bytes)

AT the program combinations (EXITP, EXPR) in ESA-surroundings bigger than 16 MB, all combined programs must lie on the same page on the 16 MB-line!

EXPR-VAR Execution of an external program (not for QPG)

OP EG (SV)

OP	EXPR-VAR must be entered
EG	Name in the 8-digit alpha field
SV	'IND', 'SAVe'

Examples:

```
    - MOVE-LEFT  'PHASE'  TO PROGN
    - EXPR-VAR   PROGN
    - EXPR-VAR   PROGN    SAVE
```

Purpose:

Call up an external program (like EXPR)

EXSR Execution of a subroutine (not for QPG)

OP F2 (BD)

OP	EXSR or PERFORM must be entered
F2	Name of a subroutine
BD	Up to 3 switches may be set

Examples:

```
    - EXSR  UP1
    - PERFORM  DATUMSPRUEFUNG
```

Purpose:

A subroutine shall be executed

Description:

With the operation EXSR the program branches to the subroutine indicated in F2, which has to be programmed at the end of the Procedure Division.

Up to 3 switches may be indicated; then they will be set before branching to the subroutine and set off after the return from the subroutine.

FILL Fill a field with a specified character

OP F2 (DY) EG

OP	FILL must be entered
F2	caracter in inverted commas or as hex-literal
DY	dummy word TO or INTO
EG	name of an alphanumeric field

Examples:

```
     FILL  ' '    PAGE
     FILL  X'00'  TO INFO
```

Purpose:

An alphanumeric field or an array is to be filled with a character.

Description:

With the operation FILL an alphanumeric field can be filled with any representable and not representable character.

F2 contains the filler either as one-place literal in inverted commas or as hexadecimal literal in the form X'00'.

Example: `FILL '*' FIELD`

Field beforehand	`'123    '`
Field afterwards	`'*******'`

FIND Searching in a table

(F1) OP F2

F1	search argument
OP	FIND must be entered
F2	(four characters) name of the table

Examples:

```
     K  FIND  TAB1
     PLZ2  FIND TAB2
```

Purpose:

A (externally created) table is searched for a field.

Description:

Condition is that a table was created, which can be processed with FIND. For creating such a table see 'QTS' in the manual CPG3 service programs.

F1 contains the name of a field, which indicates the column of the table and for its contents will be searched in this column. If F1 is not indicated, then all entries of the table will be sequentially processed.

In F2 the four-characters name of the table is indicated.

If the search argument was found or not, then the status of the operation will be put into the field CPGFRC.

' ' if the argument was found.

'EF' if the end of the table was reached, i.e. the argument was not found.

GETCHANNEL   Get higher Storage

GETHS   Get higher Storage

GETHS

Purpose:

In a dataset the contents of the fields shall be restored in such a way, as they were transferred at the time of the call of the superordinate program.

Description:

GETHS is used in datasets. With the programming of a logical file in a dataset the difficulty may occur, that contents of fields which were transferred by a superordinate program, are replaced by reading operations, for example with CHAIN before an update.

With GETHS it is possible to implement in a QPG dataset the update function also, if only a part of the fields were transferred by the calling program.

The operation takes in each case the contents of all fields of the superordinate program, which are defined in the program, and resets their values to the contents, that were available at the start of the dataset.

GET-UPDATE   Direct data reading

identical to CHAIN (see above)

GO (TO) Transfer processing (not for QPG)

(ON) OP F2 (SV)

ON	condition query (up to 3 switches)
OP	GOTO or GO must be entered
F2	label
SV	E enables to brunch into subroutines

Examples:

```
- GOTO ENDE;
```

Purpose:

The process shall be continued at another place.

Description:

This operation branches to the character determinated in F2.This character must be defined with a 'TAG' operation.

```
- ON PF1 GOTO NEXT
     ...
- NEXT;           * TAG may be leaved out !!
```

IF Conditional processing

OP F1 (DY) OK F2 (DY) (SV)

OP	'IF' must be entered
F1	first matching field
OC	Operator, see below : Table of the operators
F2	second matching field
DY	Dummy words (for example IS, THEN)
SV	Boolean operators AND or OR

Examples:

```
IF A = B;                          * if A = B
IF A GT B  AND
IF SUMME > FG(IND) OR
SUMME < 10000
IF A IS <= B;
```

Purpose:

One or two program sections are to be executed (or not) as the result of a comparison.

Description:

The IF operation enables the execution of a group of statements on the condition, that a consistent relation between F1 and F2 exists. The group is concluded by a END(IF) - statement.

```
     IF TYPE = 'KREIS'
        PI = 3,14;
        X = D * PI;
     END
```

If required, also an alternative branch can be passed through, if the relation does not exist. This is caused by the operation 'ELSE', which optionally can be inserted between IF and END statement and which seperates the IF-branch from the ELSE-branch.

```
     IF TYPE = 'KREIS'
        PI = 3,14;
        X = D * PI;
     ELSE
        X = D * D
     END;
```

Table of the operators :

Operator			Meaning
GT	**>**		GREATER THAN F1 greater F2
LT	**<**		LESS THAN F1 less than F2
EQ	**=**		EQUAL F1 equal F2
NE	**><**	**<>**	NOT EQUAL F1 not equal F2
GE	**>=**	**=>**	GREATER EQUAL F1 greater or equal F2
LE	**<=**	**=<**	LESS EQUAL F1 less or equal F2

In connection with the boolean operators AND and OR the following syntax rules apply:

To a logical combination of IFs always belongs only one END(IF). OR and AND are always behind the IF statement.

Examples:

```
     IF A = B  AND;                        *  and-connection
     IF B > C
       EDIT INFO1
     END
     IF KDNR > '      '  OR                *  or-connection
     IF SUMME > 0  OR
     IF X = 0
       EDIT INFO2
     ENDIF
     IF A > B  AND                         *  and-/or-connection
     IF A > C  OR
     IF A = B  AND
     A = C  OR
     A = D  AND
     C = 0
       EDIT INFO3
     ENDIF
```

AND and OR can be used mixed. Thereby the rules of the mathematical propositional calculus apply. Shortened: AND binds more than OR. So the example above is clear. EDIT is executed, if one of the two AND connections is true. IF does not need to be indicated in continuation lines.

DO and IF can be interlocked up to **20 levels**.

IF-DAT Comparing date fields

OP IF-DAT or IF-DATE

F1	name of a date field
OK	matching operator (see IF)
F2	name of a date field
BO	logical linking with AND and OR

Examples:

```
IF-DAT  LOEDAT > UDATE
IF-DAT  ALPHA8 < NUM20

C            LOEDAT   IFGT UDATE             D
C            ALPHA8   IFLT NUM20             D
```

Purpose:

Comparison of date fields which are filled in the form ddmmjj (day, month, year). By using IF-DAT the converting of the fields before the comparison is not necessary.

Description:

IF-DAT compares the fields CPGWD1 and CPGWD2, which contain internal the date 8-characters alphanumeric in the ISO-format.

IF-DAT may compare alphanumeric and numeric fields with each other, even if lengths and/or types are different. The matching fields will be converted internally into the 8-characters CPGWD-fields. If the field lenghts of the matching field does not allow any values for month and/or days, these will be internally replaced by '01'.

Example:

A 2-digit field that contains the value 97, becomes for the IF-DAT CPGWDx with the value 19970101.

Note:

IF-DAT does not examine the matching fields. The programmer is responsible that the fields are available in the right format.

Supported formats for IF-DAT

Length	alpha	numerical with 0 decimals	internal value
2	97	97	19970101
3		97	19970101
4	1297	1297	19971201
5	Dez 97	1297	19971201
	Dez 97	11297	19971201
		21297	20971201
6	311297	311297	19971231
7		311297	19971231
		1311297	19971231
		120502	20020512
		1120502	19020512
		2120502	20020512
8	31121997	31121997	19971231
	31.12.97		19971231
	31.12.97		19971231
9		31121997	19971231
10	31.12.97		19971231
	31.12.97		19971231

Basically the internal routine operates according to the rule that transferred data are not changed. If a matching field is e.g.defined with 4-characters alpha with the value '0097', then a value'01001997' is internally processed.The (missing) day is replaced by '01', the (wrong) month '00' will not be modified.

With odd number numeric fields the first place stands for the millenium: 0 or 1 for 1900, 2 for 2000.

The IF-DAT-operations operates with the sliding window technique with a default value of 30. That means, years greater than 30 are assigned to the century 19xx, years smaller and equal 30 to the century 20xx. Another window can be given in the customer configuration (CPGURSIT, see CPG installation).

IF-DATI Compare date fields in the ISO-format

OP F1 OK F2 (BO)

OP	IF-DATI
F1	first comparison date
OK	comparison operator (see IF)
F2	second comparison date
BO	logic connection with AND and OR

Examples:

```
IF-DATI LOEDAT > CPGDAI
IF-DATI ALPHA8 < NUM20
IF-DATI LOEDAT > CPGDAI AND
IF-DATI DATUM > 991231 OR
VDAT < '000101' AND
IF BETRAG > 0
```

Purpose:

Comparison of date fields, which are present in the ISO-format, independent of the length. That means in particular, that comparisons of dates with up to six-characters date fields functionate for the change of the millenium.

Description:

IF-DATI corresponds to the operation IF-DAT, however with the difference, that here date values are compared in the ISO-format YYMMDD.

Supported formats for IF-DATI (by the example 31.12.1997):

Length	Alpha	Numeric with 0 Decimals	internally
2	97	97	19970101
3		97	19970101
		197	19970101
4	9712	9712	19971201
5	97.12	9712	19971201
	97/12	19712	19971201
6	971231	971231	19971231
7		971231	19971231
		1971231	19971231
8	19971231	19971231	19971231
	97.12.31		19971231
	97/12/31		19971231
9		19971231	19971231
10	1997.12.31		19971231
	31.12.97		19971231

IF-DATK Compare date fields with calendar day in ISO-format (not for QPG)

OP F1 OK F2 (BO)

OP	IF-DATK
F1	name of a 5- or 7-digit date field
OK	matching operator
F2	name of a 5- or 7-digit date field
BO	logical linking with AND and OR

Purpose:

Comparison of calendar days, which are placed in the form jjttt or jjjjttt in a 5- or 7-digit alphanumerical or numerical field.

Note:

This operation is at first a moving help for the change of the millenium. After the year 2000 the operation IF will be sufficent (like now) for the query.

Internally the datas jjjjttto will be available and compared.

The operation is not placed at the disposal of the CPG.

JLB Justify left, and fill up with blanks

OP F2

OP	JLB or LEFT-SHIFT must be entered
F2	name of an alphanumeric field

Example :

```
JLB  FIELD
LEFT-SHIFT  FIELD
```

Purpose:

Left shifting of a contents of a field.

Description:

With this operation the contents of an alpha field will be shifted in such a way, that the first character not equal to blank is put in the leftmost place of the field after the execution. The field is filled up with blanks to the right.

Example: **`JLB  FIELD`**

Field beforehand	1.	' 123'	2.	'1 2 3'
Field afterwards		'123 '		'1 2 3'

Note:
Example 2 describes the exact mode of operation of JLB: The blanks are not only sorted to the right; Character strings remain unchanged, even if they include blanks. In order to obtain the same result as in example 1, a DELC ' ' FIELD must be programmed before the JRB.

JRB Justify right and fill up with blanks

OP F2

OP	JRB or RIGHT must be entered
F2	Name of an alphanumeric field

Example :

```
JRB  FIELD
RIGHT  FIELD
```

Purpose:

Right shifting of a field.

Description:

With this operation the contents of an alpha field can be shifted in such a way, that the last character not equal to blank is put in the rightmost place of the field after the execution. The field is filled up with blanks from the left.

Example: `JRB FIELD`

Field beforehand	1.	'123 '	2.	'12 3 '
Field afterwards		' 123'		' 12 3'

Application:
Margin alignment for text editions, unpacked numeric data, etc..

Note:
Example 2 describes the exact mode of operation of JRB: The blanks are not only sorted to the left; Character strings remain unchanged, even if they include blanks.In order to obtain in this case the same result as in example 1, a DELC ' ' FIELD must be programmed before the JRB.

JRC Justify right and fill with specified character

OP F2 EG

OP	JRC or RIGHT-CHAR must be entered
F2	character, with which shall be filled up, in inverted commas
EG	Name of an alphanumeric field

Example :

```
     JRC  '*'  FIELD
     RIGHT-CHAR  '-'  FIELD2
```

Purpose:

Right shifting of a field.

Description:

The operation JRC operates like JRB. The blanks sorted to the left are replaced by the character, which is indicated in F2.

Example: `JRC '*' FIELD`

Field beforehand	`'123   '`
Field afterwards	`'***123'`

JRZ Justify right and fill up with zeros

OP EG

OP	JRZ or RIGHT-ZERO must be entered
EG	name of an alphanumeric field

Example :

```
     JRZ   FIELD
     RIGHT-ZERO  KDNR
```

Purpose:

Right shifting of a field.

Description:

The operation JRZ operates like JRB. The blanks sorted to the left are replaced by zeros.

Example: `JRZ FIELD`

```
Field beforehand    '123   '
Field afterwards    '000123'
```

LEFT-SHIFT   Justify left, blanks to the rear

identical to JLB (see above)

LIST   Output of externally described lists

(F1) OP F2 (EG) (SV)

F1	Printer name (only online necessary)
OP	LIST must be entered
F2	name of a QTF-document
EG	name of a section in the print document
SV	P=List Phase, I=Phase if the document is missing

Examples:

```
     'L86C' LIST KUNDE
     DRID   LIST DOKUM5 HEADER
     DRID   LIST QPG$L  OPCODE
```

Purpose:

Printing a list, which is described program-externally in the QTF (Quick Text Facility).

Description:

In F1 (except during batch-processing and with direct printing in the CICS) the printer name must be indicated as a literal or as a field.

In F2 the name of the QTF document, which contains the description of the list is indicated. This document must be stored in the library LIST of the QTF. A '$'-Character in the document will be replaced by the language code (QTF) of the user. So the program can be used without complex programming for the valid languages (at present D=German, E=English).

Additionally the name of a section can be indicated, a section is a part of a document.

The current linecounter can be queried after the LIST operation, if the field CPGLCT (3.0 characters) is defined in the program. Then CPGLCT can be used e.g. for the OVERFLOW control.

<u>LIST-VAR Variable programming of the LIST command</u>

<u>OP EG (SV)</u>

OP	LIST-VAR must be entered
EG	Name of a 32-characters alpha field
SV	P=List Phase, I=Phase if the document is missing

Purpose:

The LIST command (described as above) shall be variable programmed. The necessary data are inferred from the 32-characters field at the run time with the following structure (all partial fields alpha).

Columns 1	- 8	Document
Columns 9	- 14	Section
Columns 15	- 18	Printer-Id
Columns 19	- 22	Library
Columns 23	- 23	Printer Exit (look QTF-Manual)
Columns 24	- 24	Translate (N, 1 or 2)
Columns 25	- 25	Phase-Processing (P or I)
Columns 26	- 26	Automatic new page (S)
Columns 27	- 32	not yet used

For detailed description, see QTF manual.

<u>LOADT Load the contents of a screen</u>

<u>LOADT-VAR Load the contents of a screen variable</u>

<u>OP F2</u>

OP	LOADT or LOADT-VAR must be entered
F2	Name of a Temporary Storage Queue

<u>Examples:</u>

```
     LOADT TS01
     LOADT 'HUGO'
     LOADT-VAR FELD
```

Purpose:

Loading the contents of the screen which has been saved with SAVET.

Description:

This operation writes back on the terminal the contents of the screen (see below) saved with SAVET. The name of the Temporary Storage Queue must be indicated, on which the data were stored with the operation SAVET. The name will be indicated as a four- characters literal for LOADT or for LOADT-VAR in a field, that is defined four-characters alphanumerically.

The operation LOADT can only be used together with SAVET, however it is not necessary to use the two operations in the same program.

With dialogue oriented programming mode, LOADT does not stop the program. Therefore a screen must still be read explicitly.

Note:
The reconstructing of the colours is only guaranteed, if they are set by color attribute (B, G, P, R, T, W, Y).

LOKUP Look up an array for data

OP F1 OC F2

OP	LOKUP must be entered
F1	Name of the array which has to be looked up
OC	Operator
F2	search argument

Examples:

```
     LOKUP FG(I) = FIELD
     LOKUP FG(I) <= FIELD
     LOKUP FG(I) GE FIELD
```

Purpose:

Look up an array for a certain content.

Description:

LOKUP searches arrays for a certain content. The indicated array is compared element by element with the search argument. The operation LOKUP is finished as soon as the condition is fulfilled.

For the operation LOKUP an index must be indicated, which fulfills following function:

1. The comparison begins with the element of the array marked by the index.

2. After the termination of the LOKUP operation, the index of the element which has filled the condition of the comparison is in the index field, or it is 0 (zero), if the condition was not fulfilled by any element of the array.

Example:
The 5-element array FG3 has the following contents :
`'CPG' 'HL1' 'QPG' 'QSF' 'QTF'`

the instruction: **`LOKUP FG3(I) = 'QPG'`**

has the following effects:

E.g. if I=1 before the LOKUP, then 'QPG' will be found; It will be I=3 after the LOKUP.

E.g. if I=4 before the LOKUP, then only the elements 4 and 5 will be compared with the argument 'QPG' and I=0 is set.

(for alpha fields) if the search argument and the array have different field lengths, then the comparison takes place in the length of the shorter operand.

MAP QSF Map input transfer

OP F2 (SV)

OP	MAP or RECEIVE must be entered
F2	contains the up to 8-characters name of the map
SV	services: Blank, Clear and Low

Examples:

```
     MAP KUNDEN;
     RECEIVE BRIEF LOW;
```

Purpose:

Data are to be read from a QSF map.

Description:

This instruction enables to describe all screen fields interactively outside of the program. Application programs do not define screen inputs and outputs. All literal and variable fields are described externally.

A modification of the display screen mask usually requires no compilation of the application program. All fields definied in the program may be entered in a map.

The operation MAP reads data from the screen in a pseudo conversational program.This instruction is usually coded at the beginning of a program.

With the service SV the following options are possible:

Blank deletion of alpha fields of the mask on blank and num.fields on 0.

CLEAR deletion var. attribute fields of the mask on blank.

Low processing lowercase letters. No translation is executed in uppercase letters, if the system UCTRAN is not set in the Terminal Control Table, or the program is switched off with the operation UCTRN OFF.

A'$'-sighn will be replaced by the language code(QTF) of the user, with it the program may be used without a complex programming for the valid languages (at present D=German, E = English).

With MAP, MAP-VAR, MAPD, MAPD-VAR, MAPI and MAPI-VAR the status in the internal field CPGMRC (map return code, 2 characters alpha) is set:

' '	normal input
'IC'	Invalid Charater
'NI'	NO input

MAP-VAR QSF Map -instruction variable

OP EG (SV)

OP	MAP-VAR must be entered
EG	16-characters field with variable information
SV	service: 'LOW' for lowercase letter

Purpose:

The MAP instruction (s.o.) is to be variable programmed. As operand an alpha field defined with 16-Byte is entered, which contains the map name from place 1-8. Place 12 can contain a 'B' or 'C' for the service 'blank' or 'CLEAR'(see operation MAP). All other characters are reserved for MAPO.

The status is set in the field CPGMRC (see MAP operation).

MAPD QSF Map Dialog

OP F2 (SV)

OP	MAPD must be entered
F2	contains the up to 8-digit name to the map
SV	service: LOW, CLEAR, I

Examples:

```
MAPD KUNDEN;
MAPD NUMMER LOW
```

Purpose:

A QSF map is to be output, afterwards data are to be read from this map in the dialog.

Description:

This instruction enables to describe all screen fields interactively outside of the program (see to operation MAP).

If SV contains the keyword 'LOW', then no translation is executed in uppercase letters at the QSF.

The keywords 'CLEAR' or 'I' enable the query to the key CLEAR (delete key). Otherwise this taste causes the termination of the program. Hardware-caused with the delete key the display is always deleted.

Service 'S' for the combination of CLEAR and LOW.

A $ - Character in the map name is replaced by the language code.

The status is set in the field CPGMRC (see MAP operation).

MAPD-VAR QSF Map variable dialog

OP EG (SV)

OP	MAPD-VAR must be entered
EG	16-characters field with variable information
SV	services: LOW, CLEAR, I

Purpose:

The MAPD-instruction (s.o) shall be variable programmed. The 16-Byte long EG-field has the following construction:

```
1 -  8  Mapname
9 -  9  Erase Y=Yes N=No
10 - 10 Write Control Character:
              H = BEEP
              K =        MDT + LOCK
              L =              LOCK
              M =        MDT
              N = BEEP + MDT + LOCK
              O = BEEP       + LOCK
              S = BEEP + MDT
11 - 11 F = fields only
12 - 16 reserves
```

The status will be put into the field CPGMRC (see MAP operation).

MAPI QSF Map Input

OP F2 (SV)

OP	MAPI must be entered
F2	contains the up to 8-characters name of the map
SV	service: LOW, CLEAR, I and S

Examples:

MAPI ARTIKEL
MAPI BRIEF LOW.

Purpose:

Data are to be read in the dialog from a map.

Description:

This instruction enables to describe all screen fields interactively outside of the program (see operation MAP).

If SV contains the keyword 'LOW', then no translation is executed in uppercase letters at the QSF.

The keywords 'CLEAR' or the Í' enable the query of the key CLEAR (delete key). Otherwise this key causes the termination of the program. Hardware-caused the display is always deleted with the delet key.

Service 'S' is for the combination of CLEARS and LOW.

A $ - Character in the map name is replaced by the language code.

The status is set in the field CPGMRC (see MAP operation)

MAPI-VAR   QSF Map Variable Input instruction

OP EG (SV)

OP	MAPI-VAR must be entered
EG	16-characters field with variable information
SV	services: LOW, CLEar, I and S

Purpose:

The MAPI instruction (s.o) is to be variable programmed. For the construction of the 16-characters alpha field see MAPD-VAR.

The status is set in the field CPGMRC (see MAP operation).

MAPO   QSF Map Output

OP F2

OP	MAPO or SEND must be entered
F2	contains the up to 8-characters name of the map

Examples:

```
     MAPO MENUE;
     SEND FEHLER;
```

Purpose:

A QSF map is to be output.

Description:

The operation MAPO outputs a separately created QSF map. This instruction can be given several times in the program.

In F2 the name is entered into the map, which is to be output on the display.

A $ - character in the map name is replaced by the language code.

MAPO-VAR QSF Map Output variable

OP EG

OP	MAPO-VAR must be entered
EG	16-characters field with variable information

Purpose:

The MAPO instruction (s.o) shall be variable programmed. For the setting up of the 16-characters alpha field, see MAPD-VAR.

MAPP QSF Map Output on a printer

F1 OP F2 (SV)

F1	name of the online printer (as field or as literal)
OP	MAPP must be entered
F2	contains the up to 8-characters name of the map
SV	AFTer, BEFore, S for feed on the page start

Examples:

```
     DRID  MAPP ARTIKEL
     'DR15' MAPP POSTEN BEFORE
```

Purpose:

A QSF map shall be output on an online printer.

Description:

The operation MAPP prints the map entered in F2 on the online printer defined in F1. F1 can be a variable 4-characters alpha field or a literal.

Note:
Only lines are printed, in which at least one character is. If 24 lines are printed, then the map must describe at least one blank (represented by a '#') in each line.

The following services are supported:

AFT - after the MAPP a feed on the page start is executed. BEF - before the MAPP a feed on the page start is executed. S - before and after the MAPP a feed on the page start is executed.

MAPP-VAR QSF Map Variable MAPP instruction

F1 OP EG (SV)

F1	online printer as field or as literal
OP	MAPP-VAR must be entered
EG	16-characters field with variable informations
SV	AFTer, BEFore, S for feed on the page start

Purpose:

The MAPP instruction (s.o) shall be variable programmed. For the construction of the 16-characters alpha field, see MAPD-VAR.

MOVE   Transfer right-adjusted

OP F2 (DY) EG

OP	MOVE or MOVE-RIGHT must be entered
F2	field name of the origin field
EG	field name of the array
DY	dummy word: TO

Examples:

```
MOVE A B;
MOVE 3.14 TO PI;
MOVE UMS(M) TO UMSATZ
MOVE FG(1) TO FG(I)
MOVE '*' TO STERN;
MOVE X 'FFFFFF' TO ENDTAB; * hexadecimal constant
```

Purpose:

The contents of a field shall be transferred into another field.

Description:

F2 is transferred from the left to the right after EG. The fields may be defined alphanumerically, numerically or differently.

MOVE (R)   Transfer right-adjusted

(ON) OP F2 (DY) EG (SV)

ON	condition query (up to 3 switches)
OP	MOVE, MOVER or MOVE-RIGHT must be entered
F2	field name of the origin field
EG	field name of the result field
SV	service 'C'; 'F', 'I', 'INDex'
DY	dummy word 'TO'

Examples:

```
- MOVE A B;
- ON EF MOVER A TO B;
- ON PF1 MOVE-RIGHT 'X' TO TEST;
- MOVE 3, 14 TO PI;
- MOVE UMS(M) TO UMSATZ;
- MOVE FG(1) TO FG(I);
- MOVE '*' TO STERN;
- MOVE X 'EFEFEF' TO ENDTAB;  * hexadecimal literal, up to 3 bytes
```

Purpose:

The contents of a field shall be transferred into another.

Description:

F2 will be transferred to EG right-adjusted. The fields may be defined alphanumerically, numerically or differently.

This operation is completely indicatable.

At a MOVE-operation numerically to alpha, the right zone of the alpha-field may be modified.This happens with an entry in SV. A 'C' or 'F' means that the sign 'C' or 'F' is valid divergent for the standard name.

A 'I' or 'INDex' causes that no test of the used index will be realized. The programmer is responsible that no storage area will be transferred by mistake.

MOVEA Move array

OP F2 (DY) EG

OP	MOVEA or MOVE-ARRAY must be entered
F2	transferring area
DY	dummy word TO
EG	name of the array

Examples:

```
MOVEA F256 TO OF FG16
MOVEA FG(IND) LINE24
MOVE-ARRAY FG1 TO FG2
```

Purpose:

Transfer of alphanumeric characters, whereby either the area which is to be transferred or the result field an array is, or both arrays are.

Description:

The operation MOVEA transmits alphanumeric characters beginning with the left place from F2 to EG. Contrary to the operations MOVE and MOVEL field groups are not shifted item wise, but as a coherent area. The length of the MOVEA operation is determined by the length of the shorter field (F2 or EG).

Examples:

Fields beforehand:
```
FG25:        ' ABCDE '   ' FGHIJ ‘
FG34:        ' **** '    ' **** '    ' **** '
```

```
     MOVEA  FG25 TO OF FG34;
```

FG34 after MOVEA : `' ABCD '    ' EFGH '    ' IJ** '`

```
     MOVE-ARRAY FG25(2) TO OF FG34(2);
```

FG34 after MOVEA : `' **** '    ' FGHI '    ' J*** '`

MOVEL Move data left-adjusted

OP F2 (DY) EG

OP	MOVEL or MOVE-LEFT must be entered
F2	field name of the origin field
EG	field name of the result field
DY	dummy word: 'TO'

Examples:

```
     MOVEL A  TO  B
     MOVE-LEFT A  TO  B
     MOVEL 'X'  TO TEST;
     MOVEL UMS(M)  TO UMSATZ;
     MOVEL FG(1)  TO FG(I);
     MOVE-LEFT STERN  TO FG;
     MOVEL X '0D0C0D'  TO ESCAPE.  * hexadecimal literal, up to 3 bytes
```

Purpose:

The contents of a field shall be transferred into another field.

Description:

F2 is transferred from left-adjusted to EG. The fields may be defined alphanumerically, numerically or differently.

MOVEN transfer alphanumeric into numeric field

OP F2 (DY) EG (SV)

OP	MOVEN must be entered
F2	field-name of the alphanumeric origin-field
EG	field-name of the numerical result-field
DY	Dummyword: 'TO'
SV	service H for extended field-test

<u>Examples:</u>

```
     MOVEN A N
     MOVEN A TO B
     MOVEN UMS(M) TO UMSATZ H
     MOVEN FGA(1) TO FGN(I)
     MOVEN A TO N H
```

Purpose:
Contents of an alphameric field are to be transferred into a field, using the same algorithm as the screen input for numeric fields.
Description:

The field in factor 2 is transferred into the result field, in the same way as if the field was read from the screen into a numeric field. That means that a decimal position adjustment will be made and the positions with the highest value are cut off in front if necessary. The comma (or point in the 'english' installation)is detected as separator of the decimal positions.
Minus signs and the character sequence 'CR' in the alpha field cause that the field is interpreted as negative value. Invalid characters are eliminated from the field and the field is compressed accordingly before the MOVEN.

Three indicators can be indicated:
First is set, if the alpha field contains invalid indications.
Second is set , if the alpha field contains too many places before or after a comma.
Third is set, if the alpha field is blank.

For QPG only:

In the internal field CPGPRC (Program Return code, 2 positions alpha) the status for the transmission is set:

' ' normal transmission.
'BL' blank: the alpha field is empty.
'IC' Invalid Charater: the alpha field has invalid indications.
'OF' OVERFLOW: the alpha field contains too many paragraphs.

With the service function 'H' an extended examination of invalid characters is achieved.In this case, invalid characters are detected if blanks or minus signs appear within a sequence of numbers.

<u>Examples:</u> `MOVEN ALPHA TO NUM`

The numeric field is seven-position defined with 2 decimal positions.

QPG only!

ALPHA	NUM	intern	Status CPGPRC	(with H)
'123 '	123	0012300C	' '	' '
' 123 '	123	0012300C	' '	' '
' 1 2 3 '	123	0012300C	' '	'IC'
'-123 '	123,00-	0012300D	' '	' '
'12-3 '	123,00-	0012300D	' '	'IC'
' 987,65CR '	987,65-	0098765D	' '	' '
',1 '	0,1	0000010C	' '	' '
'12.345,678'	12345,67	1234567C	'OF'	'OF'
'56,7D '	56,7	0005670C	'IC'	'IC'
'ELF DM '	0	0000000F	'IC'	'IC'
' '	0	0000000F	'BL'	'BL'
'12A4567 '	24567	2456700C	'OF'	'OF'

MOVEV variable MOVE-Operation

OP F2 (DY) EG (SV)

OP	MOVEV must be entered
F2	ten-digit alphameric field
EG	ten-digit alphameric field
SV	ARRay, LEFt, Numeric, RIGht
DY	Dummyword:'TO'

Examples:

```
MOVEV A B
MOVEV A  TO  B  LEFT
MOVEV F1 TO F2  ARRAY
```

Purpose:

Contents of the field, whose name is in F2, are to be transferred into the field, whose name is in EG.

Description:

In case of same field length and same field type, alphanumeric fields are transferred left-adjusted and numeric fields are transferred right-adjusted.

If factor 2 is smaller than the result field, then the transfer from alpha to numeric will be right-adjusted and from numeric to alpha left-adjusted.

If factor 2 is larger than the result field, then the transfer from alpha to numeric will be left-adjusted and from numeric to alpha right-adjusted.

The service functions have the following meaning:

Arr	for array. MOVEV operates in this case such as **MOVEA**.
Left	for Left. MOVEV operates in this case such as **MOVEL**.
Num	for numeric transfer.
Right	for right. MOVEV operates in this case such as **MOVE**

MOVEV is not indicatable in the entries, but can be processed indicatedly as follows:

Example:

```
MOVE X TO A
MOVEL 'FG    ' TO A
              MOVEL 'RESULT' TO B
              MOVEV A TO B RIGHT
```

After these operations, array element number X of the array FG is transferred into the field RESULT.The transfer is right-adjusted.

So that this indicated processing form is possible, the two entries must be always 10 places large alpha fields. In the first six places of these fields the field name is always located. In the case of indicated processing thus the name of the array is in the places 1 to 6 but the value of the index field in the places 7 to 10.

Service function numeric

This service is not supported for arrays or array items.

1.From alpha to numerical: Transfer as with MOVEN.

Example:	**Alpha (15)**	**' Num. (9,3)**
field:	' 123.999999 '	000123999C
field:	' 123,999999- '	000123999D
field:	' 1234567.999999 '	234567999C
field:	' -1234567.999999'	234567999D

2.From numerical to alpha.
- the transfer takes place right-adjusted.
- the receiving alpha field must be large enough.
- the value is edited with the edit code J.
- the alpha field should be initialized with blanks.

Example:	**Num. (9,3)**	**alpha (15)**
field:	000123999C	' 123.999 '
field:	000123999D	' 123.999-'
field:	234567999C	' 123,456.999'
field:	234567999D	' 123,456.999-'

With the operation MOVEV if an error is detected, e.g. that the field name is invalid or the index is incorrect, then no transfer takes place.

MOVE-ARRAY   array transfers

identical to MOVEA (s.o.)

MOVE-LEFT   left adjusted field transfer

identical to MOVEL (s.o.)

MOVE-REST   remainder of a division

identical to MVR (see below)

MOVE-RIGHT   right adjusted field transfer

identical to MOVE (s.o.)

MVR   move remainder

OP EG

OP	MVR or MOVE-REST must be entered
EG	field name of the result field

Examples:

```
     MVR REST
     MVR DIVRST
```

Purpose:

Save a division remainder.

Description:

With the operation MVR (only) immediately after a division the remainder can be transferred into the field entered in EG.

Example:

```
     QUOT = 50 / 3
     MOVE-REST TO REST
```

After this sequence of instructions the field REST has the content of 2.

OPEN Open file

OP FN (SV)

OP	OPEN must be entered
FN	name of the file which shall be opened
SV	service (input/output mode for VSAM files and (data sets)

Examples:

```
     OPEN CPGWRK;
     OPEN KUNDEN;
     OPEN FILE Input
     OPEN FILE Update
     OPEN DATASET Output
     OPEN DATASET Reuse
```

Purpose:

A file is to be opened.

Description:

The file indicated in FN is opened.

For the batch processing of VSAM files, the input output mode can be specified with the service. So can be explicitly indicated for the OPEN if the file must be opened for Input, Update, Output or Reuse(output). The mode is preserved for the entire batch processing.

With datasets the service will be transferred as extension of the operation code. E.g. at OPEN input 'OI' is transferred in the field CPGHIC to a HL1 dataset.With a QPG dataset at OPEN Reuse, 'OR' will be transferred in the field CPGFRC.

The status after the OPEN will be transferred in the internal field CPGFRC.

After the OPEN, the file return code is set as follows:

' '	file was successfully opened.
'NF'	file was not found (in the FCT).
'NO'	file could not be opened.

With datasets the return code must be set accordingly by the dataset.

PARAMETER Parameter transfer to the CALL (not for QPG)

Change in relation with the operation CALL

PARM Parameter transfer to the CALL (not for QPG)

OP F2

OP	PARM or PARAMETER must be entered
F2	field name of the parameter

Examples:

```
    - CALL 'PROGRAMM'
    - PARM DATEI
    - PARM SATZ
```

Purpose:

A parameter list shall be build up for the CALL-instruction. The parameter list corresponds to the conventions of CALL-interfaces.

The instruction is only sensefull after a CALL-instruction. As many as you like PARM-instructions may be coded, one for every parameter.

Between CALL and PARM no other statement may be coded.

PERFORM Execute a subroutine (not for QPG)

identical to EXSR (s.o.)

PROG(RAM) call QPG program

OP F2 (EG)

OP	PROG or PROGRAM must be entered
F2	program name
EG	Library

Example:

```
PROGRAM OTTO
PROG HUGO TASK
```

Purpose:

A QPG program is to be called as external subroutine.

Description:

The QPG program indicated in F2 is executed as external subroutine. Fields are transferred automatically, if the field names and -lengthes in the calling and called program correspond (see chapter 3100, data exchange).

In EG the library is indicated, if the subroutine is in another library than the calling program.

Note: In CPG2 programs the instruction PROGRAM is to be used.

PROG-VAR call QPG program variable

OP EG

OP	PROG-VAR must be entered
EG	32-Byte long alpha field

Example:

```
PROG-VAR HUGO
```

Purpose:

A QPG program is to be called as external subroutine.

Description:

The 32-Byte long parameter field EG has the following structure:
Agency:

1	- 8	program name
9	- 12	library (necessary only if the called program is in other library)
13	- 32	reserved

PROT(ECTION) protection code has to be given out (see manual CPG3)

OP F2

OP	PROT or PROTECTION must be entered
F2	field that contains the protection code

Example:

PROT A
PROTECTION FELD

Purpose:

A program is to be protected in connection with CPG3..Sign On against unauthorized access.

Description:

PROT is executed according to CPG3 Sign -On. A 12-digit alpha field is needed, which is structured as follows:

Place

1-	8	Symbolic name of the protection code
9-	9	Type of the error handling ' ' - by CPG3-Serviceprograms 'R' - own programming after call of the Return Code
10-	10	Return code '0' - access is allowed '1' - None Terminal-Task, PROT not authorized '2' - user is not signed on '4' – program not in the Protection- Table '7' - errors within the area 'Clients' '8' - errors within the area of 'Access Areas'
11-	12	Reserved

PURGE Delete temporary storage queue

OP F2

OP	PURGE must be entered
F2	name of the Temporary Storage Queue which shall be deleted

Example:

```
PURGE STOR
```

Purpose:

A Temporary Storage queue is to be deleted.

RANDOM Reset file to random processing

identical to RNDOM (see below)

READ read a file sequentially

(F1) OP FN (SV)

OP	READ must be entered
F1	field name of the key
FN	file name
SV	S for save at TS

Examples:

```
READ CPGWRK
KDNR READ KUNDEN;
READ STORAGE S
1 READ TPTC
READ HQTFC
```

Purpose:

A record of a file is to be read sequentially.

Description:

<u>VSAM (KSDS/RRDS/ESDS)</u>

Records of a VSAM file are sequentially read.

In F1 the field name of the key can be indicated.During the first execution it is possible that the key field contains the key of the record, with which the sequential processing begins.

FN contains the name of the file. The key can be generically indicated. If the entered key is missing in the file, the record with the next higher key is read.

With end of file the status 'EF' is set in the field CPGFRC. With VSAM files 'EF' must be queried after the READ operation because after another READ after EF the program would abord with a system error message.

<u>STORAGE</u>

F1 can remain free or contain a numeric code. Normally the area is released after reading. A 'S' or 'SAVe' in SV causes at simulated TS queues that the area is preserved after reading.

<u>HL1 and QPG datasets</u>

F1 must remain free.

READ-BACK Read file backwards

READB Read file backwards

<u>(F1) OP FN</u>

F1	field name of the key can be entered
OP	READB or READ-BACK must be entered
FN	file name

Examples:

```
KEY READB DATEI;
READ-BACK KUNDEN
READB HQTFC
```

Purpose:

A record or several records(e.g. a VSAM file)is to be readed. The file is processed sequentially backwards.

Description:

The operation READB for VSAM files operates like READ, however the records are read backwards. That means that the logically next record is the one with the next lower key.

At the start of file the status 'EF' is set in the internal field CPGFRC.

Differently to the READ the first record read with READB must be available in the file. If the record which should be read is missing, then 'EF'is set.

READB-PAGE Read file backwards

(F1) OP F2

F1	field name of the key may be declared
OP	READB or READ-BACK must be entered
F2	file name

Examples:

```
- KEY READB DATEI;
- READ-BACK KUNDEN
```

Purpose:

A record or several records of a VSAM-file shall be readed. The file will be processed sequentially backwards.

Description:

The operation READB for VSAM-files works like READ, however the records are to read backwards. That means that the logical next record is that with the next smaller key. At the file-start the switch EF will be set and the internal field CPGFRC will be filled with 'EF'.

Differently to the READ the first record read with READB must exist in the file. If the record to be read does not exist, 'EF' will be set and no input is processed.

READI Read Segment of an input file

OP FN EG

OP	READI must be entered
FN	name of the file, from which a segment is read

EG	name of the segment, that is described in the input

Example:

```
     READI AUFTRAG POSITION
     READI AUFTRAG TEXTE
```

Purpose:

From a file already read a certain structure is to be transferred again into the program.

Description:

Transfer of segments. In particular with VSAM files with different record types it is of advantage possibly to read first a part of the data record. According to contents of the read data one decides then, into which structure the input record will be transferred.

One achieves this additional transfer of input data already read with the instruction READI. According to the input regulations for a segment the input data of the lastly read record will be transferred again.

Note: READI is always used after READ- or READ-BACK operation or after a CHAIN instruction (whereby in the CICS in addition the service 'U' or 'P' is necessary). If this regulation is not followed, then the program abords during the CICS-execution.

If READI is used under the CICS after CHAIN-U or CHAIN-P, without an update being made, then the file has to be released after the(last) READI operation with RNDOM.

READP Read records of a disc-file into a page

(F1) OP F2 EG

F1	name of the key-field may be declared
OP	READP or READ-PAGE must be entered
F2	name of a disk-file
EG	name of an array, into which is read.

Example:

```
     - KEY READP POSTEN PAGE
     - READ-PAGE ARTSTA S01
```

Purpose:

Records of a file shall be read into an array.

Description:

The operation READP works like READ. The key field in F1 declares, at which position of the file the processing begins. If the key is not available, the next higher one will be read.

From the file indicated in F2 may be read as much records as the array indicated in EG has elements.

Prerequisite is that in the output-division is described how array-elements shall be edited.

Before reading the array is deleted.

RECEIVE transaction-oriented Read of a QSF map

identical to MAP (s.o.)

REPLACE replace a character by another

REPLC Replace a character by another

(F1) OP F2 EG

F1	Character which is to be replaced
OP	REPLC or REPLACE must be entered
F2	character which is set as literal or as variable
EG	alpha field / - array / -array element

Examples:

```
     REPLACE '0' WERT;   * replace blanks by zeros
     REPLC X'00' F10;    * replace blanks by X'00'
     REPLC F2 EG;        * replace blank by content of F2
     'x' REPLC '*' EG;   * replace 'x' by '*' in EG
```

Purpose:

Any character in a field shall be exchanged with another. In the case of default blank is replaced.

Description:

F1 contains the character, which is to be replaced, as alphanumeric or hexadecimal literal or as variable in a one-place alpha field. If F1 is not indicated, then blank (X'40') is replaced.
F2 contains the character, which is to be inserted instead, as alpha numeric or hexadecimal literal or as variable in a oneplace alpha field.

In EG is the name of the field which shall be changed.

RIGHT Shift alpha field right-adjusted, blanks in front

identical to JRB (s.o.)

RIGHT-CHAR Shift alpha field right-adjusted, character in front

identical to JRC (s.o.)

RIGHT-ZERO Shift alpha field right-adjusted, zeros in front

identical to JRZ (s.o.)

RNDOM Reset file to random processing

OP FN

OP	RNDOM or RANDOM must be entered
FN	file name

Examples:

```
     RNDOM KUNDEN
     RNDOM ARTIKEL
```

Purpose:

A file, which was sequentially processed, may be 'switched' to direct access.

A record closed with CHAIN UPDate is to be unlocked again, if no update took place.

FIND tables are reset to the first element of the table. The next search with FIND starts at the first table element.

ROLL Roll array contents up one element

OP EG

OP	ROLL must be entered
EG	name of a array (possibly with index)

Example:

```
     ROLL PAGE
     ROLL PAGE(I)
```

Purpose:

Shift forward of array elements

Description:

With ROLL within an array all elements are shifted on the next Lower index. The first element of the array is lost thus; the last element remains unchanged. If an index is indicated for the array, the shift starts at this element.

Example: Contents of the array FG

Before ROLL		after ROLL FG
FG, 1	'AAA'	'BBB'
FG, 2	'BBB'	'CCC'
FG, 3	'CCC'	'CCC'

ROLLB Shift an array backwards

ROLL-BACK Shift an array backwards

OP EG

OP	ROLLB or ROLL-BACK must be entered
EG	name of a array (possibly with index)

Examples:

```
ROLLB PAGE
ROLL-BACK FG
ROLL-BACK FG(I)
```

Purpose:

Shift from items of a array rearwards

Description:

With ROLLB within an array all elements are shifted on the next higher index. The last element of the array is lost thus; the first element remains unchanged.If an index is indicated for the array, then the shift starts at this element.

Example: Contents of the array FG

Beforehand		after ROLLB FG
FG,1	'AAA'	'AAA'
FG,2	'BBB'	'AAA'
FG,3	'CCC'	'BBB'

SAVET Save screen contents

SAVET-VAR Save screen contents variable

OP F2

OP	SAVET or SAVET-VAR must be entered
F2	four places name of a temporary storage queue

Example:

```
SAVET  TS03
SAVET-VAR FELD
```

Purpose:

The actual screen is saved.

Description:

The operation SAVET saves the actual screen contents onto Temporary Storage.

F2 contains in max. 4 byte the name of a TS queue, in which the displayed screen page is saved (this name is extended in front internally by the four places terminal id). With SAVET-VAR the Storage name must be made available in a four place alpha field.

With the operation LOADT the saved picture may be loaded again. This operation is used e.g., in order to call a help window by a program and to restore the old screen contents afterwards.

Note: Reconstructing the colors is only guaranteed if these are set by color attribute (B,G,P,R,T,W,Y).

SCAN searching for a character sequence in an alpha field

F1 OP F2 EG (SV)

F1	search argument (character sequence)
OP	SCAN must be entered
F2	name of the field to be looked up
EG	numerical Field for initial value and found position
SV	service V for looking up in variable length

Examples:

```
FELD  SCAN  FG(I)  POS
FG    SCAN  SATZ   INDEX
ARG   SCAN  FELD   INDEX  Var
'*'   SCAN  SATZ   X
```

Purpose:

Searching a character sequence in an alphanumeric field.

Description:

The operation SCAN looks up the field in F2 for contents of the field in F1. If the character sequence is found, then the position is transferred into the result field.

As result field a numeric field can be indicated with 0 decimal positions. If this field is greater 0 before the instruction, then the search begins at the appropriate position of the field. After the operation the field contains the position, at which the character sequence, i.e. the first character of the sequence was found. If the character sequence is not found, then the result field is set to 0.

The field length of factor 2 must be larger than the field length of factor 1. At arrays in factor 2 it is to be noted that the element length must be larger than the field length in factor 1.

With the service 'V' may be searched in variable length. The length corresponds to the number of places which are filled into the search argument. The end of the argument is thereby blank or x'00'. If an argument is searched which contains e.g. blank characters, then any special character can be used to include the search argument.(the same special character at the start and at the end). Thereby the special character is not used as search argument.

SCREENDUMP debugging aid special terminaldump

SDUMP debugging aid special terminaldump

OP (F2)

OP	SDUMP, TDUMP or SCREENDUMP have to be entered
F2	dump code with an identifier of up to four places.

Examples:

```
SDUMP 1
SCREENDUMP
```

Purpose:

Produce a Special dump on the display

SELCT field selection

SELECT field selection

OP EG

OP	SELCT or SELECT must be entered
EG	name of the field into which is read

Examples:

```
SELECT FELD
SELCT CPGCOM
SELCT CPGSIN
SELCT CPGTCT
SELECT FG(X)
```

Purpose:

In the input division the field is read like a file.

SEND send a QSF-Map to a screen

identical to MAPO (s. o.)

SET-LIMIT set pointer to a record of a file

SETLL set pointer to a record of a file

F1 OP FN

F1	field name of the key, with which is positioned
OP	SETLL or SET-LIMIT must be entered
FN	file name

Examples:

```
     KEY SETLL CPGWRK
     KDNRA SET-LIMIT CPGKDN
     QNR SETLL STOR
```

Purpose:

Onto a file is to be positioned.

Description:

With this operation the sequence of a sequential READ can be interrupted and be taken up at any place in the file again with sequential processing.

The SETLL operation does not read data, but determines by the contents of the key field indicated in F1 the record which is to be read with the next READ operation. FN contains the name of the file. The key in F1 does not have to be available on the file. Thus it is possible to position with a partial key. In such a case it is positioned at the next higher key.

If 'End of File' is reached at the execution of a SETLL operation, then the internal field CPGFRC is filled with 'EF'.

SORT(A) Sort an array

OP F2 (SV)

OP	SORTA or SORT must be entered
F2	name of an array
SV	'Blanks', ' Descending'

Examples:

```
     SORT FG1
     SORTA PAGE BLANKS
     SORTA FG2 DESCEND
```

Purpose:

An array is to be sorted.

Description:

The operation SORTA sorts the array indicated in F2 ascending by.

The service function 'blank' causes that additionally to the ascending assortment, the array elements, whose contents are blank (or zero at numeric arrays) are sorted to the rear.

The service function 'D' causes that the array is sorted in descending sequence.

Examples:

```
     SORTA FG
     SORTA FG DOWN
     SORTA FG BLA
```

```
      Beforehand   after SORTA   after SORTA D     after SORTA B
FG(1) ' 2   '      ' '           ' 2 '             ' A '
FG(2) ' A '        ' A '         ' 1 '             ' B '
FG(3  ' '          ' B '         ' B '             ' 1 '
FG(4) ' B '        ' 1 '         ' A '             ' 2 '
FG(5) ' 1 '        ' 2 '         ' '               ' '
```

SQRT calculate square root

SQUARE-ROOT calculate square root

OP F2 EG (SV)

OP	SQRT or SQARE-ROOT must be entered
F2	name of a numeric field
EG	name of a numeric field (for the result)
SV	'H' or 'ROUnded' for rounding

Examples:

```
     SQRT NUMBER OF ROOT
     SQARE ROOT F30 ROUND
```

Purpose:

The square root of a number is to be calculated.

SQRT can also be used for arrays and array elements.

SYNCP (OINT) Define a synchronization point (not for QPG)

OP (SV)

OP	SYNCP or SYNCPOINT must be entered
SV	'ROL' for syncpoint Roll-Back

Example:

```
- SYNCPOINT
```

Purpose:

Define a synchronization point.

Description:

At the system recovery of the TP-control program, the operation SYNCP indicates up to which point the process was locked. All file-modifications (Update, addition, eliminate), occured after a SYNCP will be declined per Dynamic Transaction Backout after an abnormal program -/Task-end.

Prerequisite for the function of the operation SYNCP is, that a LOG-File is positioned for the TP-Monitor and that the FCT and the PCT will get corresponding entries:

Example:

FCT:	JID=System, LOG=YES	for all Update-files used in the program
PCT:	DTB=YES	for all programs containing SYNCP

With the service-function ROLlback, SYNCP-order given wrongly, may be lifted again.

Before the operation SYNCP all used files in the program must be released with RNDOM*ALL.

TAG Define a label

F1 (OP)

OP	TAG may be entered
F1	label (label in the program)

Examples:

```
- START TAG
- END-OF-INFOICE
```

Purpose:

A program-position should get a name.

Description:

This operation sets a label, to which may be branched with help of a GOTO-Operation. The name of the label stands in F1.

The name of the label must not correspond with a field-name which is already used.

Exeptionally the operation-code (TAG) is not applicable for this operation. A single name in the Procedure-Division is always interpreted as a TAG.

TEST-FIELD check field for numeric characters

TESTF check field for numeric characters

OP F2 EG (SV)

OP	TESTF or TEST-FIELD must be entered
F2	alphanumeric field to be checked
EG	one-place alpha field for the result of TESTF
SV	L, in order to check also the last byte (normal case)

Examples:

```
TESTF FELD STATUS LAST
```

Purpose:

An alphanumeric field has to be examinated for numeric contents.

Description:

The alphanumeric field in F2 is checked for numeric contents. The result of the operation is stored in EG:

'B'	the field contains only blanks (hexadecimal '40')
'M'	the field contains only digits and leading blanks
'N'	all characters are digits. ' ' will be set in all other cases.

Under normal conditions the service function L is entered, so that also the last byte is checked for digit.

(In special cases, in which numerically processed fields were stored unpacked to files, one operates without the service L. In these cases the PACK /UNPACK function of the assembler have led to the fact that in the last byte of the field the zone is located in the first half byte. If a zone is C or D, then here are letters A to I or J to R, which would be transferred correctly into a value during the pack. Also such fields can be tested with TESTF: Without service L also the letters A to R in the last place are detected and treated like the corresponding digits. See table of the EBCDIC code.)

TIME set time

OP (EG)

OP	TIME must be entered
EG	name of a numeric field

Examples:

```
     TIME
     TIME MMSS;
```

Purpose:

The internal fields UTIME and CPGTIM are updated.

Description:

Into the field entered in EG the numeric field with six digits with zero decimal places CPGTIM will be transferred right adjusted.

TWA-LOAD Read private TWA from Temporary Storage

TWALD Read private TWA from Temporary Storage

OP F2 (SV)

OP	TWALD or TWA-LOAD must be entered
F2	name of a Temporary Storage area
SV	VARiabel, if Storage Name stands in al field

Example:

```
     TWALD TS23;  * Syntax for CPG and QPG
     TWA-LOAD NAME VAR
```

Purpose:

A TWA saved with TWA SAVE on Temporary Storage is read in again.

Description:

In F2 a name for the TS area is indicated, maximum length is 4. The TWA is read only if it fits with the program, i.e.has the correct length; therefore TWA-LOAD will normally only be executed in the program in which the operation TWA-SAVE was used.

In the field CPGFRC the status is set whether the TWA was found:

' '	the private TWA was loaded.
'EF'	the private TWA was not found.

TWA-SAVE Save private TWA onto Temporary Storage

TWASV Save TWA onto Temporary Storage

OP F2 (SV)

OP	TWASV or TWA-SAVE must be entered
F2	name of a Temporary Storage area
SV	VARiabel, if Storage Name stands in a field.

Example:

```
     TWA-SAVE TS23; * Syntax for CPG and QPG
     TWASV NAME VAR
```

Purpose:

A TWA is saved onto Temporary Storage.

Description:

In F2 a name for the TS area is indicated, either as a 4 places alphanumeric string (without quotation notes) or in a 4 places alphanumeric field, if the service ‘VAR’ is used.

The private TWA stored with TWA-SAVE can be read later with TWA-LOAD again.

TWASV-VAR TWA on temporary storage rescue

identical to TWASV and the service-function VAR

UCTRAN translating into uppercase letters

UCTRN translating into uppercase letters

OP F2 (SV)

OP	UCTRN must be entered
F2	entries: ON or OFF
SV	service 'T' for translating the CICS-TRANS-Id's

Examples:

```
UCTRN ON
UCTRN ON T
UCTRN OFF
```

Purpose:

Switch translation on or off.

Description:

CICS

The entry ON in F2 sets for the screen, at which the program is executed, the terminals parameter UCT on.In this case the TP monitor translates all inputs of lowercase letters into uppercase letters.

The entry OFF switches the translation off. Thus it is possible to enter also pseudo conversational text inputs in upper- lower case.

The service 'T' can be used, if only the CICS Trans-Id's are to be translated.

BATCH

UCTRN ON switches the translation for all printouts on.
UCTRN OFF switches the translation off.

UPDAT(E) modify data record in a file

OP F2 — for Temporary Storage Queues or HL1/QPG datasets

OP	UPDAT or UPDATE must be entered
F2	name of the file

F1 OP F2 EG — for disk files

F1	Name of the key files
OP	UPDAT OR UPDATE must be entered
F2	Name of the disk file
EG	Field, that has been read in via Input Division

Example:

```
UPDATE TSQ1
KEY UPDAT CPGWRK REC86
```

Purpose:

Modify a record on a file without using the Output Division.

Rules:

- For storage and Dataset, the structure defined in the Input Division may only contain alphanumerical and packted numerical fields.
- For disk files, the oparation is rstricted to 256 bytes. The field in EG has to be a field that is described as input in the input division.

WAIT waiting

OP (F2)

OP	WAIT must be entered
F2	numeric field for the duration of waiting

Examples:

```
WAIT                                  * wait 1 second
WAIT 5;                               * wait 5 seconds
WAIT SS;                              * wait SS seconds
```

Purpose:

Waiting for the timer.

Description:

With WAIT a timer is started, which runs after the number of seconds indicated in F2. If F2 is not indicated, then the TIMER is set to 1 second. The execution of the program is suspended up to the outflow of the timer.

WAIT enables to insert a tracing in time-consuming program flows to offer to CICS the possibility to serve other transactions. SO an abort with AICA of such quasi applications of batches in the CICS can be avoided.

WHEN indication of a condition

WHEN-DAT(E) date query standard format TTMMJJ in a condition

WHEN-DATI date query ISO format JJMMTT in a condition

OP F1 OC F2 (SV)

OP	WHEN, WHEN-DAT(E) or WHEN-DATI must be entered
F1	first matching field (indicatable)
OC	Operater = > < > = < = >< EQ GT LT GE LE NE
F2	second matching field (indicatable)
SV	boolean functions AND/OR can be established.

The WHEN statement indicates an alternative condition within a EVALUATE operation, see also operation EVALUATE.

WHEN-DAT and WHEN-DATI permit the query of date values in the standard (- DAT) or in the ISO format (- DATI). With AND/OR date queries can be combined with each other and with 'normal' WHEN queries.(See IF-DAT/IF DATI for further description).

WHEN OTHER indication of a condition

WHEN OTHER

The WHEN OTHER statement indicates the statements, which are executed within a EVALUATE operation, if all preceding WHEN-conditions were not fulfilled.

WRITE add data record to a file

OP F2 for Temporary Storage Queues or HL1/QPG datasets

OP	WRITE must be entered
F2	name of the file

F1 OP F2 EG for disk files

F1	Name of the key files
OP	WRITE must be entered
F2	Name of the disk file
EG	Field, that has been read in via Input Division

Example:

```
WRITE TSQ1
KEY WRITE CPGWRK REC86
```

Purpose:

Add a record on a file without using the Output Division.

Rules:

- For storage and Dataset, the structure defined in the Input Division may only contain alphanumerical and packted numerical fields.
- For disk files, the oparation is rstricted to 256 bytes. The field in EG has to be a field that is described as input in the input division.

XFOOT calculate the sum of an array

OP F2 EG (SV)

OP	XFOOT must be entered
F2	name of a numeric array
EG	name of the total field
SV	'H' or 'ROUnded' for rounds

Examples:

```
XFOOT FGN SUMME
XFOOT FGN FELD ROUND
```

Purpose:

Count the total of the elements of a numeric array.

Description:

After execution of the operation the total of the elements of the numeric array indicated in F2 is indicated in EG. With the service function can be determined whether the result is rounded or not.

Table of operation codes

Opcode	Coding	Comment
=	EG = F2	
+	EG = F1 OP F2	
-	EG = F1 OP F2	
*	EG = F1 OP F2	
/	EG = F1 OP F2	
ACCEPT	F1 ACCEPT FN	SV = Check,Prot,Upd RC = CPGFRC
AFOOT	AFOOT FG EG (SV)	SV = ROUnded
AVERAGE	OP	
BREAK	OP (SV)	SV = All
BEGSR	F1 OP	
CALL	OP F2 (EG)	
CHAIN	F1 CHAIN FN	SV = Check,Prot,Upd RC = CPGFRC
CHANG	F1 CHANG F2	
CHANGE	F1 CHANGE F2	
CHECK	OP FN	RC = CPGFRC
CHECK-VAR	OP FN	RC = CPGFRC
CLEAR	OP	
CLOSE	OP FN	RC = CPGFRC
COM-REG	OP F2	
COMRG	OP EG	
COMPUTE	OP	Formule
CONT	OP	
CONTINUE	OP	
CONVERT	OP F2 (DY) (EG) (SV)	DY = INTO SV see CONVT
CONVT	OP F2 (DY) (EG) (SV)	SV = Char Date Hex Low Sec Time X Year
DEBUG	OP (F2)	F2 = ON or OFF
DELC	OP F2 EG (SV)	SV = Array
DELET	OP (F1) OP F2	
DEQ(UEUE)	F1 OP (SV)	
DISPLAY	F1 OP (F1) OP EG	
DO	OP (DY) (F1) (F2 DY) (DY EG)	DY = LOOP FROM TIMES WITH
DO UNTIL	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
DO UNTIL-DATe	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
DO UNTIL-DATI	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
DO WHILE	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
DO WHILE-DATe	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE *
DO WHILE-DATI	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
DSPLY	F1 OP oder (F1) OP EG	
EDIT	OP EG	
ELIM	OP F2 EG	
ELIMINATE	OP F2 (DY) EG	
ELSE	OP	
END	OP (F2)	
END-EVALUATE	OP	
ENDDO	OP (F2)	
ENDEV	OP	
ENDIF	OP	
ENDPR	OP	
ENDSR	(F1) OP	
ENQ(UEUE)	OP F1 (SV)	
EVALUATE	OP	
EXCPT	OP FN	
EXHM	OP F2 (EG) (SV)	EG = Data channel, SV = I,T
EXHM-VAR	(ON) OP F2	
EXIT-TRANS	OP F2	
EXITD	OP EG (SV)	SV = T
EXITI	OP F2	

Opcode	Coding	Comment
EXITP	OP F2 (SV)	
EXITP-VAR	OP EG (SV)	
EXITS	(F1) OP F2 EG (SV)	
EXITT-VAR	OP EG	
EXITT	OP F2	
EXPR	(F1) OP F2 (SV)	
EXPR-VAR	OP EG (SV)	
EXSR	(ON) OP F2 (BD)	
FILL	OP F2 (DY) EG	DY = INTO
FIND	(F1) OP F	
GETCHANNEL	OP	
GETHS	OP	
GET-UPDATE	F1 CHAIN FN	
GO (TO)	(ON) OP F2 (SV)	
IF	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
IF-DATe	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
IF-DATI	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
IF-DATI	OP F1 OK F2 (BO)	
IF DATK	OP F1 OK F2 (BO)	
JLB	OP EG	
JRB	OP EG	
JRC	OP F2 EG	
JRZ	OP EG	
LEFT-SHIFT	OP EG	
LIST	(F1) OP F2 (EG) (SV)	SV = I,P
LIST-VAR	OP EG (SV)	SV = I,P
LOADT	OP F2	
LOADT-VAR	OP F2	
LOKUP	OP F1 OC F2	OC = > < >= <= >< EQ GT LT GE LE NE
MAP	OP F2 (SV)	SV = Low
MAP-VAR	OP EG (SV)	SV = Low
MAPD	OP F2 (SV)	SV = Low,Clear,I,S
MAPD-VAR	OP EG (SV)	SV = Low,Clear,I,S
MAPI	OP F2 (SV)	SV = Low,Clear,I,S
MAPI-VAR	OP EG (SV)	SV = Low,Clear,I,S
MAPO	OP F2	
MAPO-VAR	OP EG	
MAPP	F1 OP F2 (SV)	SV = Aft,Bef,S
MAPP-VAR	F1 OP EG (SV)	SV = Aft,Bef,S
MOVE	OP F2 (DY) EG	DY = INTO TO
MOVE-ARRAY	OP F2 (DY) EG	DY = INTO TO
MOVE-LEFT	OP F2 (DY) EG	DY = INTO TO
MOVE (R)	(ON) OP F2 (DY) EG (SV)	
MOVE-REST	OP EG	
MOVE-RIGHT	OP F2 (DY) EG	DY = INTO TO
MOVEA	OP F2 (DY) EG	DY = INTO TO
MOVEL	OP F2 (DY) EG	DY = INTO TO
MOVEN	OP F2 (DY) EG (SV)	DY = INTO TO, SV = A,L,N,R
MOVEV	OP F2 (DY) EG	DY = INTO TO
MVR	OP EG	
OPEN	OP FN (SV)	SV = Inp, Upd, Out
PARAMETER	OP F2 (EG)	
PARM	OP F2	
PERFORM	(ON) OP F2 (BD)	
PROG	OP F2 (EG)	EG = Library
PROG-VAR	OP EG	
PROGRAM	OP F2 (EG)	EG = Library
PROT	OP F2	
PROTECTION	OP F2	

Opcode	Coding	Comment
PURGE	OP F2	
RANDOM	OP FN	
READ	(F1) OP FN (SV)	RC = CPGFRC, SV = S
READ-BACK	(F1) OP FN	
READB	(F1) OP FN	RC = CPGFRC
READB-PAGE	(F1) OP F2	
READI	OP FN EG	EG = record identification
READP	(F2) OP F2 EG	
RECEIVE	OP F2 (SV)	SV = Low
REPLACE	OP F2 EG	
REPLC	OP F2 EG	
RIGHT	OP EG	
RIGHT-CHAR	OP F2 EG	
RIGHT-ZERO	OP EG	
RNDOM	OP FN	
ROLL	OP EG	
ROLL-BACK	OP EG	
ROLLB	OP EG	
SAVET	OP F2	
SAVET-VAR	OP F2	
SCAN	F1 OP F2 EG (SV)	
SCREENDUMP	OP (F2)	
SDUMP	OP (F2)	
SELCT	OP EG	
SELECT	OP EG	
SEND	OP F2	
SET-LIMIT	F1 OP FN	RC = CPGFRC
SETLL	F1 OP FN	RC = CPGFRC
SORT	OP F2 (SV)	SV = Bla,Desc
SORTA	OP F2 (SV)	SV = Bla,Desc
SQL	OP Statement	continuation lines in column 72 '+'
SQRT	OP F2 EG (SV)	SV = Round
SQUARE-ROOT	OP F2 EG (SV)	SV = Round
SYNCP (OINT)	OP (SV)	
TAG	F1 (OP)	
TASK	OP (F2) (EG) (SV)	SV= A, B, I, S, T
TASK-VAR	OP EG (SV)	SV = A, B, I, S, T
TESTF	OP F2 EG (SV)	SV = L
TEST-FIELD	OP F2 EG (SV)	SV = L
TIME	OP (EG)	
TWALD	P (F2)	
TWASV	OP (F2)	
TWASV-VAR	OP (F2)	
TWA-LOAD	OP (F2)	
TWA-SAVE	OP (F2)	
UCTRAN	OP F2 (SV)	SV = T
UCTRN	OP F2	F2 = ON,OFF
UPDAT	OP FN	RC = CPGFRC
UPDATE	OP FN	RC = CPGFRC
WAIT	OP (F2)	
WHEN	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
WHEN-DATe	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
WHEN-DATI	OP F1 OC F2 (SV)	OC = > < >= <= >< EQ GT LT GE LE NE
WHEN OTHER	OP	
WRITE	OP FN	RC = CPGFRC
XFOOT	OP F2 (DY) EG (SV)	DY = TO, SV = Round

OP = Operation   OC = Query   SV = Service   () for choice   F1 = Factor 1   2 = Factor 2
EG = Result   DY = Dummyword   FN = Filename   RC = Return code

H L 1 5000

HL1 programming 5010

HL1 programs are written in the programming language CPG. HL1 is an expansion of the CPG, that is disposable up from the service level CPG3.

The HL1 compiler is necessary for the generation of HL1 programs, and is executed with EXEC HL1. This compiler includes CPG and can also generate 'normal' CPG programs via an entry in the OPTIONS. The necessary entry is MAIn or ROOt for main program or root phase. These two entries are identical in their function. (So: HL1 main program = CPG program)

HL1 program modules can be called from every CPG program and from every HL1 module with the operation 'EXECUTE HL1 Module'

```
      EXHM NAME
```

So factor 2 contains the name of the HL1 module which should be processed.

HL1 modules are object compatible, that means, that they can be processed without any alteration and without new compilation under every TP monitor for which a CPG interface is available. Phase names for HL1 programs are up to 6 positions long and freely selectable.

The phase names for HL1 modules will be extended by the compiler on the left by the literal 'HM', that means, the program XZEIL1 will be cataloged as HMXZEIL1. So all HL1 modules are found under 'HM' in the Core Image Library, so that a differentiation from all other programs is given.

HL1 modules can not be processed independently.

The call only takes place from a main program or another HL1 module with help of the operation EXHM. Main programs are also produced by HL1 (OPTIONS MAIn or ROOt).

HL1 programs are written reentrant, by the generated assembler source program contains only pure assembler instructions and is free from all input/output macro instructions and free from data fields.

All macro instructions are controled in a central interface, that offers if necessary a fast adaptation possibility in case of a change of the basis software, so that a high degree of flexibility is given to the applications.

The data channel 5020

The data fields of a CPG program are positioned in the Transaction Work Area (TWA), which is disposed separately for each terminal during the online processing.

Each HL1 module has additionally an own Transaction Work Area (PWA Private Work Area).

HL1 modules are independent of each other and can exchange data among each other.

The programmer can transfer a data channel into the private TWA of this module when calling a HL1 module. The name of this data channel is indicated behind the module name:

```
     EXHM MODUL KANAL1
```

In this case, the data channel will be described in the input descriptions of the calling program:

```
      I
         FILE KANAL1 HS DD
            1 6 CUSTNO
            7 24 NAME
         P 25 26 0 NUMBER
```

Numerical data fields are always packed in the channel!

The data channel in the called program must be described in the first positions of the data division. So the use of the data dictionary is recommended.

The fields CUSTNO, NAME and NUMBER are transferred into the private TWA of the module MODUL with the exit to the program (EXHM). After the execution of the module MODUL, the eventually changed field contents will be retransferred.

The entry HS in the input record description causes, that no optimization is processed for the input fields.

Data description 5030

The data is described in the called module at the beginning of the data division.

```
     -D
        DEFINE KANAL1
```

In the data channel, numerical fields are always packed!

The data channel should not contain any empty spaces, because in the channel, the data is transferred on the base of the position, not on the base of the field name!

HL1 Processing 5050

HL1 modules can be called from every CPG online program and from every HL1 Batch program.

HL1 modules are no independently processable programs. They are comparable with subroutines, because they can only be executed under control of a higher program. The difference to conventional subroutines is, that they are not connected with the calling program, but are loaded to the program according to the need at processing time.

So HL1 modules do not lie (like subroutines) several times in the library for processable programs as components of many programs, but only once in each case.

HL1 programs need much less place in the library than conventional programs.

For the online processing, HL1 modules will be loaded alternatively under control of the TP monitor or into a program pool of the HL1 Central

Routine Library. The program pool is initialized automatically with the first call of a HL1 module. All further modules will be loaded into the Central Routine Library in each case at the first call.

The Central Routine Library is a central library for processable subroutines, which can be queried by all programs running in the TP partition.

HL1 modules do not have to be entered into the program tables of the TP monitor, however, an entry in a HL1 table is necessary. With this entry, only the name of the module is meant. All necessary parameters for normal program tables are determined by HL1 it self.

VSE users can let manage their HL1 modules by CICS. Thereby decisive advantages emerge during the test phase, but only if the modules are relatively big. CICS tries to use this place again with a newcopy, if the HL1 module has not been changed above a 2K boundary. So the HL1 call and the processing of a module are not influenced.

Peculiarities for the file processing with HL1 5060

In principle there is no difference to the file processing in the CPG for the file processing within the HL1 main program and within the HL1 modules. There are two types of modules: Standard modules and dataset modules (OPTIONS DAT or PWA).

Standard modules should only be used, if they can work without file accesses. Standard modules set the used working area (PWA) free after each call.

Dataset modules are always used, if files or data bases are to be processed. For such modules, the PWA (private Work Area) remains within the calling task, that means that the contents of all user fields are available (OPTIONS DATaset). At the OPTIONS entry PWA, the user fields are initialized in the PWA, only the CPG internal fields remain.

For sequential processing with dataset modules, the performance improves online as well as in batch. But it is to note, that the PWA, that remains in these modules, requires place in the system. This will be noticeable especially at the dialogue programming. Furthermore, dataset modules use in each case own VSAM strings for READ or CHAIN(U).

Private HL1 Libraries 5070

HL1 Modules will not be entered into the program tables of the TP interface because they are controled and administered by the HL1 Central Routine Library. VSE users can nevertheless let administer their HL1 Modules by CICS. Hereby advantages emerge during the test phase or if the modules are especially big. CICS tries to use this place again with a newcopy, if the HL1 module has not been changed above a 2K boundary.

However, an entry of these modules into a HL1 program table is necessary. The entry necessitates 4 bytes per module and will be generated with the HL1 table generator (HL1HTG).

The size of the program table is limited to 4K bytes. So a HL1 program table can not contain more than 1024 entries.

The sequence of the entries is requested for the processing. A later alteration of the sequence would necessitate the new conversion of all programs, which call modules from this table. It is meaningful, to estimate the later need already before the installation. If there are expected more than 1000 modules for the end extension, it is proposed to install private libraries already while the installation for the different applications. These libraries also offer a higher security for program tests and for the program maintenance, because possible mistakes rest limited on the own application.

At the installation of private libraries, the program modules are sorted according to their use. If the modules are clearly application connected, for example cash discount calculation, reading article data and so on, these modules will be entered into the private library for this application, for example bookkeeping, order processing etc. If the modules are application neutral, for example program description, screen header and so on, these modules are entered into the table for the general library that is added to each private library.

Private libraries will be indicated by an one digit suffix. A pattern table for a general library and a private library with the indicator 'A' is delivered with a CPG3 installation. The described HL1 libraries are laid out for a general library and at most 23 private libraries.

While using program tables for private libraries, the column 22 of the H card must get an corresponding entry for a HL1 compilation.

The OPTIONS parameter is HL1 or LIB.

Example:

```
     OPTIONS LIB H
```

to use modules from the general library and the private library H.

Modules, that are delivered with the CPG3 installation, (for example the standard interfaces to other Lattwein products) are part of the private library H.

HL1 Datasets 5080

HL1 datasets are a subset of the HL1 modules. In principle, they do not differ from other HL1 modules.

HL1 Datasets are especially suitable for the file independent programming, because they are called with accustomed file operations (like READ, CHAIN,...) for the file or data base processing without additional programming.

Following points are to note for HL1 datasets:

- HL1 datasets must work with the dataset logic, that means the parameters DAT or PWA must be set in the OPTIONS.
- The file definition for a HL1 dataset has 'HL1' or 'HL1DS' as unit (and should be deposed in the data dictionary).
- The data transfer from the calling to the called modules and back can be processed very simply via the HL1 data channel. HL1 data channels contain only alphanumerical and packed numerical values. Data channels should not contain any empty space and should be described in the data dictionary. They are described in the calling program in the Input, in the called program in the first positions of the data division.
- The data channel must contain the four digit alpha field CPGHIC (HL1 interface control) that controls the dataset processing.
- All file operations are supported for the HL1 datasets.
- The programmer, who works with a HL1 dataset, only has to code Input Division and Procedure Division. Files will be changed with the operations UPDAT, WRITE and DELET.
- The entry of a key field for the file operation is not necessary, because the key must be transferred in the data channel.

Programming with HL1 Datasets 5090

The following example shows a HL1 main program with the possible file accesses via a HL1 dataset.

```
OPTIONS ROOT PHASE PROGRAMM;
-F;
    FILE SAMPLE
-D;
    FG 20 * 70
-I;
    FILE SAMPLE HS DD;          * Data dictionary generates the
        1 14 KEY;                * three following statements:
       15 100 SATZ
      101 104 CPGHIC
-C;
       CHAIN SAMPLE
       CHAIN SAMPLE CHECK;      * without reading of data
       CHAIN SAMPLE UPDATE
       CHAIN SAMPLE P;          * Check for Update
       CHECK SAMPLE
       CLOSE SAMPLE
       DELETE SAMPLE
       OPEN SAMPLE
       RANDOM SAMPLE
       READ SAMPLE
       READ PAGE SAMPLE FG;     * page wise into an array
       READ BACK SAMPLE
       READB BACK SAMPLE FG;    * page wise backwards into an array
       SETLL SAMPLE
       UPDAT SAMPLE
       WRITE SAMPLE
```

The file description has as unit HL1.

A data channel is necessary for the transfer of the fields. This corresponds to the input statements for the file processing. Its name must be equal to the phase name of the HL1 dataset module. The key word HS (for HL1 structure) is indicated in the input description.

The data channel describes the fields of the datasets. Additionally it must contain the 4-places HL1 interface control field CPGHIC.

Data channels should be described in the data dictionary!

The File Return code is retransferred in the internal field CPGFRC.

The following example shows a HL1 dataset module for a 1:1 file processing:

```
-  OPTIONS DATASET PHASE SAMPLE;
-   FILE CPGWRK;
 -  -D;
-    DEFINE CPGWRK TYPE DS;      * Data channel will be generated
                                 * like follows:
        KEY   14;
        SATZ  86;
        CPGHIC 4;
-    ORG CPGHIC;                 * Redefinition of CPGHIC
-       OC  2;                   * HL1 interface opcode
-       RC  2;                   * HL1 interface return code
-       ORG;                     * End of the redefinition
- -I;
-    FILE CPGWRK DD TYPE DS;     * Dictionary generates therefrom:
         1  14 KEY;
        15 100 SATZ;
- -C;
-    EVALUATE;
-      WHEN OC = 'B ';           * READ BACK
-        KEY READ BACK CPGWRK;
-        IF CPGFRC = 'EF';       * End of File
-           MOVE 'E' TO RC;
-        ENDIF;
-      WHEN OC = 'C';            * CLOSE
-        CLOSE CPGWRK;
-        IF CPGFRC >< '  ';      * Not found or not closed
-          MOVE 'F' TO RC;
-        ENDIF;
-      WHEN OC = 'D ';           * RANDOM
-        RANDOM CPGWRK;
-      WHEN OC = 'G';            * CHAIN
-        IF OC = 'G ';           * CHAIN
-          KEY CHAIN CPGWRK;
-        ELSE;
-         IF OC = 'GU';          * CHAIN for Update
-            KEY CHAIN CPGWRK UPDATE;
-          ELSE;
-            IF OC = 'GC';       * CHAIN Check (without reading)
-             KEY CHAIN CPGWRK CHECK;
-           ELSE;                * CHAIN Check for Update
-             KEY CHAIN CPGWRK P;
-           END;
-         END;
-       END;
-       IF CPGFRC = 'NF';       * Not found
-         MOVE 'G' TO RC;
-       END;
-     WHEN OC = 'L ';           * DELETE (delete)
-       EXCPT LOESCHEN;
-     WHEN OC = 'N ';           * WRITE (new record)
-       EXCPT NEU;
-       IF CPGFRC >< '  ';      * Duplicate record
-         MOVE 'D' TO RC;
-       END;
-     WHEN OC = 'O ';           * OPEN
-      OPEN CPGWRK;
-      IF CPGFRC >< '  ';       * Not found or not open
-        MOVE 'F' TO RC;
-      END;
-    WHEN OC = 'R ';            * READ
-      KEY READ CPGWRK;
-      IF CPGFRC = 'EF';        * End of File
-        MOVE 'E' TO RC;
-      END;
-    WHEN OC = 'S ';            * SET LOWER LIMIT
-      KEY SETLL CPGWRK;
-    WHEN OC = 'U ';            * UPDATE
-      EXCPT AENDERN;
-    WHEN OC = 'Z ';            * CHECK
-      CHECK CPGWRK;
-      IF CPGFRC =  '  ';       * !!!  Exeption !!!
-        MOVE 'G' TO RC;
-      END;
-  END EVALUATE;                *
- -O;
-  FILE CPGWRK DD TYPE DS       AENDERN
-  FILE CPGWRK            DEL  LOESCHEN
-  FILE CPGWRK DD TYPE DS ADD  NEU
```

Notes for the programming

The example shows a 1:1 dataset, that can contain as much additional statements as necessary.

The whole CPG functions are supported here of course, also the access to any other data base. The programmer of the application program can work with the usual reading and writing operations also for another data base.

Data channels should be described in the data dictionary, because they are required twice for the programming: In the input of the calling program and in the data definition of the called program.

For 1:1 datasets, the dataset structure is often identical with the file structure, but it must contain the field CPGHIC additionally. In this case at all three positions (data set channel input in the calling program, channel description in the called module and file input descriptions in the called program) can be worked with the same data dictionary structure. Thereby CPG generates the CPGHIC field not in the input for the VSAM file.

4000 bytes can be processed approximately for a logical file access via a dataset.

The first statements of the data division must correspond with the data channel of the calling program. Fields are allowed to be named differently, the position in the data channel controls the data transfer.

The interface control field CPGHIC must be indicated in the data channel and controls the dataset processing: The type of the file access will be transmitted via the first two positions of the CPGHIC field to the dataset module. The 3.position is used internally by CPG.

The 4.position is used for the return code, that is moved (automatically) into the field CPGFRC of the calling program.

The application of the HL1 interface control field CPGHIC is described in the following table.

```
CPGHIC byte 1: Operation code:      'B' READB
                                    'C' CLOSE
                                    'D' RNDOM
                                    'E' EXCPT
                                    'G' CHAIN
                                    'L' DELET
                                    'N' WRITE
                                    'O' OPEN
                                    'R' READ
                                    'S' SETLL
                                    'U' UPDAT
                                    'Z' CHECK
```

CPGHIC byte 2: Opcode extension: first letter of the service function:

```
     'C' for CHAIN CHECK
     'U' for CHAIN for UPDATE
     'P' for CHAIN CHECK for Update

     'I' open for Input
```

CPGHIC byte 3: used internally

CPGHIC byte 4: Returncode: is to be set in the HL1 dataset
by the programmer:
Valid values:
'D' Duplicate record
'E' End of file after sequential READ
'F' File error at Open and Close
'G' record not found at CHAIN or CHECK

HL1 Batch Programs 6300

Batch programs can also be produced with HL1.

Therefore only BATch has to be entered into the OPTIONS.

Batch programs are intended for a program size up to 20K and a TWA size up to 8K. With the OPTIONS parameters BIG and 12K, considerably bigger programs can be written.

The Batch programs work like CPG- and HL1 online programs with a Central Routine Library. The Batch Central Routine Library is not component of the program but will be loaded to the program at the processing time.

The data areas CSA, TCA and PIW (interface areas) will also be requested with the program start. So the HL1 Batch programs become smaller than comparable RPG- or COBOL programs. So the space required in the Core Image Library will be considerably reduced.

For the moment, the following file types are supported in the Batch:

READER for the card input of SYSIPT.

PRINTER for the print output on SYSLST. In the interface two different DTF's are disposed for the print output:

a) standard DTFDI (device independant), which allows however only the output up to 120 positions. In this case, the skip control takes place via the L-card. (FORMS DIVISION)

b) furthermore DTFPR, which allows an output up to 132 positions. This DTF is used, if the file name begins with 'PRIN'.

As much printers as wished that are distinguished with the SYS number can be indicated.

PUNCHER for the punch output on SYSPCH.

KSDS for VSAM index files with fixed or variable record length.

This statement can also be used for VSAM PATH files, if the access takes place via an alternative index. Here, the path name has to be entered into the FILES DIVISION, and the key length of the alternative index has to be indicated.

ESDS for sequential VSAM files with fixed or variable record length.

RRDS for VSAM RRDS files (have always fixed record length)

Disk for sequential disk files. The key length in the file description must remain empty.

TAPE for tape files.

DL1 for data base accesses to DL1.

HL1 for the processing of HL1 datasets.

STORAGE for the processing of Temporary Storage. Single records as well as queues are supported.

TABLE for the processing of tables with CPG operation FIND.

Furthermore CPGTCT can be used as Temporary Storage area for reasons of compatibility to CICS programs and -modules.

The files are opened automatically with the program start and closed with program end, if 'NO OPEN' is not indicated in the files division. In this case, the file will be opened and closed per program only with the instructions 'OPEN' and 'CLOSE'.

For the READER you have to note, that the condition 'end of file' must be tested via the query of CPGFRC. Reading after end of file would read '/*' or '/&' and lead to a system error.

If a printer is indicated, so the used DTF can be selected with the printer name. Skips to channel will be controled at DTFDI via L Card, (FORMS DIVISION) and at DTFPR via FCB. If a channel is not defined, then the program cancels with an I/O error. The overflow switch 'OF' can be used to query the page overflow.

The switch 'OF' is set, if the overflow line was overstepped. The overflow is defined by the size of the list document (number of lines per page) or by the channel 12 in the FORMS division. If no channel 12 is indicated, so OF is set, if the form length was overstepped.

It is recommended, that the first print output contains a skip to channel 01, because otherwise the control with the 'OF' switch can be deficient.

For VSAM files, 'I', 'O' or 'U' will be entered for the in-/output mode in the data dictionary or in the program in the files division. With 'I' the file is only opened for input, with 'O' and 'U' for output. That is important for VSAM files with share option 2. Files, that are for example opened in the CICS for output, can afterwards only be opened in the Batch for input. An opening for output in the Batch would lead here to a VSAM open error and so to a program abend.

For VSAM files, the option REUSE is supported. Therewith working files can be produced, without a DELETE/DEFINE being necessary.

The file processing logic is the same in the Batch as for the online Processing.

Different to online processing, records in the Batch can also be updatet or deleted sequentially.

For **adding** records, note the following:

- Records in KSDS files can be added directly and sequentially in key sequence. For the sequential adding, the records must be available in rising key sequence. Records can be inserted between existing data records. If a record with the indicated key already exists, so 'Duplicate Record' is set into the field CPGFRC and as switch 'DR'.

- Records in ESDS files will be added sequentially at the end of the file. It is not possible to set records between existing records.
- Records in RRDS files can be added sequentially and directly. For the sequential adding, the records are added at the end of the file. For the direct adding, records can only be inserted, if no record exists for the indicated record number, for example if the record has been deleted before. If a record with the indicated record number already exists, so 'Duplicate Record' is set into the field CPGFRC and as switch 'DR'.

For the addition, 'ADD' must always be entered into the output. This is also valid for the first loading of files.

Deleting records is only possible in KSDS and RRDS files. The records can be deleted sequentially as well as directly.

A file must always be opened in the program, in which it is processed, otherwise a system error will be reported during the execution.

Restrictions and Notes for the Batch Version 6310

In principle all CPG operations are supported in the Batch, exept for:

1. Operations, that are specifically conceived for online processing:

```
COMRG  DEBUG  DEQ    ENQ    EREAD  EXITD  EXITI  EXITT  EXPR
LOADT  MAP    MAPD   MAPI   MAPO   MAPP   PROT   READI  SAVET
SDUMP  SYNCP  TESTT  TWALD  TWASV
```

1. The operation RNDOM*ALL is not supported.

2. Other operations that are not supported:

```
2. CALLD  CLEAR  EXITP  IFC  VBOMP  VSLCT
```

3. The operation CHAIN with the service UPD or 'P' does not lock the read record for the following CHAIN operations.

Following switches are not supported:

The screen switches **A1-A3 CL DE P1-PC Q1-QC SP** and **NI**

If operations, that are not allowed, are processed in a Batch programm, so the program abends with a formatted Dump.

Modules for Batch and Online Processing 6325

A module can be used online as well as in the Batch. If a different flow between Batch- and online processing should be necessary, so this can be controled with the field CPGTID (Terminal Id):

```
- IF CPGTID = '    ';              * Blank in the Batch (X'40')
 :
- ELSE;                            * Online: Terminal Id or
 :                                 *       X'00' for Non Terminal
- ENDIF;                           *             Tasks
```

Process a HL1 Batch Program 6330

The Batch Job can be constructed as follows:

```
// JOB HL1BATCH
   ... insert eventually ASSGN, DLBL and EXTENT cards
// EXEC BATCHPRG,SIZE=AUTO
   ... eventually data cards
/*
/&
```

Batch programs require the GETVIS area of the partition. Therefore the SIZE parameter must always be indicated in the EXEC instruction. The partition should be big enough (for example 256K + eventually necessary storage while using the Temporary Storage).

If errors should appear during a program processing, so this can be caused as follows:

1. Through an error in the application program, for example caused by a data error or through an operation that is not supported in the Batch. Hereby the program cancels and generates a formatted DUMP. Mostly, the error has to be searched in the data area (register 12).

2. Through an error in the interface, for example a VSAM error. Hereby a formatted Dump is also printed. The cause of the abend is printed in clear text at the beginning of the Dump area. VSAM errors and return codes have to be looked up in the manual 'VSAM messages and codes'. The statement of these codes is made decimally. A program abend follows after a VSAM error and the following job steps are not processed.

UPSI switches 6335

At VSE systems, switches may be set with JCL statement // UPSI xxxxxxxx. These switches can be queried in the program with U1 up to U8.

UPSI switches can be switched on with SETON and switched off with SETOF, this applies only for the duration of the program processing.

Eventually later running job steps are not concerned therefrom, they must be set eventually by further JCL statements.

UPSI switches can be set for example to select or exclude certain files from the processing. In this case, the operations OPEN and CLOSE have to be used.

Example:

```
- IF CONDITION U1
-    OPEN DATEI1
- ENDIF
```

Error messages during the compilation 6900

The diagnostics work in 2 steps:

1. Error identification in the Source code. The errors are documented b e l o w the faulty statement.

2. Error identification in the generated code. The diagnostic of the CPG1 compiler documents the errors a b o v e the faulty statement (via the OPTIONS alternatively also b e s i d e the statement);   if possible, a '$' indicates the faulty position.

Several syntax errors can lead to a wrong decision of the compiler that makes a further diagnostic useless. In these cases, the processing abends and the rest of the program is displayed without commentary. This cancel of the precompilation only concerns the OPTIONS and the files, data, and input division.

Syntax errors 6910

AND / OR NOT COMPLETE

Cause: An incomplete group of logically linked IF queries was coded. That means, that no IF follows after AND or OR.

Example:

```
- IF A = B AND
        - IF B > C AND;      * here is an AND too much
        -   VA = 'PRR'
        - ENDIF
```

DO UNTIL / WHILE MIXED

Cause: At a logically linked DO loop, DO UNTIL and DO WHILE were used mixed.

Example:

```
- DO WHILE ERRORCODE = ' ' AND
  UNTIL I > MAX
```

Action: Decision for DO UNTIL or DO WHILE and rewording of the loop condition, in the example:

```
- DO WHILE ERRORCODE = ' ' AND
- WHILE I <= MAX
```

END OF SOURCE

Cause: Assigning an alphanumeric literal (with =) to a numerical field.

Assignment an alphanumeric literal (with =), that is longer than eight positions, to an array element.

Half-, full- or duplicate word border for a numerical field.

Action: Check and rectify the faulty statements with help of the syntax rules in this manual.

ENTRY IS INVALID

Cause: A syntactically wrong indicator was indicated for the beginning of a new division.

Examples: For the output division

`'- O;'` was coded instead of **`'- -O;'`**
or **`'- -OUTPUT'`** instead of **`'- OUTPUT DIVISION';`**

Action: Rectify faulty division indicator.

ENTRY IS MISSING

Cause:
Wrong service function or missing 'INTO' at CONVERT-. In the data division, the number of digits is too big, or '*' has been forgotten at the array definition.

An operation code, for example LIST, expects further entries.

Action: Check and rectify the faulty statements with help of the syntax rules in this manual.

EVALUATE IN EVALUATE.

Cause:
Between an EVALUATE and the affiliated END-EVALUATE must not be coded any further EVALUATE.

FIELDNAME IS INVALID

Cause: at assignment with '=' invalid signs identified:

- assigned numerical literal begins with decimal comma or a decimal point: Not supported
- assigned alpha literal does not begin with ', but with another special character sign, for example ".
- assignment of a hexadezimal literal with = is not supported.
- for an extended array name, a name of the index field with a length of more than one byte was chosen.
- a blank is set between array and index

The complete support of extended array names is only given, if the Options contain a parameter LONg or if the column 100 of the standard header card (CPGSTH) contains an A.

FIELDNAME IS UNDEFINED

Cause: The result field of an assignment (with =) is not yet defined in the program code.

Action: The field has to be defined in the data division.

INDICATOR IS INVALID

Cause: An indicator has been coded for a field edition in the record regulation of the output division.

Example: **`- -O; FIELD SATZ ON 10`**

Action: Field editions can not be locked or selected with switches. Switches in the field regulations are allowed. Selecting for a certain edition is possible, if you work with the key word TYPE in the procedure division and output division.

INVALID CONDITION

Action: Check of the conditioning indicators and rectification. IF CONdition may not be logically linked with other IF queries.

INVALID LENGTH

Cause: An alphanumeric literal, that contains more than 26 places, was tried to be assigned to an alphafield with the operation =.

Action: Shorten faulty literals or code a field edition with EDIT via the output division instead of a direct assignment.

INVALID OPERATION

Cause: An invalid operation code was indicated.

Action: Check the opcode table in this manual and improve the operation code.

NUMERIC ENTRY EXPECTED

Cause: An alphanumerical word has been indicated at a position at which a numerical value is expected.

Example:

```
- FIELD 1 10
-      1 10 FIELD;
```

(`- FIELD 1 10` was indicated instead of `-      1 10 FIELD;`)

In a field description of the input division

Action: Check and rectify the faulty statements with help of the syntax rules in this manual.

NUMERIC ENTRY TOO BIG

Cause: A field with a length bigger than thousand was defined in the data division.

Action: Rectify the field length (the maximum is 256) with help of the syntax rules in this manual.

OPTIONS NOT CLOSED

Cause: SIMICOLON; FULLSTOP or the key word END are missing as conclusion of the options; point is indicated, but simicolon is expected.

Action: Set right record end sign.

OUTPUTPOSITION IS INVALID

Cause: With the parameter COLumn in the options, a column smaller than 7 or bigger than 51 was used.

Action: Check and rectify the options entry with help of this manual.

TOO MANY ENTRIES IN STMNTS

Cause: A statement has more entries, than is supported for the used operation code.

The error also appears, if no record end sign is set between two statements in a line. The error often appears before commentary statements.

Note: A line numbering at the right edge is eventually recognized as entry, because it will be read until position 79.

Remedy for this problem:

1. Finish each statement with a simicolon
2. Determine in the standard header card, that the line numbering should not be considered as program code. In this case, the line numbering will be replaced by blanks for the time of the precompiling. The entries in CPGSTH: 6 or 8 for six or eight places numbering.

Action: Check and rectify the faulty statements with help of the syntax rules in this manual.

TOO MANY FIELDS

Cause: The precompiler can not generate any more CP names from extended field names.

Action: The system programmer must increase the table CPG*CETBS.

TOO MANY STMNTS. IN LINE

Cause: The compiler finds a word at a position, where no more statement can begin, for example

- for the coding from position 8, a phase name is set up from position 75.
- for the coding from position 1, the coding line may be used up to the column 72.

Action: Check and rectify the faulty statements or entry in the standard header, to ignore always the last six or eight positions as source code.

Warnings 6920

After a CPG compilation without errors, it is possible, that a warning is printed. Then should be checked on the base of the below presented explications of the warning code, if the danger exists, that the program delivers undesirable results.

Warning:	Cause
CPG0010	a (E)READ or MAPx instruction with a service function AT, C, K or T has been encoded in a subroutine. an EXITP or EXPR instruction with a service function IND or SAVe has been encoded in a subroutine.
CPG0020	more than 100 list documents or file entries were used in the program. The field optimization as well as the list reference can not be processed according to the rules.
CPG0030	a dataset has been defined in the files division in a program, but the program works with the EXPR instruction and the service function IND or SAVe. If a dataset is also defined in the called program, so the warning can be ignored. However, if there is no dataset defined, then the saved fields for the dataset are corrupted at return of the called program.
CPG0040	it was tried to attract a structure from the data dictionary that does not exist. At warning CPG0040 the compilation will **CANCEL.**
CPG0050	a field was defined several times. The warning is output, if a file name is used as field name.
CPG0060	an overlay is not completely defined in the data division.
CPG0061	this warning is set, if a field has not been defined.
CPG0070	the generated program is not ready for ESA mode (see chapter 2970).
CPG0080	A CP name was generated from a word, that is identical with an opcode in the first five positions, for example PURGETPTC.
CPG0090	An OPTIONS parameter without meaning was found.
CPG0100	This warning will appear, if with EXCPT the name in factor 2 begins with ADD or DEL.
CPG0110	This warning will appear, if with a HL1-module with files in the options DAT/PWA is missing.

Error Messages in the Assembly 6930

Assembler errors detected during the compilation are usually **addressability errors**, because they can not be determined at the CPG compilation. That means, that either the program or the TWA is too big.

The page behind the programmer check list gives information about the faulty program part. (See below).

```
                              EXTERNAL SYMBOL DICTIONARY        PAGE 1
SYMBOL   TYPE            ID  ADDR    LENGTH  LD-ID
PROGN    SD (CSECT)     001 000000  005FFF

                             DUMMY SECTION DICTIONARY

SYMBOL   ID LENGTH

CPGTCADS 1FB 000FFF
```

For standard programs the length for PROGN (program name) must not be
bigger than X'5FFF' (24K) and CPGTCADS not bigger than X'FFF' (4K).

At programs with 8K TWA (OPTIONS ADD 1 or ADDQ) the program size must not
overstep X'4FFF' (20K) and CPGTCADS must not be bigger than X'1FFF' (8K).

AT OPTIONS BIG or 12K several CSECTS are generated:

```
SYMBOL      TYPE          ID         ADD                 LENGTH LD-ID

PROGN       SD (CSECT)    001        000000              001FFF
BILD        SD (CSECT)    002        000000              002FFF
CPGF002     ER (EXTRN)    003
CPGF002     SD (CSECT)    004        000000              002FFF
SSSELCT     SD (CSECT)    005        000000              000FFF
BILD        ER (EXTRN)    006
SUBRN       SD (CSECT)    007        000000              002FFF
CPGAUSG     SD (CSECT)    008        000000              002FFF
PPEDIT      SD (CSECT)    009        000000              000FFF
```

If BIG was entered, the length for PROGN must not be bigger than X'1FFF' (8K), the length of field EDITS and SELECT must not be bigger than X'FFF'(4K) and every further CSECT (subroutine) not bigger than X'2FFF' (12K).

If 12K has been entered, the program length as well as the subroutine length must not be bigger than X'1FFF' (8K) and CPGTCADS not bigger than X'2FFF!

Error Messages for the Processing 6950

For the processing of CPG programs, following error messages can appear on the screen:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
SYSTEM ERRORS       CPG 2.5 CLE        TT01   TST001   0002A0
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

FILE CPGTST  NOT OPENED

PLEASE CONTACT YOUR SUPPORT CENTER

          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          PF1 == => ABEND  +  PF3 == => IGNORE  +  OTHER == => END
          * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

While using the PF1 key, the program abends and a CICS dump will be generated. While using the PF3 key, a return is made to the used program.

This selection can be restricted at the CPG installation.

While using another function key, the program ends without dump. With the customer configuration can be controled, if always a dump is set, and an abnormal end of the program (because of DTB). The dump code and the error messages on the screen give information about the error cause. Following error messages can appear on the screen:

DECISION TABLE PROCESSING

If always only the first action is processed, eventually a numbering in the CPG program from column 73 is processed.

DIVIDE BY ZERO:

At the operation DIV, the field content for the divisor (factor 2) was zero.

DIVISION ERROR:

At the operation DIV, the rules for the field size have not been kept.

ESA ERROR

A HL1 module was called from an program prepared for ESA mode(see chapter 2970) that has not been compiled with CPG/ESA (CPG versions less than 1.6).

HL1 ERROR

At the EXHM is detected, that a module is called that does not work with the dataset logic and contains files, that are not defined in the higher main program.

The error is remedied, if you enter a parameter for dataset processing into the options (for example DATaset or PWA).

HL1 POOL ERROR

At the EXHM is detected, that either the HL1 pool is full or that a HL1 module can not be properly processed (for example because a private library has not been found at the processing time).

File XXXXX AREA TO SMALL:

- Add the adding of data records no more empty disk storages were found.
- No print output can be processed any more, there is no free transient data area disposable.

File XXXXX INPUT ERROR:

The system can not call up the corresponding unit.
(Hardware defect).

- With a print output was tried to output a record, whose length is bigger than the record length of the VSAM cluster DFHNTRA.

File XXXXX NOT AVAILABLE:

- An update has been tried, without having processed a CHAIN for this record previously (Put no Get).It was tried to change a not existing record with update.
- The file component between Define Cluster and CICSFCT does not correspond.
- The service for this access is not generated in the CICSFCP (Reg 9 = X'00') VSAM in the FCP not upported.
- The service for this access is not generated in the CICSFCT. (Reg 9 unequal X'00' VSAM delete and delete were not listed as SERVREQ).
- A SYNCP has been set, without previously releasing the files with RNDOM.

File XXXXX NOT IN THE FCT:

An input or output was processed for a file, that is not defined in the CICS-FCT.

File XXXXX NOT OPENED:

The file was not opened for the online processing. A CICS dump is not possible with this error messages.

File XXXXXXXX PROGRAM ERROR

An operation sequence (file access) should be processed, that the TP monitor forbids. Following operations must not be processed online for the same file:

Error code		Operation	Operation to be processed	
SEQ FU	after a	READ	UPDATE	must not be processed
SEQ FU	" "	READ	CHAIN UPD	"
SEQ FA	" "	READ	ADD	"
SEQ FL	" "	READ	DELET	"
SEQ EF	" "	EF	READ	"
UPD FC	" "	CHAIN UPD	CHAIN	"
UPD FR	" "	CHAIN UPD	READ	"
UPD FB	" "	CHAIN UPD	READB	"
UPD FU	" "	CHAIN UPD	CHAIN UPD	"
UPD FA	" "	CHAIN UPD	ADD	"

If these rules should not have been kept in your applications, this error messages can be suppressed for a transitional period.

In this case, a corresponding error message will be output on the system console and the programs should be reengeneered as quick as possible.

Example of such an error message on the console:

```
     ----------------------------------------------------------------
     F2 002 *** FILE CPGWRK PROGRAM-ERROR UPD FL
     F2 002 *** TRANSID TT02 PROGRAM TST002 TERM RZR1 ****
     ----------------------------------------------------------------
```

File XXXXX VSAM ILLOGIC:

This error message appears, if a VSAM error appears, that corresponds to no other error code. The VSAM error and return code is also indicated on the screen or can be looked up on displacement 2C in the CICS transaction dump in the transaction storage file (Length = D0) (for example 0008006C = returncode 8 and errorcode 6C). Too much same keys appeared for the alternative index processing, so that the record length of the VSAM-AIX was too small. VSAM error/return codes can be looked up in the IBM manual VSE/VSAM messages and codes (SC 24-5146).

WRONG INDEX:

An operation was processed with variable index. The content of the index field is either zero or bigger than the number of the defined elements of the corresponding array, or the index is too big at SETIX or SETIN.

READI ERROR

A READI was processed, without a READ, READB or CHAIN UPD having been processed before.

SQRT NEGATIV:

At the operation SQRT a negative field content was found.

TSQUE XXXXXXXX NNN TCA RCODE:

Errors appeared at the output of a Temporary Storage Queue.

XXXXXXXX indicates the name of the storage area, on which should be output.

NNN indicates the temporary storage return code (decimally). (Further details see CICS manual DFHTS TYPE=PUTQ).

The dump code from the CICS dump gives information about the error type.

Following codes can appear:

'CPGL' the command level interface is not loaded.

'ECPG' general CPG error messages

'ECPS' storage control. There was not enough storage available for the initialisation of a task.

'ECPT' terminal control. There was no storage available for a screen input or output.

'ECPW' the entry 'TWA size' in the CICS PCT is too small. At HL1 modules, private TWA has been overwritten per channel.

'NDLI' DL/I users. In the OPTIONS DIVISION the entry ADDressing D is missing.

'QSFI' At a MAP operation, the indicated Map in the QSF has not been found.

'QSFM' errors in the directory or phase not found.

'QSFS' Size detected.

'QSFT' a Map instruction was set in a Non Terminal Task. That abend code appears in the statistic.

'QSFV' the directory is full.

'QSFZ' there are too much fields in a Map.

'U098' in the CPG2 ..print programm, the output contains only X'00'.

'U099' in the CPG2..print program, the output is bigger than 1904 bytes.

Error Messages during the Processing 6970

During the processing of online programs under the TP monitor CICS following errors can appear:

Unplanned CICS shut down

Cause:	Storage violation (see storage violation).
Removal:	1. Print a dump. The cause is the transaction with more than 20 pages in the dump. After the restart, the program must be immediately disabled.

Storage violation

Cause: 1. TWA size too small (see unplanned CICS shut down). (Is checked at programs which were compiled with CPG Release 1.2. or higher)

2. The operations BEGAS and ENDAS were used and an assembler statement transfers data to an address outside the program.

3. The operation COPY was used and an assembler statement transfers data to an address outside the program.

4. at Temporary Storage two areas with different lengths have the same name.

5. Assembler error messages in the CPG program were not heeded.

6. Before a SYNCP instruction, files were not released with RNDOM.

7. For the record length for VBOMP, the Prefix has not been counted.

8. At the ADD function the same file has been called two times with EXCPT.

9. In the output, DIAG was used twice with indicators under the same file output.

Removal: Go to 'DFHCSA' in the CICS-DUMP. From 'storage +1' plus 4C the address of the last active transaction is indicated.

CICS Loop

Cause: 1. The print output in the CPG program has no positive output line.

2. Two EXCPT lines are coded behind each other without output field.

3. VSAM file errors (Task has been terminated at the CA split).

Removal: Cancel CICS and print Dump. For a file error, Forward Recovery must be used. At task errors the program has to be disabled immediately after the restart.

CICS Data Base Loop

Cause: A VSAM file with SHR 3 was destroyed in the Batch as well as online.

Removal: Cancel CICS. In register 1, in the partition SAVE-AREA, the address of the defective VSAM file (ACB) is indicated.
Determine online VSAM files with SHR 3:

In the FCT, the files have the entry 'ACCMETH = (VSAM, KSDS,KEY)'. Run a verify for the corresponding file.

At unsuccessful Verify a Delete/Define Cluster must be processed.

CICS terminal Dataset Loop

Cause: Local control units were not ready at CICS start.

Removal: Cancel CICS. Process IPL for VSE. Switch on local control units. Start CICS again.

Locked Transactions

Cause: An Update file is locked for other tasks.

For a conversational oriented program with DTB = YES in the PCT,
the instruction 'SYNCP' has been coded wrong or not been coded.

Removal:

1. CSMT Term,Outserv, all
2. Switch dump area
3. CSMT Term,inserv,all
4. Print dump area

For the locked tasks give the corresponding dumps to theapplication programmer. The corresponding file name is in the Task Dump 'TCA USER' from position X'84'.

VSAM Illogic Error

Cause: 1. Pseudo record was missing for ADD file.

2. Delete/Define was faulty.

3. At a VSAM file with variable record length, the four bytes offset has been forgotten.

4. At an ESDS/RRDS file the right RBA was not found.

Removal: The exact VSAM error code will be indicated on the screen.

CICS works unusually slowly

Cause: 1. VSIZE of the CICS partition is too small.

2. In the PCT, DTB = YES was indicated and in the CPG program no 'SYNCP' was given.

3. AMXT in the SIT is positioned too small.

4. In the CPG program many accesses are started on one or several files (VBOMP)

5. Priority of CICS has been given wrongly.

Removal: 1. Extend VSIZE.

2. Cancel CICS and print Dump. Examine CICS dump on occupied storage areas in the DTB area. Disable the determined program after the restart.

3. Increase the number of the active tasks with 'CSMT AMX'. Converse afterwards DFHSIT with new AMX.

4. Print CICS statistic. Check the listed files in the section 'file statistic'.

5. The priority for CICS must be set high by the operator.

Task abend with ASRA

Cause:

1. Wrong output pattern.

2. Read packed data without 'PAC'.

3. Read unpacked data with 'PAC'.

4. A numerical field contains no packed data.

5. Error in an assembler subroutine.

6. The task has been destroyed by storage violation

7. A CPG program was called from an assembler program with XCTL, thereby the TWA has not been initialized from the XCTL.

8. At the EXPR and CSA offset, the offset is not in the CPGUCCBA and consequently not in the CPGCCI or not in the CPGURSIT and consequently not in CPGWRK.

Removal:

1 to 5. Print dump, search faulty field and remedy the error cause.

6. Print dump and determine the storage violator. Disable the storage violator with DISAB. Load the task with 'CSMT NEW,PGRMID='.

7. Enter a XC 256(116,12),256(12) before a XCTL (for normall CPG-TWA), otherwise consider the offset in accordance (for example for datasets).

Task abend with AICA

Cause:

1. Program loop. A GOTO operation branches to a TAG lying previously and the loop will not be interrupted by a READ screen or a WAIT.
2. In the CPG program was branched to BEGSR or ENDSR in the subroutine with GOTO.
3. Without an input/output interruption, a routine is processed so often, that the maximum time determined by the CICS is reached.

4. Within a DO loop, the operation CLEAR was used.

5. At a DO WHILE operation an indicator was entered by mistake.

Removal:

1. Remove or interrupt the loop.

2. A TAG line must be defined for GOTO in the SR routine.

3. Set time higher in the CICS or interrupt routine by a 'WAIT'.

Task abend with ASCF

1. In the input for temp. storage group switches were used.

Task abend with ATNI

1. It was tried to output not representable signs onto the screen.

Task abend with APCT

1. A HL1 module has not been found.

Tables, Operation codes 7000

Tables 7000

Overview of the Procedure Division 7000

The following table gives an overview of the possible entries for the several operations. Compare therefore the syntax rules in the chapter 'Operations'. The general description of an operation is described there as follows:

```
----------------------------------------------------------------------
(ON)  OP                                                    (SV)  (BD)
      (F1)  OC  (OE) (DY) (F1) (DY) (OK) (F2) (DY) (EG) (LG)
----------------------------------------------------------------------
```

The abbreviations have the meanings that are described in the following list. Expressions in brackets can be used at choice, expressions without brackets are absolutely requested.

ON		condition query
OP		operation
	OC	operation code (ADD, READ, DO)
	OE	extended operation code (WHILE, UNTIL)
	F1	Factor 1
	OK	Operator (bigger smaller equal)
	F2	Factor 2
	EG	result field
	DY	Dummy word (IS FROM TO THAN THEN)
	LG	Length field (only for the OCs DLI and QSSA)
SV		Service query
BD		set conditions (switches 01 to 99)
BO		logical linking with AND and OR

In the following list, the different descriptions for all operations are listed.

```
(ON)     F1    ACCEPT    F2 (KW EG) (SV) (SV) BD
(ON)           AFOOT     F2 EG (SV) (BD)
          (ON)           AVERAGE   F2 EG (SV) (BD)
                    F1   BEGDT
                    F1   BEGSR
          (ON)           BREAK     (SV)
          (ON)           CALL      F2 (EG)
          (ON)      F1   CHAIN     F2 (KW EG) (SV) (SV) (BD)
          (ON)      F1   CHANG(E)  F2
          (ON)           CHECK     F2 (SV) (BD)
          (ON)           CLOSE     F2 (SV) (BD)
          (ON)           COMRG     EG
          (ON)           COM-REG   EG
          (ON)           CONT(INUE)
          (ON)           CONVERT   (F2 INTO) EG (SV)
          (ON)           CONVT     (F2 INTO) EG (SV)
          (ON)           DEBUG     (SV)
          (ON)           DELC      F2 EG
          (ON)     (F1)  DELETE    F2
```

```
         F1    DEQ(UEUE) (SV)
(ON)     F1    DISPLAY
(ON)     F1    DLI       F2 EG LG (SV) BD
(ON)           DO        (OE) (DY) (F1) (OK) (DY) (F2) (DY) (EG) (SV)
or             DO        OE F1 OK F2 (BO)
(ON)     F1    DSPLY
(ON)           DUMP      (F2)
(ON)           EDIT      EG ((TYPE) F2) (SV) (BD)
(ON)           ELIM(INATE) F2 EG
               ELSE
(ON)           END
               END-EVALUATE
(ON)           ENDDO
               ENDDT
               ENDEV
               ENDIF
        (F1)   ENDSR
               ENQ(UEUE)  F2 (SV)
               EVALUATE
(ON)           EXCPT      (F2) (SV) (BD)
(ON)           EXECUTE    F2 (SV)
(ON)           EXHM       F2 (EG) (SV)
(ON)           EXHM-VAR   F2
(ON)           EXITD      EG (SV)
(ON)           EXITI      F2
(ON)           EXITP      F2 (SV)
(ON)           EXITP-VAR  F2 (SV)
(ON)    (F1)   EXITS      F2 EG (SV)
(ON)           EXITT      F2 (SV)
(ON)           EXITT-VAR  EG
(ON)    (F1)   EXIT-SEND  F2 EG (SV)
(ON)           EXIT-TRANS F2 (SV)
(ON)    (F1)   EXPR       F2 (SV)
(ON)           EXPR-VAR   F2 (SV)
(ON)           EXSR       F2 (BD)
(ON)           FILL       F2 (DY) EG
(ON)     F1    FIND       F2 (BD)
(ON)           GETCHANNEL
(ON)           GETHS
(ON)     F1    GET-UPDATE F2 (KW EG) (SV) (SV) BD
(ON)           GO(TO)     F2 (SV)
(ON)           IF         F1 (DY) OK F2 (DY) (BO)
(ON)           IF         SV BD (BD) (BD)
(ON)           IF-DAT     F2    (BO)
         F1    IF-DATI    F2    (BO)
         F1    IF-DATK    F2    (BO)
(ON)           JLB        F2
(ON)           JRB        F2
(ON)           JRC        F2 EG
(ON)           JRZ        EG
(ON)           LEFT-SHIFT F2
(ON)    (F1)   LIST       F2 (DY) (SV) (EG)
(ON)           LIST-VAR   EG
(ON)           LOADT      F2 (SV)
(ON)           LOADT-VAR  F2 (SV)
               LOKUP      F1 OK F2
               LOOK-UP    F1 OK F2
(ON)           MAP        F2 (SV1) (SV2)
(ON)           MAP-VAR    EG (SV)
(ON)           MAPD       F2 (SV1) (SV2)
(ON)           MAPD-VAR   EG (SV)
(ON)           MAPI       F2 (SV1) (SV2)
```

(ON)		MAPI-VAR	EG (SV)
(ON)		MAPO	EG (SV)
(ON)		MAPO-VAR	EG
(ON)	F1	MAPP	F2 (SV1) (SV2)
(ON)	F1	MAPP-VAR	EG (SV)
(ON)		MOVE(R)	F2 (DY) EG (SV)
(ON)		MOVEA	F2 (DY) EG
(ON)		MOVEL	F2 (DY) EG (SV)
(ON)		MOVEN	F2 (DY) EG (SV) (BD)
(ON)		MOVEV	F2 (DY) EG (SV)
(ON)		MOVE-ARRAY	F2 (DY) EG
(ON)		MOVE-LEFT	F2 (DY) EG (SV)
(ON)		MOVE-REST	F2 (DY) EG (SV)
(ON)		MOVE-RIGHT	EG (SV)
(ON)		MOVE(R)	F2 (DY) EG (SV)
(ON)		MVR	EG
(ON)		OPEN	F2 (SV) (BD)
		PARM	F2
		PARAMETER	F2
(ON)		PERFORM	F2 (BD)
		PROGRAM(M)	F2 (EG)
(ON)		PROT	F2 (SV)
(ON)		PROTECTION	F2 (SV)
(ON)		PURGE	F2
(ON)	(F1)	QSSA	F2 EG LG (SV) OK (SV)
(ON)		RANDOM	F2
(ON)	(F1)	READ	F2 (SV)
(ON)	(F1)	READB	F2
(ON)	(F1)	READB-PAGE	F2 EG
(ON)		READI	F2 (SV)
(ON)	(F1)	READP	F2 EG
(ON)	(F1)	READ-BACK	F2
(ON)	(F1)	READ-PAGE	F2 EG
(ON)		RECEIVE	F2 (SV1) SV2)
(ON)	(F1)	REPLC	F2 EG
(ON)		RIGHT	F2
(ON)		RIGHT-CHAR	F2 EG
(ON)		RIGHT-ZERO	EG
(ON)		RNDOM	F2
(ON)		ROLL	EG
(ON)		ROLLB	EG
(ON)		ROLL-BACK	EG
(ON)		SAVET	F2 (SV)
(ON)		SAVET-VAR	F2
	F1	SCAN	F2 (EG) (BD)
(ON)		SCREENDUMP	(F2)
(ON)		SDUMP	(F2)
(ON)		SELCT	EG (TYPE F2) (SV)
(ON)		SELECT	EG (TYPE F2) (SV)
(ON)		SEND	EG (SV)
(ON)	F1	SETLL	F2
(ON)	F1	SET-LIMIT	F2
(ON)		SORT(A)	F2 (SV)
(ON)		SQRT	F2 EG (SV)
(ON)		SYNCP(OINT)	(SV)
	F1	(TAG)	
(ON)		TESTF	F2 (DY) EG (SV)
(ON)		TESTT	F2 BD
(ON)		TEST-FIELD	F2 (DY) EG (SV)
(ON)		TIME	EG
(ON)		TWALD	F2 (SV)
(ON)		TWASV	F2 (SV)

```
(ON)          TWA-LOAD   F2
(ON)          TWA-SAVE   F2
(ON)          TWALD-VAR  F2
(ON)          TWASV-VAR  F2
(ON)          UCTRN      F2
(ON)     F1   UPDAT(E)   F2 EG
(ON)     F1   USSA       (SV)
(ON)     F1   VBOMP      F2 EG
(ON)          VSLCT      F2
(ON)          WAIT       (F2)
              WHEN       F1 OK F2 (BO)
(ON)     F1   WRITE      F2 EG
(ON)          XFOOT      F2 EG (SV) (BD)
(ON)     EG   =          F1 (OK) (F2) (SV)
```

Dummy Words 7010

For the moment, following words are identified as dummy words and must
not be used as field names:

BE BEI BY
FROM
GIVING
INTO IS
WITH
ON
SHALL
THAN THEN TIMES TO

Reserved Names 7015

Following names are reserved and must not be used as field names:

- Operation codes

- Dummy words

- CPG internal field names
- **CPGxxx**
- Date and time fields like **UDATE, UTIME** etc.
- **CP0000-CP9999**, which can be generated internally for names with more than six places and internal Tags.
- **CP00-CP99**, which can be generated internally for long array names.

It is also recommended, to note the following reserved names:

Phases: With the different CPG service levels, a high number of phases are delivered. They start with

CPG......HL1......HMH......HMQ

Masks: The QSF maps delivered with the CPG installation have names which start with Q.

Storages: The storages used in the delivered CPG programs have names which start with TP or Q.

CPG screen attributes 7020

The following table shows the field characteristics in dependence of the CPG attribute for screen outputs.

			D: Double		Selector			
CPG	CICS	Protected	bright	Numeric	Pen	MDT	Color	Color
Attribut	Hex	/ None	N: Normal	al	detectable	set	Term	Printer
U	40	N					green	black
K	C1	N				J	green	black
Q	C4	N			J		green	black
B	C5	N			J	J	green	black
W	**50**	**N**		**J**			**green**	**black**
E	**D1**	**N**		**J**		**J**	**green**	**black**
J	**D4**	**N**		**J**	**J**		**green**	**black**
G	**D5**	**N**		**J**	**J**	**J**	**green**	**black**
A	C8	N	D		J		red	red
C	C9	N	D		J	J	red	red
N	D8	N	D	J	J		red	red
H	D0	N	D	J	J	J	red	red
X	**4C**	**N**	**NONE**					
D	**4D**	**N**	**NONE**			**J**		
Y	**5C**	**N**	**NONE**	**J**				
Z	**5D**	**N**	**NONE**	**J**		**J**		
0	60	J					blue	blue
1	61	J				J	blue	blue
2	E4	J					blue	blue
3	E5	J				J	blue	blue
S	**F0**	**J**						
R	**F1**	**J**				**J**		
I	**F4**	**J**						
O	**F5**	**J**				**J**		
4	E8	J	D				white	green
5	E9	J	D			J	white	green
P	F8	J	D				white	green
L	F8	J	D				white	green
M	F9	J	D			J	white	green
6	**6C**	**J**	**NONE**					
7	**6D**	**J**	**NONE**			**J**		
8	**7C**	**J**	**NONE**					
T	**7D**	**J**	**NONE**			**J**		

If there is no attribute indicated, **S** will be set as default.

CPG output Write Control Character 7030

The following table shows the WCC (Write Control Character) entries, that can be indicated in the record description of the output division, alternatively to a combination of the key words 'BEEp', 'MODified' and 'LOCked'.

CPG				Modified	Keyboard
Entry	WCC	Hex	Beep	Data Tag On	Locked
K	C0			On	Lock
L	C1				Lock
M	C2			On	
	C3				
N	C4		Beep	On	Lock
O	C5		Beep		Lock
S	C6		Beep	On	
H	C7		Beep		

Highest values for a CPG-program 7040

Standard or maximum program extension	24 K
biggest Batch program and HL1 modules	20 K
HL1 Batch standard or maximum TWA size	8 K
Standard for maximum size of the TWA	4 K
Standard for maximum length of an input or output record	8 K
Maximum size of the TIOA (Terminal I/O-Area)	
Maximum length of alphanumerical fields	256 Bytes
Maximum length of numerical fields (15 places)	8 Bytes
Number of entries in the name table	2000
maximum number of arrays	100
maximum number of files and LIST operations	100
maximum number of entries in data structures	1000
maximum number of structure	50
maximum number of files in the standard file table	100
max. number of DO or IF operations in the program	999
max. number of subroutines (BEGSR) in the program	999
max. interlocking of DO or IF operations	
maximum length of literals	
(including inverted commas for alphanumerical)	
numerical in the procedure division (F1, F2)	10 places
alphanumerical in procedure and output division	26 places

A wished program size with a maximum TWA-size of 8K can be used alternatively. This will be reached while using another adress block. Therefore you must use the entries 'BIG' and 'ADD x' in the options parameter list and note the resulting limits.

The same applies for an extension of the TWA on 12 K with the options parameter 12K. In this case, the necessary change of the adressing will be processed internally.

The max. input length of 8K may be extended via the field CPGFIS.

Temporary Storage of Data for CICS Users 7050

1. C W A Common Work Area 7050

Max. Length : 3584

Storage medium :	**Main storage**
Validity :	**System wide**
Definition :	System initialisation
Update :	**yes**
Adressing :	EDIT/SELCT CPGCSA
Aptitude :	For general information, which must be disposable sytem wide.
Important	Lay out conventions should be installed. The data will not be deleted automatically.

2. T C T User Area 7055

Max. Length	**255 Bytes**, by CPG program **245** Bytes.
Storage medium :	**Main storage**
Validity:	ordered to the data station
Definition:	In the TCT
Update :	**yes**
Adressing	EDIT/SELCT CPGTCT
Aptitude	For information, that has to be stored temporarly dependent on a terminal
Important :	Data will not be deleted automatically.

3. Temporary Storage 7060

Max. Length	32.000 records of 8 K length.
Storage medium	Main storage or 'Auxiliary' Subpool 5 or VSAM ESDS
Validity :	Dependent on name conventions
Definition :	From the first user, who stores data under a name.
Update :	Yes
Aptitude :	Information to be temporarily stored
Important	An existing TS record must not be extended during a modification. Therefore a list of the several TS queues should be kept.

Edit Codes for numerical Fields 7070

	Edit Code with 2 decimal positions	Pos. Value without decimal Positions	Pos. Value with 3 decimal Positions
Not edited	1234567	1234567	00120
1	12.345,67	1.234.567	0,120
2	12.345,67	1.234.567	120
3	12345,67	1234567	0,120
4	12345,67	1234567	120
A	12.345,67	1.234.567	0,120CR
B	12.345,67	1.234.567	120CR
C	12345,67	1234567	0,120CR
D	12345,67	1234567	120CR
J	12.345,67	1.234.567	0,120-
K	12.345,67	1.234.567	120-
L	12345,67	1234567	0,120-
M	12345,67	1234567	120-
Y			0/01/20
Z	1234567	1234567	120
Not edited	00012	000000	000000
1	120	0,00	0
2	120		
3	120	0,00	0
4	120		
A	120CR	0,00	0
B	120CR		
C	120CR	0,00	0
D	120CR		
J	120-	0,00	0
K	120-		
L	120-	0,00	0
M	120-		
Y	0.01.20	0.00.00	0.00.00
Z	120		

In the american spelling, decimal signs and thousand points are exchanged, and for the date a '/' is set instead of a '.'. To reach this description, the system programmer must take an corresponding entry in the system initialisation table CPGURSIT.

For the date editing with Y, the leading zeros can be suppressed. If a date field is equal zero, so a blank will be indicated. You reach this editing, if you set a protection star in addition to the edit code Y.

Vocabulary 7200

Dummy word	A dummy word is a connecting word in the text, which adjustes the text in the colloquial language, however without importance for the program. Example: The program text **IF OTTO IS LESS THAN HUGO** may also be written in the form **IF OTTO LESS HUGO**. The words IS and THAN are dummy words. Dummy words can be different depending on the division.
Field	A field is a particular number of storage places to take up data, which are grouped to an unit. A field must be defined before the start of the processing, that means, it must be communicated to the system, under which name the data should be called, of which type the data are, and how much characters the field should take at most. See also chapter 2, data fields.
Array	An array is a particular number of fields with the same field characteristics (name, type, length, decimal places). An array must be defined like a field before the processing. See also chapter 2, arrays. The several fields of the array are also called elements. An array can be processed group wise as well as element wise.
Card	If the expression card is used in the text of this manual, a text line or a CPG2 statement is meant.
Record	A record is a sequence of words, which are attached to each other according to the syntax rules, and are closed with a record end sign.
Key word.	A key word is a word, that always contains an instruction for the compiler. Example: In the text **IF OTTO IS LESS THAN HUGO**, the words **IF** and **LESS** are key words. Key words must not be used as names for variables. Key words may be different depending on the division.
Statement	A statement is a sequence of words, that causes the program to act in a certain way.
Word	In CPG, a word means a sequence of wished characters in the program text, which follows after a blank, and will be closed by a blank or a record end sign. A word can be at most 30 characters long.

Example for Words:

HUGO
1
END-OF-WORKING-DAY

Special forms for words are 'key words' and 'dummy words'.

Syntax Rules 7300

This chapter contains the syntax rules in short form. Detailed description, see chapter 3.

OPTIONS

```
----------------------------------------------------------------------
         KW (KW) (DY) (EN) (DY) (KW) (DY) (EN) (DY) ...
----------------------------------------------------------------------
```

KW Key word. As first key word, 'OPTIONS' is absolutely requested. Afterwards, key words can be completed with entries or dummy words and follow in any order according to chapter 3300.

DY	Dummy word
EN	Entry to a key word

```
----------------------------------------------------------------------
Example - OPTIONS MVS LOWER CASE LETTERS TITLE IS EXAMPLE;
           KW      KW  KW    DY   DY      KW    DY EN
----------------------------------------------------------------------
```

FILES

```
----------------------------------------------------------------------
1.       KW DN
2.       KW DN DN (DN) (DN) ...
3.       KW DN EN (EN) EN (EN) EN (EN EN EN) EN
----------------------------------------------------------------------
```

KW	Key word. As key word 'FILE' is requested or at multiple definition (case 2.) 'FILES'.
DN	File name
EN	Entry for manual description (see chapter 3400).

```
Example  - FILE KUNDEN;
         - FILES KUNDEN ARTIKEL;
      -    FILE TEST INPUT FIX 1000 200 5 KEY INDEX DISK
----------------------------------------------------------------------
```

DATA DIVISION

```
----------------------------------------------------------------------
         FN (EL OK) LG (DP)
or       KW ST (DY) (SA)
or       KW (FN)
----------------------------------------------------------------------
```

FN	Name of the field or the array
EL	Number of elements of an array
OK	Operator '*' multiplies length with number of elements.
LG	Length of the field in characters (places)
DP	Number of decimal places

----

or

KW	'DEFINE' for a data dictionary structure
ST	Structure name
DY	Dummy word 'TYPE'
SA	Record type of the structure

----

or

KW	'ORG' to position in the TWA
FN	Field name

```
Example  - VALUE 11 2;           * VALUE is num. 11 places 2 Dez.pl.
         - PAGE 24 * 80;         * Array 24 fields with 80 places each
         - TEXT 25;              * Alpha field with 25 places
or       - DEFINE CPGWRK;        * Definition of the structure 'CPGWRK'
or       - ORG;                  * Positioning
```

FORMS

```
----------------------------------------------------------------------
         KW FN (KW LG) ((KW) LN (KW) CH) ((KW) LN (KW) CH) ...
----------------------------------------------------------------------
```

KW	Key word: 'FORMS' is requested as first key word. 'LENGTH' indicates the following entry as form length in lines. 'LINE' or 'LINE' indicates the following entry as line number. 'CHAN' or 'KANAL' indicates the following entry as skip channel
FN	Name of the printer in the FILES DIVISION
LG	Length of the form in the line
LN	Line nubmer
CH	Channel

Example

```
         - FORMS DR01
         - FORMS DR02 LENGTH 36
         - FORMS DR03 LINE 3 CHAN 01 LINE 66 CHAN 12
         - FORMS DR05 LENGTH 36 003 01 066 12
         - FORMS DR06 LENGTH 36 ZEILE 3 KANAL 1 ZEILE 66 KANAL 12
```

INPUT DIVISION

Record description

```
----------------------------------------------------------------------
         KW DN (VAR) (BA)
----------------------------------------------------------------------
```

KW	Key word 'FILE' or 'FIELD'
DN	Name of the file (up to eight places long)
VAR	variable input positions for big files
BA	Condition query (see below)

Condition query (BA)

The condition query is different for the several file types. The general form depends on the file type:

```
----------------------------------------------------------------------
         Disk                    (TP) (BD) (CA) (AND) (CA) (AND) (CA)
            CA =                 PS (NOT) CD CH
         Screen                  (TP) (BD)
         Data Dictionary         TP (KW SA)
         Datastructure           TP (LG)
         HL1 Data channel        TP
----------------------------------------------------------------------
```

In this description mean:

TP	Filetyp (z.B. DS = Data structure)
BD	Condition (Switch 01 to 99)
SA	Record type (with the key word 'TYPE')
CA	Character query
PS	Position in the record
CD	Character code (Character Digit Zone)
CH	Character
LG	Length of the data structure
AND	Connection of several queries
NOT	Reversal (condition is not fulfilled)

Example

```
            - FILE BILD
            - FILE DISK
            - FILE KUNDEN DD
            - FILE DISK KF 11 1 NOT CHAR A 2 CHAR B 3 CHAR #
            - FILE ARTIKEL AA 01 1 C A AND 2 C R
            - FILE OTTO DS
            - FILE HUGO DS 200
            - FILE MYROOT DS
            - FILE XKANAL HS
            - FILE A0001 HS
            - FIELD HUGO
            - FIELD ALPHANUMERIC-STRING
```

Field description

```
----------------------------------------------------------------------
         (SF) VP BP (DP) FN (SP) (BD)
----------------------------------------------------------------------
```

SF	Storage form (packed, binary, logical)
VP	from position in the record (num. value max. 4 places)
BP	up to position in the record (num. value max. 4 places)
DP	Number of decimal places (character from 0 to 9)
FN	Field name
SP	Key word for the selector pen indicator
BD	a selector pen indicator or a control level indicator L0 to L9 or up to three input indicators

Example

```
            - 1 2 SA;                      * 1 to 2 alpha field SA
            - 3 9 2 WERT;                  * 3 to 9 WERT with 2 decimals
            - P 10 11 0 LNR;               * 10 - 11 LNR  packed
            - 1 3 0 DIV # # 99;            * Switch 99, if DIV = 0
            - 511 520 KNR  SP 01;          * Switch 01, if KNR was choosen
            -  11 13  WGRP  L1;            * Group change L1 at WGRP
```

Syntax Rules 7304

PROCEDURE DIVISION

```
---------------------------------------------------------------------
        (ON) OP (SV) (BD)
---------------------------------------------------------------------
```

ON	Condition query
OP	Processing of the operation
SV	Service query
BD	Set conditions

The condition query (ON).

```
----------------------------------------------------------------------
         KW (NOT) BD (AND NOT) BD (AND NOT) BD
----------------------------------------------------------------------
```

KW	Key word 'ON'
BD	Condition switch (01 to 99, Px, Tx, Ax, Cx, DE..)
AND	Dummy word and -connection
NOT	Reversal (condition not fulfilled)

```
----------------------------------------------------------------------
         OC (OE) (DY) (F1) (DY) (NOT) (OK) (F2) (DY) (EG)
short:   OC (F1) (F2) (EG)
----------------------------------------------------------------------
```

OP	Operation key word (see chapter 4)
OE	widened OP. code (UNTIL, WHILE ...)
F1	Faktor 1 (field name)
F2	Faktor 2 (field name)
EG	Result (field name)
DY	Dummy words (FROM, TO, WITH, TIMES ...)
OK	OPERATORS (GT, LT, EQ, >, <, =)

Example:

```
     - EXCPT                              OP
     - DO 10 TIMES                        OP F2 DY
     - DO FROM X TO Y                     OP DY F1 DY F2
     - MOVE TEXT TO LINE                  OP F1 DY F2
     - X = 0                              EG OP F2
     - X = A + B                          EG OP F1 OK F2
     - READ OTTO                          OP F2
     - MAP ARTIKEL                        OP F2
     - START                              F1
     - START TAG                          F1 OP
```

OUTPUT DIVISION

Record description

```
----------------------------------------------------------------------
         KW DN (KW) (ON) (LB)
----------------------------------------------------------------------
```

KW	1. Key word 'FILE' or 'FIELD'
DN	Filename (max 8 places) or field name
KW	Key words according to the unit
	Screen (ERASE, UNP, BEEP, MOD, LOCK)
	Disk (ADD, DEL, ALG)
	Printer (SPACE X X, SKIP)
ON	Condition query (like procedure division)
LB	Label

Field description

```
----------------------------------------------------------------------
         (ON) FN PS (LT) (AT) (EC) (CR) (CL) (EH)
----------------------------------------------------------------------
```

ON	Condition query (see procedure division)
FN	Field name
PS	Position in the record (num. value max. 4 places)
	For the screen LLCC (LL = Line, CC = column)

LT	Literal (text literal or pattern)
AT	Screen attribute
EC	Edit code for numerical values
CR	CURSOR or delete after the output (BLANK)
CL	Color (W, R, B, G, Y, P, T)
EH	EH value (BLINK, REVERSE, UNDERSCORE)

Key words 7400

Key words in the Procedure Division 7410

The following key words are reserved for the operations in the procedure division. They must not be used as names of variables or as names of labels:

```
AFOOT
BEGDT  BEGSR
CAB    CALL   CHAIN  CHECK  CHANG
       CLEAR  CLOSE  COMP   COMRG  CONT   CONVT  COPY
DEBUG  DELC   DELET  DEQ    DLI    DO     DSPLY  DUMP
EDIT   ELIM   ELSE   END    ENDDO  ENDDT  ENDEV  ENDIF
       ENDSR  ENQ    EREAD  EXCPT  EXHM   EXITD  EXITI  EXITP
       EXITS  EXITT  EXPR   EXSR
FILL   FIND
GETHS  GOTO
IF
JLB    JRB    JRC    JRZ
LIST   LOADT  LOKUP
MACRO  MAP    MAPD   MAPI   MAPO   MAPP   MOVE   MOVEA
       MOVEL  MOVEN  MOVEV  MULT   MVR
OPEN
PARM   PROT   PURGE
QSSA
READ   READB  READI  READP  REPLC  RNDOM  ROLL   ROLLB
SAVET  SCAN   SDUMP  SELCT  SETLL
       SORTA  SORTA  SQRT   SYNCP
TAG    TESTB  TESTF  TIME   TWALD
       TWASV
UCTRN  UPDAT  USSA
WAIT   WHEN   WRITE
XFOOT

ACCEPT       AVERAGE
CHANGE       COMPARE      COM-REG      CONVERT
DEQUEUE      DELETE       DISPLAY
ELIMINATE    END-EVALUATE ENQUEUE      EVALUATE     EXECUTE
             EXHM-VAR     EXIT-SEND    EXIT-TRANS   EXITP-VAR
             EXITT-VAR    EXPR-VAR
GET-UPDATE   GETCHANNEL   GO
IF-DAT       IF-DATE      IF-DATI
LEFT-SHIFT   LIST-VAR     LOADT-VAR    LOOK-UP
MAP-VAR      MAPD-VAR     MAPI-VAR     MAPO-VAR     MAPP-VAR
MOVER        MOVE-ARRAY   MOVE-LEFT    MOVE-REST    MOVE-RIGHT
PARAMETER    PERFORM      PROGRAM      PROGRAM     PROTECTION
RANDOM       READ-BACK    READ-PAGE    READB-PAGE   RECEIVE
             REPLACE      RIGHT        RIGHT-CHAR   RIGHT-ZERO
```

```
               ROLL-BACK
SAVET-VAR      SCREENDUMP   SELECT       SEND         SET
               SET-LIMIT    SQARE-ROOT
               SYNCPOINT
TDUMP          TEST-FIELD   TEST-ZONE    TWA-LOAD     TWA-SAVE
               TWALD-VAR    TWASV-VAR
UPDATE
```

Accessories 7500

Change CPG2 syntax into RPG like syntax 7520

The following program offers the possibility to change programs coded in CPG2 syntax into RPG like source codes.

```
- OPTIONS PHASE xxxxxx
-         BATCH
-         TITEL free#form#punching
-         END
- FILE IJSYS03 INPUT  FIX  1600 80  DISK
- FILE OKARTE  OUTPUT VAR       80  PUNCHER
- -I; FILE IJSYS03
-        1 80 SATZ
-        6  7 CHECK
- -C
-     DO UNTIL CPGFRC >< '  ';    * up to End of File
-        READ IJSYS03
-        IF CPGFRC = '  ' AND
-        IF CHECK >< '-*'
-           EXCPT
-        ENDIF
-     ENDDO
- -O;
-     FILE OKARTE
-        SATZ   80
```

Processing mode:

The program indicated above has to be coded and compiled. Afterwards, the ASSGN- and EXEC LNKEDT instructions in the JCL cards of the CPG2 source code will be replaced by the instruction

// EXEC xxxxxx,SIZE=AUTO   (Phase of the program above)

After the compiling of this program, the CPG2 program is disposed in the Punch Queue in RPG format.

The compilation of this Batch program is only possible with CPG3.

Creation of a Data View 7530

In relational data bases, a data view is installed online and rests in the main storage for the whole time of its processing.

To simulate this processing form, a view is created per program, per CPG3..query or per QTS and stored on a file. The view will be loaded out of this file into the main storage only if it has been called in a program with the instruction FIND. Then the loaded view rests in the main storage up to the Shut Down of the TP Monitor. A view can also be processed in a Batch program, if the available GETVIS area in the partition is sufficent for the processing.

Processing mode for the CPG2 User 7531

In this case, the view will be installed in two steps: First, the structure of the table is described with a CPG program and its elements are filled with values, then the view will be documented with the service program CPGZCTB and stored in the file CPGWKV.

1. Step: Filling of the view with a CPG-program

The following CPG-program shows the filling of a view examplary. Simplistically is assumed, that the whole table will be entered via the screen (via the Map INPUT).

```
- FILE CPGWKV UPD VAR  4020 20  KSDS
- -D
-      SATZ  0 * 22
-      KDNR       5 0
-      FIRMA     13
-      PLZ        6
-      LNR        5 0
-      KEY       20
- -C
-      DO LOOP
-         MAPD EINGABE
-         LNR = LNR + 1
-         EDIT  KEY
-         KEY CHAIN CPGWKV CHECK
-         IF CPGFRC = '  '
-            EXCPT AENDERN
-         ELSE
-            EXCPT NEU-ANLEGEN
-         ENDIF
-      ENDDO
- -O
-     FILE CPGWKV  ADD  NEU-ANLEGEN
-        KEY  20
-        RECORD 42
-     FILE CPGWKV       AENDERN
-        SATZ 42
-     FIELD KEY
-              2 '22'
-        LNR  20  PACKED
```

The installed program serves as input for the service program described in step 2. Therefore the file access and the key positions must correspond absolutely with the example.

The example has been chosen, to describe the main cards clearly. Normally a Batch programm has to be written, that groupes the data of the view out of one or several files.

2.Step: Document and store the View

The phase CPGZCTB is disposed to document and store the view. A job has to be consctructed like follows:

```
// JOB JCPGZCTB
// PAUSE         close file CPGWKV
// EXEC CPGZCTB,SIZE=AUTO
SYSIPT input data
/*
// PAUSE         open file CPGWKV
/&
```

The SYSIPT input data has following composition:

Places	1 - 4	Name of the table, with which it is called in the instruction FIND.
Places	5 - 7	Length of the table elements
Place	8	L for 'delete a table' (optional)
Places	10 - 12	Persons short signs of the specialist (optional)
Places	13 - 40	Documentation, part 1 (optional)
Places	41 - 80	Documentation, part 2 (optional)

Processing mode for CPG3 Users 7532

The handling of data views will be simplified for the CPG3 users. The CPG3..query can be used for example to create a view. The service program QTS (Quick Table Service) enables online to delete a view out of the file or in the main storage, to load a new version from the file and the indication of all tables.

This service programs are described in the CPG3 manual.

CPG Compilation without IJSYS04 7540

With the following JCL Job an example will be given, how to make CPG compilations without an ASSGN SYSIN instruction. The IJSYS04 file is not necessary for this execution.

This type of the CPG compilation offers advantages in running time, if 3380 disks are installed, or if the compilation areas are managed with VSAM Space Manager.

```
     * $$ JOB CPGUMW
     * $$ PUN DISP=I
     // JOB CPGUMW
     // EXEC CPGZPUN,SIZE=AUTO
            * $$ JOB CPGASS
            // JOB ASSEMBLY
     /*
     // EXEC CPG2
         - OPTIONS NOSYSIN;
          :
          :    CPG PROGRAM
          :
     /*
     // EXEC CPGZPUN,SIZE=AUTO
           // EXEC LNKEDT
           /&
           * $$ EOJ
     /&
     * $$ EOJ
```

Notes for the JCL cards:

The Power Punch instruction with the parameter DISP=I stores the punched cards into the Power Reader Queue.

NOS must be entered for No SYSIN as options parameter. The phase CPGZPUN is part of the CPG installation and delivered with the tape.

CPG for ESA Command Level Programs without CRL 7550

Such programs, that do not use the central routine library, and work with EXEC CICS commands, are identified with the options parameter CICSESA or the combination CICS ESA.

The compilation of such a program will be processed in three steps. The particular input will be processed via the Punch Queue with DISP=I.

STEP 1 :	CPG compilation with	EXEC CPG / CPG2 / HL1
STEP 2 :	Command Level Prepocessor	EXEC DFHEAP1$
STEP 3 :	Assembly with Link	EXEC ASSEMBLY

```
     * $$ JOB JNM=CPGUMW
     * $$ PUN DISP=I
     // JOB CPGUMW
     // EXEC CPGZPUN,SIZE=AUTO
        * $$ JOB JNM=CPGPREP
        * $$ PUN DISP=I
        // JOB CPGPREP              JOB DFHEAP1$
        // EXEC CPGZPUN,SIZE=AUTO
           * $$ JOB JNM=CPGASM
           // JOB CPGASM            JOB ASSEMBLY
     /*
     // EXEC CPG
          - OPTIONS  NOSYSIN  CICS ESA.
          :
```

```
/*
// EXEC CPGZPUN,SIZE=AUTO
   // EXEC CPGZPUN
      /*
      // EXEC LNKEDT
      /&
      * $$ EOJ
   /&
   * $$ EOJ
/&
* $$ EOJ
```

Notes for the JCL cards:

The Power Punch instruction with the parameter DISP=I stores the punched cards into the Power Reader Queue.

NOS must be entered for No SYSIN as options parameter. The phase CPGZPUN is contained in the delivery area of the CPG.

Accessories 7570

SQL/DS 7570

Up from the service level CPG3, the language SQL is supported directly in CPG programs. So SQL statements can be integrated in the CPG code as it is instructed by the producer. For the characterisation, SQL statements begin with the key word SQL.

An example for the SQL processing in CPG3 with Job Control Statements:

```
// JOB CPGSQL
// LIBDEF PROC,SEARCH=(PRD2.DB2510)
// EXEC PROC=ARISLIBP   *-- SQL/DS PRODUCTION LIBRARY ID PROC
// ON $ABEND GOTO REASS
// EXEC CPGPREP,PARM='USERID=SQLDBA/SQLDBAPW'
    - OPTIONS ROOT PHASE TST026  TITEL SQL-TESTPROGRAM;
    - -D;
    - SQL  BEGIN DECLARE SECTION
    -         USER           8
    -         PASSW          8
    -         KDNR           5 0
    -         FIRMA         30
    - SQL  END DECLARE SECTION
    - -C;
    -      USER = 'SQLDBA  '
    -      PASSW = 'SQLDBAPW'
    - SQL  CONNECT :USER IDENTIFIED BY :PASSW
    - SQL  DECLARE C2 CURSOR FOR                               *
    - SQL     SELECT KKDNR,KFIRMA                              *
    - SQL     FROM KANDENA                                     *
    - SQL     WHERE KKDNR  < 3000
              - SQL  OPEN C2
              -      DO UNTIL CPGMPF = 'P3'
    -        SQL FETCH C2                                      *
    -        SQL   INTO  :KDNR,:FIRMA
    -        MAPD BILD;            * The return code is set in the field
    -                              * SQCODE
    -      ENDDO
    -      SQL  CLOSE C2
    -      SQL  COMWITH WORK
    -      MAPO ENDE
/*
// IF $RC NE 0 THEN
// GOTO ENDE
*  STEP HL1
// LIBDEF PHASE,CATALOG=SP4U.ULIBL
// DLBL IJSYSIN,'F4.WORK.04',0,SD,,CISIZE=8192
// EXTENT SYSIPT,PRD201,1,0,46000,4000
ASSGN SYSIPT,122
// EXEC HL1
/*
// IF $RC NE 0 THEN
// GOTO REASS
*  STEP ASSEMBLER
CLOSE SYSIPT,READER
ASSGN SYSIN,122
*  STEP LNKEDT
// EXEC LNKEDT
// IF $RC EQ 0 THEN
// GOTO ENDE
/. REASS
CLOSE SYSIPT,READER
/. ENDE
```

Example 1: File Update (conversationally programmed) 8000

File Update   Customer Number   SHORT NAME   DD.MM.YY

```
  1 -  OPTIONS TITLE File#Update PHASE TEST;
  2 -  FILE CPGWRK;                  * described in Data Dictionary
  3 -  DATA DIVISION
  4 -     KEY     14
  5 -  INPUT DIVISION
  6 -     FILE CPGWRK
  7 -        15 100 SATZ
  8 -  PROCEDURE DIVISION
  9 -     DO LOOP
 10 -        MAPD EINGABE
 11 -        KEY CHAIN CPGWRK
 12            IF CPGFRC = '  ';       * record found
 13 -           MAPD ANZEIGE
 14 -           IF CPGMPF = 'P1'
 14 -              EXCPT
 14 -           END
 15 -        END
 16 -     ENDDO
 17 -  OUTPUT DIVISION
 18 -     FILE CPGWRK
 19 -        SATZ  100
```

Mapname : EINGABE

```
FIELD  A F E EC C B ATTFLD ALT.E

KEY    A        C
```

Mapname : ANZEIGE

```
FIELD  A F E EC C B ATTFLD ALT.E

SATZ   A        C
```

The statements were provided left with a current numbering. The following assertion refers in each case to this statement number.

1 Options: Title is File Update, phase name is TEST.

2 The file CPGWRK is processed. The file data are taken from the Data Dictionary. (The file can be processed for update, it has a fixed record length of 100 bytes, the key length is 14 bytes; it is a VSAM KSDS file.

3 The Data Division begins here.

4 The field KEY is defined as fourteen place alphanumeric field for the program.

5 The Input Division begins here.

6 From the file CPGWRK shall be read.

7 From the file CPGWRK the places 15 to 100 are read into the field SATZ. Usually the input structure is not described here in detail. Rather it is maintained in Data Dictionary and inserted from there during the compilation of the program. Statement 6 would then be: FILE CPGWRK DD, the field specifications like here in statement 7 could be completely dropped.

8 The Procedure Division begins here.

9 A continuous loop begins here.

10 The screen dialog. The screen EINGABE described in the QSF is put out. The program stops and expects the input of a key field KEY. With the next program function the program starts again and reads the modified fields of the screen. With the 'CLEAR' key the program can be terminated.

11 With the field KEY the file CPGWRK is accessed directly. If the key is available, then a record is read from the file according to the Input Division.

If the key is missing, the CPG internal field CPGFRC (File Return Code) is filled with the value 'NF' (not found). An input transfer does not take place in this case.

12 The File Return Code (see point 11) is queried here. The statements between IF and END are only executed if a record was found with the CHAIN.

13 Second screen dialog. Here the field SATZ is displayed unprotected, in order to be able to overwrite the file value.

14 If the program function key 1 was pressed, the program branches to the Output Division. With any other function key (including DE for a Data Entry) this statement will not be executed.

The program function key can be queried in the field CPGMPF. The keys are compressed on two places (in the example P1 for PF1).

15 END terminates the IF query. For better documentation you can also operate with the operation code ENDIF.

16 ENDDO terminates the continuing loop. The execution of the ENDDO causes a branch to the starting point of the DO-loop, thus here to the line 9.

17 The outputs division begins here.

18 With each EXCPT output a record is updated to the file CPGWRK.

19 The field SATZ is given out onto place 100 of the file CPGWRK.

Example 2: File browse with READ PAGE 8010

The file KUNDEN is a VSAM KSDS file.

File browsing program   Customer Number   SHORT NAME   dd.mm.yy

```
     1      - OPTIONS TITEL File#browsing#program ROOT PHASE TEST;
     2      - FILE KUNDEN
     3      - -D
     4      -     PAGE 20 * 79
     5      - -I
     6      -     FILE KUNDEN DD
     7      - -C
     8      -     DO LOOP
     9      -        KDNR READ-PAGE KUNDEN PAGE
    10      -        IF CPGFRC = 'EF';   * File Return Code 'End of File'
            -           FILL ' ' TO KDNR
            -        END
    11      -        RANDOM  KUNDEN
    12      -        MAPD ANZEIGE
    13      -        FILL ' ' TO PAGE
    14      -     ENDDO
    15      - -O
    16      -     FIELD PAGE
    17      -        KDNR     7
    18      -        NAME    33
    19      -        ORT     55
    20      -        UMSATZ  70 EDITCODE K
```

Mapname   : ANZEIGE

```
      FIELD   A F E EC C B ATTFLD ALT.E

      PAGE    P
21    ####    A   U    C          KDNR
```

Explanations:

1 CPG2 header specification.

2 A file with the name 'KUNDEN' is defined.

3 Beginning of the Data Division.

4 An array with the name 'PAGE' is defined. It consists of 20 fields of 79 bytes length each and shall take up a screen page with 20 lines. (Line length 79).

5 Beginning of the Input Division.

6 From the file KUNDEN a record is to be read; the structure of the data record is taken from the Data Dictionary. In the example the input field specification for the fields KDNR, NAME, ORT and UMSATZ are insertd into the program during the compilation.

7 Procedure Division.

8 Beginning of a continuous loop.

9 From the file 'KUNDEN' as many records will be read, as the array entered in the result field contains elements. The first read record is that, whose key is same or higher than the content of the field 'KDNR'. The page is stored in the array 'PAGE'. The number of lines results from the number of fields defined under NR.3. For the editing of the lines see No. 15 to 19.

10 With end of file the field CPGFRC will be filled witf 'EF'. It must be queried immediately after the READ operation.

Here the key for the next read access is filled with blank, to position at the first record of the file in case of 'End of File'.

11 The file KUNDEN is released with RANDOM. This is an instruction, which is not necessary for the function of the READ-PAGE. Rather it is considered that display programs usually can be executed by several users at the same time. Therefore with RANDOM before the following screen dialog the reading operation is terminated, to release the VSAM-strings.

12 The edited screen page will be sent to the screen in the map ANZEIGE.

13 The instruction FILL is executed here on an array. If the array is not indicated, FILL is executed for all elements in the same way. So PAGE is initialized with blanks.

14 ENDDO branches back to the appropriate DO. For the execution of the program it means that the file is displayed page-wise. In the next loop run the array PAGE is filled again with 20 read accesses to the file KUNDEN. Reading begins with the last read KDNR or with blank, if in the last run End of File was achieved.

15 The Output Division begins here.

16 Field edit. The editing of the field PAGE is initiated by the instruction READ-PAGE. The field KDNR is output in position 1 to 7 of all fields of the array, the last byte of NAME is on position 33, the last byte of ORT on position 55 in each element.

20 The numeric field UMSATZ is with its last byte on position 70 in each element of the page. UMSATZ is edited with an edit code K (keyword EDITcode): Leading zeros are suppressed; decimal places are indicated by a comma; if the field is negative, a minus sign is attached to the field; Thousand position indicators are inserted into the field.

21 In the map, a blank literal with 5 places can be given out unprotected. In the description one enters the field name KDNR for the 'Additional Input field'.

Thus one can input a key in each screen, with which the next page-wise display begins.

Example 3: File modification program with UPDATE, WRITE   8015

STANDARD TEXT   Customer Number   SHORT NAME   DD.MM.YY

```
     1     - OPTIONS TIT STANDARDTEXT PHA TEST;

     2     - FILE KUNDEN

     3     - -D
     4     -   SATZ  0 * 140
     5     -   D1          2
     6     -   KDNR        7
     7     -   D2         11
     8     -   NAME       24
     9     -   D3         28
    10     -   ORT        20
    11     -   D4         43
    12     -   UMSATZ      9 2
    13     - -I
    14     -   FILE  KUNDEN
           -      1 140 SATZ
    15     - -C
    16     -   DO LOOP
    17     -      MAPD NUMBER
    18     -      KDNR CHAIN KUNDEN UPD
    19     -      MAPD DATEN
    20     -      IF CPGFRC = '  '
           -         KDNR UPDAT KUNDEN SATZ
    21     -      ELSE
           -         KDNR WRITE KUNDEN SATZ
    22     -      END
    23     -   END
```

```
   Mapname    :  NUMMER
   ======================

   FIELD   A F E EC C B ATTFLD ALT.E

   KDNR    A        C


   Mapname    :  DATEN
   ======================

   FELD    A F E EC C B ATTFLD ALT.E

   KDNR    P
   NAME    A        C
   ORT     A
24 UMSATZ  A   J
```

Explanations:

Example 3 shows an update program for a customer master file with the CPG operations UPDAT and WRITE, that can reduce the programming and memory expenditure depending upon the application (relative to the file modification with EXCPT, compare the following example).

1 CPG control specification

2 The data of the file KUNDEN are taken from the Data Dictionary.

3 The following fields (in statement 5 to 12) are combined to a field with the name SATZ. The field is 140 places long and alphanumeric. For the field no storage space is reserved in the TWA (entry 0 *), but it is redefined from left to the right by the fields defined directly afterwards. The programmer must ensure that the sum of the individual field lengths corresponds with the field length of the overlaid field. For numeric fields the length of the packed field in bytes has to be calculated, not the number of digits.

14 From the file KUNDEN a record shall be read in from place 1 to 140. Via the definition of the overlay in 3,4 the fields KDNR, NAME, ORT and UMSATZ are filled at the same time.

16 Calculations. The starting point of a continuous loop is set. Such a continuous loop causes that the statement sequence will be executed again and again; it can be left only with the 'CLEAR' key. (In this sample program).

17 The first instruction in the loop outputs a QSF map with the name NUMBER, in order to read data from the screen.

18 A disc record is read from the file KUNDEN with the key KDNR. The disc record shall be locked against further accesses (service function UPD) until the update. If the record is missing, the file Returncode CPGFRC is filled with NF for 'not found'.

19 The QSF map DATEN is executed and waits for an input.

20 If the record was found, the field SATZ shall be written back into the file KUNDEN in the record with the key KDNR. Output Divisions are not necessary thereby, however the field SATZ must be described in the Input Divisions under the file KUNDEN.

21 Like 20, however, if the record was not found with the CHAIN, a record will be added to the file KUNDEN (CPGFRC is then NF).

24 The field UMSATZ will be edited on the screen with Edit Code J. That means: Leading zeros will be suppressed, the thousand and million places are separated by a comma, the decimal places by a point; if the amount is negative, behind the field a '-' (minus sign) will be displayed.

Example 4 : File modification program with EXCPT 8017

In todays programs the application developer can renounce on the EXCPT instruction completely, if he works with HL1 Datasets. For specialists EXCPT remains indispensable, however it should be renounced to operate with switches or to shift the logic out of the Procedure Division into the Output Division.

Under this aspect the first two programs shown below are outdated.

STANDARD TEXT Customer Number SHORT NAME DD.MM.YY

```
      1     - OPTIONS TITEL STANDARDTEXT ROOT PHASE TEST;

      2     - FILE KUNDEN

      3     - -I;  FILE KUNDEN DD

      4     - -C;  DO LOOP
      5     -         MAPD NUMMER
      6     -         KNR CHAIN KUNDEN UPD 20
      7     -         MAPD DATEN
      8     -         EXCPT
      9     -      ENDDO

     10     - -O;  FILE KUNDEN DD     ON NOT 20 AND NOT P1
     11     -      FILE KUNDEN DD ADD ON 20 AND NOT P1
     12     -      FILE KUNDEN    DEL ON NOT 20 AND  P1
```

```
Mapname    :  NUMMER
======================

FIELD  A F E EC C B ATTFLD ALT.E

KDNR   A        C


Mapname    :  DATEN
======================

FELD   A F E EC C B ATTFLD ALT.E

KDNR   P
NAME   A        C
ORT    A
UMSATZ A   J
```

Explanations:

6 In the case 'Not Found' CHAIN sets the switch 20.

8 The instruction EXCPT branches to the Output Division. Thereby the description of the output is transfered into the Output Division.

Basically all specifications of the Output Division are executed, if they are not locked by condition switches or names.

By specification of up to three switches or a name behind the EXCPT a preselection of the Output Division can be made in the Procedure Division.

The following program modification leads to the same result (exept):

```
          :
  7       -         MAPD DATEN
  8A      -         ON P1     NOT 20  EXCPT  LOESCH
  8B      -         ON NOT P1 NOT 20  EXCPT  AENDER
  8C      -         ON NOT P1     20  EXCPT  HINZU

 10       - -O;  FILE KUNDEN DD      AENDER
 11       -      FILE KUNDEN DD ADD  HINZU
 12       -      FILE KUNDEN    DEL  LOESCH
```

10 On the file KUNDEN a record shall be modified. The names of the fields which are to be modified are inserted from the Data Dictionary (DD).

11 Like 10, however a new record is to be added to the file. In addition the keyword ADD is necessary, the sequence must be kept with the key words.

12 Like 10 and 11. For the deletion of a data record the key word DEL is necessary. The following further modification leads likewise to the same result:

```
      :
 6    -         KNR CHAIN KUNDEN UPD
 7    -         MAPD DATEN
 8A   -         IF CPGFRC = 'NF'
 8B   -            ON NOT PF1  EXCPT  HINZU
 8C   -         ELSE
 8D   -            IF CONDITION PF1
 8E   -               EXCPT  LOESCH
 8F   -            ELSE
 8G   -               EXCPT  AENDER
 8H   -            ENDIF
 8I   -         ENDIF
      - -O;  FILE KUNDEN DD      AENDER
      -      FILE KUNDEN DD ADD  HINZU
      -      FILE KUNDEN    DEL  LOESCH
```

6 One can omit the switch (20). The information 'found' or 'not found' is deliverd by the internal field CPGFRC.

8 'NF' stands for 'record not found with CHAIN' (not found).

Example 5: RRDS file with numeric key field 8020

RRDS files are processed similarly to KSDS files.

Reading is both possible sequentially and in the direct access mode. Modifications are supported in the form Update, adding and deletion of a record, that means it can also be determined with the CHAIN whether a record is available or not.

The individual functions are described in the following on the base of program fragments:

```
- OPTIONS  TITEL RRDS#EXAMPLE   PHASE TEST;

- FILE RRDS
- -D;  RRN 9 0;                  * relative record number
- -I;  FILE RRDS; 1 100 SATZ
```

Sequential reading:

```
- -C;  DO UNTIL CPGFRC = 'EF'
-         1 READ RRDS
-      ENDDO
-      RANDOM RRDS
```

Add records:

```
- -C; DO LOOP
-         RRN = RRN + 1
-         RRN CHAIN RRDS
-         IF CPGFRC = 'NF';        * Not found
-            EXCPT NEU
-         ENDIF
-     ENDDO
```

Direct access:

```
- -C; RRN = 200
-     RRN CHAIN RRDS
```

Update:

```
- -C; RRN  CHAIN  RRDS  UPD
-     IF CPGFRC = '  '
-        EXCPT UPDATE
```

Deletion of a record:

```
- -C; RRN  CHAIN  RRDS
-     DELET RRDS
```

Appertaining Output Division:

```
- -O; FILE RRDS ADD NEU;  SATZ 100
-     FILE RRDS   UPDATE; SATZ 100
```

Example 6: ESDS file 8025

```
- OPTIONS TIT ESDS#-#SAMPLE#PROGRAM PHASE TEST;

- FILE ESDS

- -D
-     ARBA 4
-     KEY  4
-     RBAN 9 0

- -I
-     FILE ESDS
-        1  100   SATZ
```

Add a record. After adding the current RBA is made available (relative byte address) in the field CPGKxx (xx = current No. of the file in the Files Division):

```
- -C. EXCPT NEU;                * add a record
-     MOVE  CPGK01 TO ARBA;     * save actual RBA
```

Direct access. If the key field is alphanumeric, then it must be filled with a numeric field binary (EDIT). With numeric key the editing is made internally. With the numeric field RBAN a record could be accessed directly in the example.

```
- -C; EDIT KEY
-     KEY CHAIN ESDS
```

Update:

```
- -C; EDIT KEY;
-     KEY CHAIN ESDS
-     IF CPGFRC = '  ';         * record found
-        EXCPT ALT
-     END
```

Sequential read:

```
- -C; FILL X'00' KEY;           * first RBA, alternative :  RBAN = 0
-     DO UNTIL CPGFRC = 'EF'
-        KEY READ ESDS
-     END
-     RANDOM ESDS
```

Appertaining Output Division:

```
- -O; FILE ESDS  ADD  NEU
-        SATZ  100
-     FILE ESDS       ALT
-        SATZ  100
-     FIELD KEY
```

```
    -        RBAN    4   BINAER
10a - **     LRBA  4;            * record before
10b - **    4 '00000064' HEX;     * 2. record (with record length 100 Byte)
10c - **    4 '00001000' HEX;     * 41. record with 4K CI-Size
```

Example 7 : Add in a ESDS file 8030

The possibility is given to page through the ten records added last.

```
 1  - OPTIONS ROOT PHASE TEST;
  2  - FILE CPGESD
     - -D
  3  -    FG  10 * 4
  4  -    I  3 0
  5  -    KEY  4
     - -I
  6  -    FILE CPGESD
  7  -       6 10 SATZ
     - -C
  8  -   DO LOOP
  9  -     MAPD MAP1
 10  -     IF CPGMPF='DE';                   * key 'Data Entry'
 11  -        EXCPT;                         * output on file
 12  -        CPGK01 CHAIN CPGESD  CHECK 99; * look below !
 13  -        ROLL-BACK FG;
 14  -        MOVE  CPGK01 TO FG(1)
 15  -        I = 0
 16  -     ELSE
 17  -        IF CPGMPF = 'P1';           * key PF1
 17  -           I = I + 1;              * for paging backwards
 17  -        ELSE
 18  -           IF CPGMPF = 'P2';        * key PF2
 18  -              I = I – 1;           * for paging forwards
 18  -           END
     -        END
 19  -        IF I > 0
 20  -           IF I <= 10
 21  -              KEY = FG(I)
 22  -              IF KEY >< '    '
 23  -                 KEY CHAIN CPGESD 99
 24  -              ENDIF
 25  -           ENDIF
 26  -        ENDIF
 27  -     ENDIF
 28  -   ENDDO
 29  - -O
 29  -   FILE CPGESD ADD
 30         SATZ  10
```

Explanation of the program section 'NEW RECORD', statements 10-15 and 29,30:

The record, which was read in from the map, is added at the end of the ESDS file.

The CHAIN in statement 12 is executed, so that the record is actually written on the disk; without the CHAIN the new record would be added at this time only in the storage. With the service function CHECK in the CHAIN instruction one achieves, that no data are read in.

The array FG stores the RBAs of the last ten added records. With ROLL-BACK the RBAs already saved, are shifted to the rear (statement 12) of the array. The current RBA is stored in the first element of the RBA array.

The indicator 99 in the two CHAIN statements is necessary, because the compiler requires either the specification of an indicator or the query of the Return Code in the internal field CPGFRC.

Example 8 a: Printing in the Line mode 8035

(outdated, today it is solved with CPG4 program externally)

```
- OPTIONS PHASE TEST;

- FILES BILD L86C;

- FORMS L86C LENGTH 72  ZEILE 10 KANAL 1  Z 35 K 2  Z 70 K 12

- -C; EXCPT

- -O; FILE BILD; 550 'BEISPIEL DRUCKER'; 650 'L I N E M O D E '
-     FILE L86C SPACE 3 5  SKIP 01;
-                                 30 'FEED BEFORE THE PRINT    '
-                                 60 'AFTER CHANNEL 01 LINE 10'
-                                 90 '3 LINES BEFORE THE PRINT'
-                                120 '5 LINES AFTER THE PRINT.'
-     FILE L86C SPACE 2 1  SKIP 02;
-                                 30 'FEED BEFORE THE PRINT    '
-                                 60 'AFTER CHANNEL 02 LINE 35'
-                                 90 '2 LINES BEFORE THE PRINT'
-                                120 '1 LINE AFTER THE PRINT. '
-  FILE L86C SPACE # 2;
-                                120 '2 LINES AFTER THE PRINT '
-  FILE L86C SPACE # 1 SKIP 12;
-                                 30 'FEED BEFORE THE PRINT    '
-                                 60 'AFTER CHANNEL 12 LINE 70'
-                                120 '1 LINE AFTER THE PRINT. '
```

The LINE mode printer in the program is called L86C. In the forms specification the form length is indicated as 72 lines. Further channels are declared: the lines 10, 35 and 70 as channels 1, 2 and 12.

The rest of the program explains itself by the text constants in the Output Division.

It is recommendable to code the print output program-externally similar to the external processing of the screen in-/-output with QSF. Therefore the Lattwein product QTF (Quick Text Facility) is available. (in CPG4).

With the application of QTF, the entries in the Output Division are omitted completely for the print output. The list format only has to be entered in the text processing facility QTF instead.

Example 8 b: Printing in the Buffermode

1. With application of QSF:

Pictures, which are described already for the screen in/output in the QSF, can also be output on the printer without additional expenditure.In addition the instruction MAPP is available, which outputs a map on a freely selectable online printer.

2. Without application of QSF:

The print out is programmed like a screen output. The file description must have an appropriate entry in this case. The output is made by the operation EXCPT, which branches to the Output Division; there the output is to be programmed like a screen output.

Note for the programming in the Buffermode that the print out is optimized. For the 'print out' of a blank line, at least one blank must be output in this lint.

This rule is always to note, independent whether QSF is used or not.

Example 9: Field editing with EDIT 8040

```
OPTIONS ROOT PHASE TEST.

FILE KUNDEN

DATA DIVISION;
   BZ  20 * 78;               * screen line
   I  3 0;                    * index
   LAND  3;                   * country
   ORT  23;                   * town
   PLZ  5;                    * postcode
   STADT  35;                 * address   (last line)

INPUT DIVISION
   FILE KUNDEN DD

PROCEDURE DIVISION;
   :
   DO UNTIL EC ><  '  '
     KEY READ KUNDEN
     IF CPGFRC = 'EF'
        EC = 'EF';                   * termination criterium of the loop
     ELSE
        IF LAND = '   '
           EDIT STADT TYPE INLAND;    *  <==  EDIT with type
        ELSE
           EDIT STADT TYPE AUSLAND;   *  <==  EDIT witf type
        ENDIF
        IF WERT < MIN
           I = I + 1
           EDIT BZ(I);                *  <==  indicated EDIT
           IF I >= 20
              EC = '20';             * termination criterium of the loop
           ENDIF
        ENDIF
     ENDIF
   ENDDO
   MAPO  MASKE3
   :

OUTPUT DIVISION
   FIELD BZ;                         * edit array elements
      FIRMA 30
      STADT 66
      WERT  78 EDITCODE K
   FIELD STADT  TYPE  INLAND
      PLZ    5
      ORT   29
   FIELD STADT  TYPE  AUSLAND
      LAND   3
             5 '-';                  * separate sign country/postcode
      PLZ   11
      ORT   35
```

The example shows the editing of fields via the Output Division. In the Output Division with the key word FIELD and the field name the preparation regulation is depositted.

Also with arrays processed indicatedly (like here with BZ) only the array name is specified.

If a field shall be edited in different ways, then one operates accordingly in the Procedure Division and in the Output Division with EDIT types (demonstrated in the example for the field STADT).

The field STADT contains the last line of the address, differently edited for inland and foreign addresses, for example

```
52477 Düren                 for inland, but
  A - 1050  Wien             for foreign country
```

Example 10: READ-BACK 8045

READ-BACK KDNR SHORT NAME DD.MM.YY

```
      1  - OPTIONS ROOT PHASE TEST
         -        TITEL READ-BACK
         -        END
      2  - FILE KUNDEN
         - -D
      3  -    PAGE  20 * 70
         - -I
      4  -    FILE KUNDEN DD
         - -C
      5  -    FILL '9' TO KDNR
      6  -    DO LOOP
      7  -      KDNR READB-PAGE KUNDEN PAGE
      8  -      MAPD BILD
      9  -    END
     10  - -O
     11  -    FIELD PAGE
     12  -       KDNR  7
     13  -       NAME  33
     14  -       ORT   55
```

```
Mapname    :  BILD
======================

FELD   A F E EC C B ATTFLD ALT.E

PAGE   P
```

Explanations:

3 An array with the name PAGE is defined. The array consists of 20 fields with a length of 70 bytes each. It is to take a screen page with 20 lines.

5 The field KDNR is filled with the value '9999999'. It is assumed that on the file KUNDEN a record with the key '9999999' exists. Prerequisite for the instruction READ-BACK is, that the key field contains an existing key value.

6 From the file 'KUNDEN' as many records are read, as the array entered in the result field contains. The page elements will be stored in the array PAGE.

The file KUNDEN will be processed backwards by the instruction READ-BACK. The first read record is that, whose key is equal to the contents of the field KDNR.

Subsequently, the PAGE in the QSF map BILD is output; after the display the PAGE is filled again with data from the file KUNDEN, these again displayed etc..

11 Each line of the screen page is edited as described in line 7. The editing is initiated here by the operation READ-BACK.

12 The field KDNR is output in position 1 to 7 of all elements of the array.
13 Similarly the fields NAME and ORT are output on the positions 33 and 35
14 of the array elements with their last byte.

Example 11: Update VSAM variable record length 8050

1. Possibility: The length of the output record will be fixed by the highest output position in the Output Division.

```
  1  - OPTIONS  PHASE TEST;
  2  - FILE DATEI
  3  - -I; FILE DATEI; 1  10 KEY

  4  - -C; :
 10  -     KEY CHAIN DATEI
 11  -     ON PF1 EXCPT LG50
 12  -     ON PF2 EXCPT LG100

 50  - -O; FILE DATEI ALG  LG50;   50 'RECORD LENGTH 50'
 51  -     FILE DATEI ALG  LG100; 100 'RECORD LENGTH 100'
```

Explanations:

10 With the instruction CHAIN the file DATEI is accessed.

11 Controlled by program function keys, which are read in from the screen,
12 records with variable record lengths shall be updated on the file DATEI. With PF1 the output record should be 50, with PF2 100 bytes long.

50 This type of processing is only possible, if the key word ALG is indi-
51 cated in the file specification of the Output Division.

Additionally the file description in the File Division (or a program external description) must contain the entry for variable record length.

2. Possibility: The length of the output record will be fixed by the contents of the internal field CPGVRL.

```
  4  - -C; :
 10  -     KEY CHAIN DATEI
 11  -     ON PF1  CPGVRL = 50
 12  -     ON PF2  CPGVRL = 100

 50  - -O; FILE DATEI ALG VARIABEL
 51  -        CPGVRL  50  EDIT  Z
 52  -                46 'Satzlaenge'
```

Explanation:

50 For this type of processing the keyword VAR must be coded additionally to ALG in the record specification.

Example 12: Cursor Stop (example is outdated!) 8051

```
1   - OPTIONS TIT CURSOR-STOP PHA TEST;
2   - FILE BILD
3   - -I;  FILE BILD;  114 118 EIN1;  214 218 EIN2;  314 318 EIN3

4   - -C;  DO LOOP; EREAD BILD; END

5   - -O;  FILE BILD; 110 '1. EINGABE';  EIN1 118  ATTR A  CURSOR
6   -                 210 '2. EINGABE';  EIN2 218  ATTR A
7   -                 310 '3. EINGABE';  EIN3 318  ATTR A
8   -                 319  ATTRIBUT 0
```

Explanations:

8 A place behind the last input the screen attribute '0' (zero) is output.

The cursor branches one place to the right after the third input and the keybord is blocked for further inputs.

With the jump key the cursor can be set into an input field, in order to correct an incorrect input.

By the cursor stop it is prevented that by erroneous pressing of a key the first screen fields are overwritten.

Example 13: Temporary Storage Queuing 8060

```
 1  - OPTIONS PHA TEST;
 2  - FILE STOR UPD QUEUE FIX 80 INDEPENDENT STORAGE
 3  - -D
 4  -      PAGE 16 * 78
 5  -      I  3 0
 6  - -I
 7  -      FILE STOR
 8  -         1 78 ZEILE
 9  - -C
10  -      DO 16 TIMES WITH I
11  -        I  READ  STOR
12  -        IF CPGFRC = 'EF';
13  -          BREAK
14  -        ELSE
15  -          PAGE(I) = ZEILE
16  -        END
17  -      END
18  -      MAPD  BILD
19  -      PURGE STOR
20  -      DO 16 TIMES WITH I
21  -        ZEILE = PAGE(I)
22  -        EXCPT
```

```
    23  -      ENDDO
    24  -      EXITI 'TEST'
    25  - -O
    26  -      FILE STOR ADD
    27  -        ZEILE  78


  Mapname    :  BILD
  ======================

  FIELD  A F E EC C B ATTFLD ALT.E

  PAGE   P
```

Explanations:

The example shows the processing of a Temporary Storage Queue.

2 File description for Temporary Storage Queuing. The keyword UPD means, that the Storage area is used both for in- and for output. The entry QUEUE is necessary for Temporary Storage Queuing. The record length is fixed (FIX) and 80 places large. INDependent means, that the area is screen independent, thus for all screens available. The entry STORAGE defines the file as TS-area.

All entries should be input however not in the program, but in the Data Dictionary file.

10 With the DO operation a loop is begun, with which an index I is increased in each case by 1. The loop is passed through 16 times.

With the index I the Temporary Storage Queue STOR is read; the access takes place here (differently than with other file accesses) directly on the record with the serial number I.

In order to read with the next READ the following record, the index must be increased in each case by 1 (to which the DO loop is used here). However, it would be possible to execute the following READs without declaration of a key field - thereby also without increase of the index field the next record would be read in each case.

12 In CPGFRC 'EF' is stored, if the end of the Storage area is achieved. 'EF' is also set, if with the READ operation no Storage is found.

15 The DO-LOOP is passed through up to its end only if End of File is not set. Otherwise the BREAK terminates the DO-loop.

19 With the operation PURGE the Storage area STOR is deleted.

22 Output of a record into the Storage area (see also 23).

23 With the operation EXITI the same program (TEST) is started again. This way can be taken in place of a DO loop, which covers the whole program, in order to achieve, that before the renewed execution all fields of the program are initialized on zero or blank.

26 The entry ADD means, that records are added at the rear end of the area.

Example 14: Variable Cursor position, e.g. with error message 8065

```
:

- PROCEDURE DIVISION;
-  IF ERROR = '**'
-     CPGMCU = 'KDNR  '
-  END
-  MAPD MASKE
:
```

Explanations:

The example shows the principle of the variable positioning of the cursor in a map designed with QSF.

For cursor positioning the CPG internal field CPGMCU is available. It is 6 bytes large and alphanumerical.

With the assignnment operation the name of the field is transferred into the field CPGMCU,into which the cursor is set for the next output. The field name is transferred thereby as literal in inverted commas.

If the field CPGMCU is not filled, then the cursor is positioned, where it was indicated in the description of the map.

Pay attention, that the field CPGMCU is deleted after each screen output; if in a program several screen outputs are available, then the programmer must guarantee, that at present of a MAP-output-operation the field CPGMCU is filled 'correctly'.

If the field, into which the cursor is to be set, is an array, then the index of the desired element can be assigned to the field CPGMCI (three places numeric, no decimal places).

Examples for the operation FIND: 8070

The application of the operation FIND presupposes that a table was created, which during the first execution of the operation FIND is loaded into the main storage.

In the following examples for the FIND operation it is presupposed, that such a table is generated; it has the following structure:

Column	1 -	6	order number
Column	7 -	12	order date
Column	13 -	19	customer number
Column	20 -	29	article number
Column	30 -	31	representative number
Column	32 -	34	category of commodities

The table size is thus 34, the 'key length' 10, because each element can be key and the article number with 10 bytes is the longest possible key.

Example 15: RNDOM during the table processing

```
             :
   20        - AUFTNR FIND VIEW
             - IF CONDITION NOT EOF
   21        -    EXSR UP01
             - ENDIF
   22        - RANDOM VIEW
   23        - ARTNR  FIND VIEW
             :
```

20 The table is looked up for an order number. If this is not found, then the switch EF is set. The following FIND starts the next look up of the view at its first element. If the order number is however found in the table, then with the following FIND the look up will be restarted place of the table.

22 In this case a RANDOM VIEW is necessary, in order to set the internal pointer on to the first element of the View.

Example 16: Display of a table for selective criterion 8071

Exactly the orders are to be displayed, which are assigned to a certain representative.

```
    1        - FILE VIEW  INP  FIX  34 10 TABLE
    2        - -D
    3        -      PAGE  20 * 78
    4        -      I  3 0
    5        - -I;
    6        -      FILE  VIEW
    7        -         1  6   AUFTNR
    8        -         7 12 0 DATUM
    9        -        13 19   KDNR
   10        -        20 29   ARTNR
   11        -        30 31   VNR
   12        -        32 34   WGRP
   13        - -C;  DO LOOP
   14        -         MAPD EINLESEN
   15        -         DO 20 TIMES WITH I
   16        -            VNR FIND VIEW
   17        -            IF CPGFRC = 'EF'
   18        -               BREAK
   19        -            ELSE
   20        -               EDIT PAGE(I)
   21        -            ENDIF
   22        -         ENDDO
   23        -         MAPD ANZEIGE
   24        -      END
   25        - -O
   26        -     FIELD PAGE
   27        -        AUFTNR  6
   28        -        DATUM   16  EDITCODE Y
   29        -        KDNR    30
   30        -        ARTNR   76
```

Explanations:

1 and 6 - 12. Description of the Data View. The description of the View and its structure should be made however not in the program, but in the Data Dictionary.

14 Reading of a representative number (VNR)

The table is looked up for the entered representative number.If the number is found, then the appropriate table elements are read in. With the next passing through of the loop beginning with the following table element, the rest of the table is looked up. With FIND the table can be passed through sequentially until its end.

20 The elements of the PAGE are edited only with fields, which were found in the table.

23 The filled screen page PAGE will be edited in the map ANZEIGE.

Example 17: Orders display   8072

Orders display, additional information such as plain texts of customers and articles from files attracting.

```
 1      - FILE VIEW
 2      - FILE KUNDEN
 3      - FILE ARTSTA
        - -D;
 4      -     FEHLER  26
 5      -     I  3 0
 6      -     PAGE  20 * 78
        - -I
 7      -    FILE  VIEW  DD
 8      -    FILE KUNDEN
        -       37  61  KUNDE
 9      -    FILE ARTSTA
        -       21  45  ARTIKL
        - -C
10           :
11      -    AUFTNR FIND  VIEW
12      -    IF CPGFRC = 'EF'
13      -       EXSR FEHLER
14      -    ELSE
15      -       KDNR CHAIN KUNDEN
16      -       IF CPGFRC = '  '
17      -          ARTNR CHAIN ARTSTA
18      -          IF CPGFRC = 'NF'
19      -             FEHLER = 'Article not found'
20      -          END
21      -       ELSE
22      -          FEHLER = 'Customer not found'
23      -       END
24      -    END
        :
25      -    EDIT PAGE(I)
        :

26      - FIELD PAGE
27      -    AUFTNR   6
28      -    DATUM   16 EDIT Y
29      -    ARTIKL  44
30      -    KUNDE   72
```

Explanations:

1
7 Description of the data view and its structure is taken from the Data Dictionary (values as in the example before)

11 The table is sequenced according to order numbers. With the instruction FIND it is possible to access an order in the table directly. If the FIND instruction is situated in a loop, then (from a specified order on) all table elements are read in.

15 With the customer number found in the table one accesses the customer file, to read in the name of the customer as plain text. A prerequisite is however, that the table element KDNR corresponds to the key field of the file KUNDEN; this is to be considered when generating the data view.

17 With the article number found in the table one accesses on the article file, in order to read in the article description as plain text. A prerequisite is however, that the table element ARTNR corresponds to the key field of the file ARTSTA; this is to be considered when generating the data view.

26 The page will be edited both with values from the table and with additional informations, which are supplied by file accesses.

Example 18: Variable map name 8075

For making the MAP instruction flexible the variable map name is available. In this context not only the name of the map can be handled variable, but also the deletion of the screen before the output, the Write Control Character and the minimization of the data communication with remote screens.

The syntax and some processing examples with variable map names are described in the following:

The interface to the QSF is a 16-places alpha field; in the first eight places it contains the map name, in the ninth the information about the deletion of the screen; the tenth place contains the WCC and in the eleventh the transfer of the constants can be suppressed; the remaining five places are at present still reserve bytes. For example such a field can be described in the Data Division as overlay.

```
- -D.
     -   MAPNAM  8;   * mask name
     -   ERASE  1;    * delete screen before the mask output ?
     -   WCC  1;      * Write Control Character
     -   OCTL 1;      * output only variable fields ?
     -   FLDCLR 1;    * delete map fields or variable attributes in
     -                * the programn ? (= no map-Input/-Output)
     -   ORG MAPNAM;  * redefine of the memory from MAPNAM
     -     MAPCTL 16; * 16-places map Control-field for variable
     -                * map processing
     -   DRUC  4
```

Example:

The map BUCH01 is to be output. Before the output the screen is to be deleted, with the output the horn shall beep. Subsequently, (controled by the key PF11) this map shall be output on the printer DR01:

```
- -C; MAP   = 'BUCH01  '
-     ERASE = 'Y';         * Erase in any case
-     WCC   = 'H';         * Output with horn
-     DRUC  = 'DR01'       * give printer name
-     MAPO-VAR MAPCTL
-     ON PF11   DRUC  MAPP-VAR  MAPCTL
```

The variable map name is supported for all MAP operations.

With the use of the variable map name the programmer is responsible for the fact, that the 16-places field is filled with meaningful values. A map name must always be indicated; the remaining information bytes are optional to fill; if they remain empty, then the information indicated in the QSF is taken, otherwise it is overwritten.

Example 19: TWA-SAVE and TWA-LOAD 8080

The two operations TWASV and TWALD facilitate the buffering of data. A typical application: In transaction oriented programs it is possible with the help of these operations to transfer data to a following transaction, which is not buffered on the screen.

```
:
- -C;
-       TWA-LOAD BSP1
-       MAP BEISPIEL
:
:   *  Processing of the data read in.  Not all data, needed for the
:   *  further processing, is contained in the map Beispiel.
:
-       MAPO BEISPIEL
-       TWA-SAVE BSP1
-       EXITT  'BSP1'
```

The transaction in this example calls itself again and again. At the beginning of the program the entire TWA of the previous transaction is loaded. Thus the program also knows the data, which were not transferred over the screen. Prerequisite is, that the TWA was saved before leaving the the program with TWA-SAVE under the same name (here: BSP1).

TWA-SAVE and TWA-LOAD can be used only in such programs,which call themselves.

Example 20: Reading of files with record length greater 8K 8090

Files with record lengths greater 8 K can be read from variable input positions in connection with the CPG internal field CPGFIS.

The variable processing of the input positions is achieved in the record determination of the Input Division by the parameter VAR, the shifting factor is maintained in the Procedure Division in the internal field CPGFIS (numeric with 5 digits).

```
-  OPTIONS PHASE TEST;
-  FILE BIGFILE INP FIX  9999 10  KSDS
-  FILE BIGFIL2 INP FIX  9999 10  KSDS

- -I.
-     FILE BIGFILE;
-          15   24  F0
-        1001 1010  F1000
-     FILE BIGFILE  VAR
-           1   35  F10000
- -C;
:
-  CPGFIS = 10000
-  ' ' READ BIGFILE
:
```

It is assumed here that the file BIGFILE is 12,000 bytes large.

The fields F0 and F1000 are read in conventional way. For reading the informations from place 10001 to 100035 the shifting factor is needed; in the program it is set by CPGFIS = 10000. The shifting of the Input Divisions is only possible, because the keyword VAR was attached to the record determination of the Input Division.

Example 21a: Program for the file maintenance, conversational (outdated with switches and GOTO branches) 8101

```
-OPTIONS TITEL file#maintenance#dialog PHA TEST;

-FILE CPGWRK

-D;  PAGE 20 * 78;  I  3 0;

-I;  FILE CPGWRK DD

-C;
 A100
     MAPD MASKE1
     ON P1  GOTO A300
 A200;                         * show file sequentially
     DO 20 TIMES WITH I
       KEY READ CPGWRK
       ON EF  GOTO A250
       EDIT PAGE(I)
     ENDDO
     KEY READ CPGWRK
 A250;
     RANDOM CPGWRK
     MAPD MASKE2
     IF CON P2; GOTO A100
     ELSE;      GOTO A200
     ENDIF
 A300;                         * Modify/add/delete data records

     KEY CHAIN CPGWRK  50
     MAPD MASKE3
     EXCPT
     GOTO A100

-O;  FILE CPGWRK DD       ON DE AND NOT 50
     FILE CPGWRK DD  ADD  ON DE AND     50
     FILE CPGWRK     DEL  ON P1 AND NOT 50

     FIELD PAGE; KEY   14
                 SATZ1 78
```

The program starts with a screen dialog with map MASKE1. The key of a file can be input. With the function key PF1 one branches out to the program section 'file to maintain', which begins with the label A300; otherwise the file will be read sequentially and 20 records are displayed page wise.

Consider (statement 15 and 17):

Before the map dialog with map MASKE2 (display of a page with 20 data records) the access to the file should be released, more exactly the VSAM-strings. Therefore a further record is read and a RANDOM for the file CPGWRK is given.

More details of such programming techniques are presented in our programmers trainings.

Example 21b: Program for the file maintenance, pseudo Conv. 8110

Following program is identical to the user to the program in section 21a. The advantage is, that in the time, in which the user processes a screen, no task is active. After a MAPO the task will be terminated in each case. Only when pressing a program function key, the subsequent task is started. That is provided by the instruction EXITT (see statement 57).

```
  1  - OPTIONS TITEL File#maintenance#task ROOT PHASE TEST;

  2  - FILE CPGWRK
  3  - FILE TSQ;                     * Storage

     - -D;
  4  -    I  3 0
  5  -    PAGE  20 * 78

     - -I;
  6  -    FILE CPGWRK DD
  7  -    FILE TSQ DD;               * Place 1:        KZ
                                     * Places 2 - 15:  KEY
     - -C.
  8  -    IF CPGMPF = 'CL';          * Deletion key = Program end
  9  -      PURGE TSQ;               * Delete Storage
 10  -      MAPO ENDE;
 11  -    ELSE;                      * Execute program
 12  -      1 READ TSQ;              * Read Storage
 13  -      EVALUATE;                * One of the following alternatives
     -        *----------------------*
 14  -        WHEN CPGMPF = 'P2' OR;  * if PF2 was pressed or
 15  -        WHEN KZ = ' ';         * if first call
 16  -          KZ = '1';
 17  -          MAPO MASKE1;         * Key-Input map
     -        *----------------------*
 18  -        WHEN CPGMPF = 'DE' AND; * if DE was pressed
 19  -        WHEN KZ = '1';         * and not first call
 20  -          MAP MASKE1;
 21  -          DO WHILE I < 20 AND;  * Page is not yet full and
 22  -             WHILE CPGFRC = ' ';  * End of File was not reached
 23  -               KEY READ CPGWRK
 24  -               IF CPGFRC = 'EF'
 25  -                 KZ = ' '
 26  -               ELSE
 27  -                 I = I + 1
 28  -                 EDIT PAGE(I)
 29  -               ENDIF
 30  -          ENDDO
 31  -          IF CPGFRC = '  '
 32                KEY READ CPGWRK;   * Start-KEY for the following task
 33  -          ENDIF
```

```
34  -          MAPO MASKE2;          * Sequential notification
    -        *----------------------*

    -                                * Modify/add/delete data records
    -                                *
35  -        WHEN KZ = '1';          * Step 1:    Indicate and modify
    -                                *            on the screen
36  -          MAP MASKE1;           *   KEY read from the screen
37  -          KEY CHAIN CPGWRK;     *   Read record from the file
38  -          MAPO MASKE3;          *   Indicate record (and modify)
39  -          KZ = '2'
    -        *----------------------*
40  -        WHEN KZ = '2';          * Step 2:    Modify on file
41  -          KEY CHAIN CPGWRK;     *   Read record
42  -          MAP  MASKE3;          *   Modified record from the screen
43  -          IF CPGMPF = 'P1';     * PF1 was pressed for delete
44  -            IF CPGFRC = '  ';   * Record found
45  -              EXCPT WRK-DELETE
46  -            END
47  -          ELSE
48  -            IF CPGFRC = '  ';   * Record found
49  -              EXCPT WRK-MODIFY
50  -            ELSE;               * Record was not found
51  -              EXCPT WRK-ADD
52  -            ENDIF
53  -          ENDIF
54  -          MAPO MASKE1
55  -          MOVE '1' KZ;          * Next Step: Indicate
    -        *----------------------*
56  -      END-EVALUATE
57  -      EXCPT TS.                 * Describe Storage
58  -      EXITT 'TEST';             * Task calls itsself
59  -    ENDIF;                      * Query CLEAR-Key

    - -O;
60  -      FILE CPGWRK DD       WRK-MODIFY
61  -      FILE CPGWRK DD  ADD  WRK-ADD
62  -      FILE CPGWRK     DEL  WRK-DELETE
63  -      FILE TSQ DD          TS
64  -      FIELD PAGE
65  -        KEY 14
66  -        SATZ1 78
```

Explanations:

3 The data are stored in the Temporary Storage area TSQ. Here only the fields KZ and KEY are transferred to the subsequent task. 15 is indicated therefore in the Data Dictionary as record length of TSQ. If the application is still to be developed, one designate a (generous) reserve during the first description, so that the area can be increased during the development without problems. A Q for Queueing is standard today.

8 The programmer is responsible for the termination of the program with transaction oriented programming. Task oriented programs should therefore always begin with the query of the program function key for program termination (or other termination criterias).

9 With program end the used buffers usually are deleted. The instruction PURGE deletes the entire queue.

At the end of the processing is a map, in which (at least) the TransId of the subsequent program can be entered.

12 Usually, the storage is read first. Here it contains the sign KZ, which states, in which processing step the program is. Additionally it contains the KEY processed last.

13 EVALUATE ensures that only exactly one of the following alternatives is executed. The individual alternatives are described with WHEN (see 14, 18, 34, 39), END EVALUATE (see 55) terminates the instruction.

Example 22a: Convert fields with CONVERT 8120

In the following a program extraction is represented, which describes the different possibilities of the operation CONVERT exemplarily:

```
      Statement                              Field contents F1     F2
    -----------------------------------     -------------- ----   --------

 1  - MOVE 'Test' TO F1                                     Test

 2  - CONVERT F1  LOW                                       test

 3  - CONVERT F1                                            TEST

 4  - CONVERT F1 INTO F2 HEX                                TEST   E3C5E2E3

 5  - FILL '*' F1                                           ****   E3C5E2E3

 6  - CONVERT F2 INTO F1 CHARACTER                          TEST   E3C5E2E3

 7  - CONVERT F2 INTO F2 CHAR                               TEST   TESTE2E3

 8  - MOVE 'F0F0' F1                                        F0F0   TESTE2E3

 9  - CONVERT F1 INTO F2 CHAR                               F0F0   00STE2E3
```

Concerning 2: Basically applies: Characters, which cannot be converted in the sense of the operation, remain unmodified.

In the example: All bytes of the field shall be modified in lowercase letters. The only uppercase letter is 'T', all other characters remain unchanged; if there are digits in the field, they also rest unchanged.

During the conversion from Hex- into character format, the programmer must guarantee however, that the origin field contains valid values,   . i.e. values between '00' and 'FF'. Other values are converted in this case to X'00'.

To 7 and 9: Basically applies: The result field is not deleted before the operation.

The result of the conversion will be left justified in the result field.

Example 22b: Converting of time informations with CONVERT 8125

CONVERT for date fields, examples:

```
          Statement                          ALFA6   ALFA8     ALFA10
        --------------------------------     ------  --------  ----------

   1    - MOVE '220193' ALFA6                220193

   2    - CONVERT ALFA6 DATE                 930122

   3    - CONVERT ALFA6 INTO ALFA8 DATUM       "     220193

   4    - MOVE '2001' ALFA8                    "     22012001

   5    - CONVERT ALFA8 DATE                   "     20010122

   6    - ALPHA8 = '23.02.93'                  "     23.02.93

   7    - CONVERT ALFA8 DATE                   "     93.02.23

   8    - CONVERT ALFA8 INTO ALFA10 DAT        "        "      23.02.93

   9    - MOVE '1999' TO ALFA10                "        "      23.02.1999

  10    - CONVERT ALFA10  DATE                 "        "      1999.02.23

  11    - CONVERT ISO8 INTO ALFA10  UDATE-FORMAT
```

Concerning 5: Here we see, that in some cases the data of the origin field can not be interpreted clearly. For these cases the service function YEAR is available, which assumes with the converting that the year is located in front (left) in the data field.

Additionally the converting of date is supported for six to eight digit numeric fields. An exception of the rule that alphanumeric fields are always converted into alphanumeric and numeric fields into numeric, is the CONVERT with the service UDAte:

11 - CONVERT ISO8 INTO ALFA10 UDATE-FORMAT

Here the eight-digit numeric field ISO8 (19960109) is stored convertedly into an alpha field and edited correspondly to the UDATE-format (09.01.1996).

CONVERT for time fields:

```
          Statement                          NUM5      NUM5C     NUM70
        --------------------------------     -----     -----     -------

  11    - NUM5 = 10000                       10000

  12    - CONVERT NUM5 INTO NUM5C  SECONDS   10000     03600

  13    - NUM5 = 90000 + NUM5C               93600       "

  14    - CONVERT NUM5 INTO NUM70  SECS        "         "       0034560

  15    - CONVERT NUM70 INTO NUM5C TIME        "       93600
```

Concerning 12:	The converting of 10000 (HMMSS, thus 1:00:00, that means 1 hour) in seconds results in 3.600.
Concerning 14:	The converting of 9 o'clock and 36 minutes results in 34.560 seconds.
Concerning 15:	The converting of 34.560 seconds results in 9 hours an 36 minutes.

Example 23: Storing in the Common Area (CPGCOM) 8130

CPGCOM offers the possibility of storing data in the Common Area. This form of the storing is used for communication with other programming languages and tools.

CPGCOM does not need to be defined by the programmer as a field. Max. 4080 bytes data can be transferred. CPGCOM is processed with the instruction EDIT and SELECT.

```
- OPTIONS PHASE PROGCPG;
- -I;
-   FIELD CPGCOM;
- *    1  8  DATUM;             * Eight digit date
-      9 18  WTAG;              * Day of the week text in clear
-     19 20  KW;                * Calendar week
- -C;
-   DATUM = CPGDAT;             * Fill parameter
-   EDIT CPGCOM;                * Supply Common Area
-   EXPR  DRCOBOL;              * LINK in the COBOL-Date routine
-   SELECT CPGCOM;              * Read results from CPGCOM
-   IF WTAG = 'SATURDAY'  OR
-   IF WTAG = 'SUNDAY'
-   :

- -O;
-   FIELD CPGCOM
-      DATUM   8
```

A CPG program PROGCPG uses here an existing date routine. The routine expects the input parameter in the places 1 to 8 of the Common Area, the result is transferred in the places 9 to 18 or in the places 19 and 20 of the Common Area.

Field edits can be differentiated also according to types. If for example a CPG program communicates with several Cobol programs, then the field CPGCOM can contain different informations. With the keyword TYPE direct editings can be addressed.

Example:

```
- -C;  EDIT CPGCOM TYPE PROG1

- -O;  FIELD CPGCOM TYPE PROG1
-         ALFA20  20
-      FIELD CPGCOM TYPE PROG2
-         DATUMV   4 PACKED
-         TEXT    16
-         DATUMB  20 PACKED
```

The length of the Common AREA can be set to a desired value, if for example the called program requires this.

In addition a numeric field in front of the EXPR instruction is to be filled accordingly.

```
-   CAL  =  750;                * Common Area length: 750 Bytes
-   EDIT CPGCOM;                * Editing of the Common Area
-   CAL EXPR PROGRM3;           * Call of the sub program
```

Example 24: Sequential processing of record types (segments)   8140

Both VSAM files and HL1 structures can have different structures in their records.

In the following example you can see, how such a file can be processed during the input.

```
OPTIONS PHASE TST000  TITEL Segment-Verarbeitung;
*
FILE AUFTRAG
*
INPUT DIVISION
   FILE AUFTRAG
       1  2  SA
       SEGMENT  KOPF  DD  TYPE 01  REFERENCE  AUFTRAG
       SEGMENT POSTEN DD  TYPE 02  REFERENCE  AUFTRAG
       SEGMENT  TEXT  DD  TYPE 03  REFERENCE  AUFTRAG
*
PROCEDURE DIVISION
   DO WHILE CPGFRC = '  '
      READ AUFTRAG
      IF CPGFRC >< 'EF'
        EVALUATE
          WHEN SA = '01'
            READI AUFTRAG  SEGMENT KOPF
          WHEN SA = '02'
            READI AUFTRAG  SEGMENT POSTEN
          WHEN SA = '03'
            READI AUFTRAG  SEGMENT TEXT
        END-EVALUATE
         :
         :  Verarbeitung
         :
      ENDIF
   ENDDO
```

Note:

The segments must be described in the Input Division directly following the appropriate file.

Description:

During the sequential reading the record type is always read in first. Depending on the read record type, the same data record is read in again with the operation code READI thereafter.

In the Data Dictionary the file AUFTRAG is described with the record types 01, 02 and 03. In order to be able to address these different structures directly in the program, they get segment names, in this case KOPF, POSTEN and TEXT.

By the entry REFerence, a reference is made for the segment names to the record types of the file AUFTRAG in the Data Dictionary.

In the Procedure Division the desired record type of the file is then addressed, by indicating the key word SEGMent and a segment name with the READI operation in addition to the file name.

Example 25: Direct processing of record types with segments 8145

Direct processing of segments is supported only with the instruction CHAIN in connection with the key word UPDate.

Note: CHAIN for updates locks a record or a whole data-CI up to the following update. If no update is made, then a RANDOM must be programmed.

```
OPTIONS PHASE TSTXXX  TITEL Segment Processing;
*
FILE BESTELL
*
INPUT DIVISION
   FILE BESTELL
       1  2  SA
       SEGMENT SATZ1
           11 17 NFNR
*
PROCEDURE DIVISION
   :
   KEY CHAIN BESTELL UPDATE
   IF CPGFRC = '  ';                * found !
     EVALUATE
       WHEN SA = '01';              * next free order number
         READI BESTELL  SEGMENT SATZ1
         NFNR = NFNR + 1;
         EXCPT ERSTER-SATZ
         NFNR = NFNR - 1
       WHEN SA = '02';              * 'normal data record'
         EXCPT AENDERUNG
     END-EVALUATE
   ELSE
     EXCPT NEU
   ENDIF
   :
```

Description:

In a transaction oriented program modified data are available for updating or adding for the file. In the first record of the file (for the case of the new installation) the next free number is for the key editing.

The following problem results:

For CHAIN only one input description is available. However two different functions are to be executed, depending on the accessed key.

1. Update: The CHAIN may not read file data, because these would overwrite the data to be updated.
2. Determine the record number for later adding. Here data must be read in, however only the record number.

Way out: Use of segments!

In the program above with the CHAIN basically only the record type is read in. If record type '01' is found, then the next free number is read in with READI, then counted up and updated again. For the READI a segment is always indicated. This is described in the Input Division under the file and contains the input specifications for further data of the last read record. With record type '02' a valid key is found, no data is read, but an update is made.

Example 26: Logically connected IF queries 8150

For the logical connection of IF and WHEN queries there are different syntax rules.

```
options root phase TSTxxx       * Source Code in upper/lower case letters
        titel Demo#IF#AND#/#OR
        end
file stor
-d
  feld1 8
  feld2 8
  meld 24
-i
  file stor dd
-c
  if cpgmpf = 'CL' or;          * PF-Key CLEAR
  if cpgmpf = 'PC' or;          * PF-Key PF12
  if cpgmpf = 'QC';             * PF-Key PF24
     exhm aexit;                * general EXIT-routine
  else
     1 read stor
     map BSP026
     if feld1 > '        ' and;  * Wrong input, if both fields are filled
        feld2 > '        '   or; * or if field1 is empty and field2 begins
        feld1 = '        ' and;  * with a '$'.
        feld2 = '$';
        meld = 'Falsche Eingabe !'
        cpgmcu = 'FELD1';        * Cursor positioning
     else
        exhm absp26;             * Processing
     end
     mapo BSP026
     exitt 'BS26'
  endif
```

Rules for logically connected IF or WHEN queries:

- To a group of logically connected queries belongs (only) one ENDIF, WHEN does not have an appropriate END.
- Starting from the second query the IF or WHEN can be coded (like above with the CPGMPF QUERY) or be omitted (as with the query of FELD1 and FELD2).
- Brackets are not possible. Clarity of the query is given by the rule 'AND binds more strongly than OR' (comparably with the known rule 'point calculation is higher than line calculation').

Example 27: Upper and lower case printing in pseudo conversational application 8160

The following example shows, how the translation in uppercase letters is deactivated, the original UCTRN status saved and being restored at the program end. (UCTRN stands for Upper Case translation and is a parameter of the Terminal Control Table of the CICS).

```
 1  OPTIONS ROOT PHASE TST031;
 2  FILE STOR UPD QUE FIX 1 STORAGE; * Storing the UCTRAN-status

    -I;
 3     FILE KANAL HS
 4        1  1    STATUS;
 5     FILE STOR DD
    -C;
 6     1 READ STOR;                  * UCTRN-Status at program-start
 7     IF CPGMPF = 'CL';             * Delete key
 8       EXHM PENDE KANAL T;         * UCTRN turn back
 9       PURGE STOR;                 * Delete Storage
10       MAPO ENDE
11     ELSE
12       IF CPGFRC = 'EF';           * If first program call
13         EXHM PSTART KANAL;        * UCTRN-Status determine/save
14       ENDIF
15       MAP TEST LOW;               * Service 'LOW', the MAP instruction
         :                           * does not translate in uppercase
         :                           * letters
17       MAPO TEST
18       UPDAT STOR;                 * Storing the UCTRAN status
19       EXITT 'TT31'
20     ENDIF
```

Explanations:

Concerning 8: Service function 'T' for EXHM in transaction oriented programs:

If in a dialog oriented program the CLEAR key is used, the program automatically branches to the end. In transaction oriented programs the way to the program end is determined by the programmer, the automatic program termination therefore might not be appropriated.

The operation EXHM checks during the return branch to the calling program, whether the CLEAR key is pressed and branches then to the end of the program. This automatism can be switched off in transaction oriented programs by the service function 'Task'.

Concerning 15: Not only CICS knows the translation in uppercase letters with the screen input. Also CPG translates according to the standard in uppercase letters. This translation of the CPG is switched off with the service function LOW in the MAP operation.

The following HL1 module receives the UCTRAN status of the CICS by using the operation COMRG.

```
    OPTIONS PHASE PSTART;
    -D;
            STATUS  1;                    * Data channel
                                          *-----------------------------*
            SYSINF  32;                   * COMRG-area, must be 32-places alpha
            ORG SYSINF;                   *
               DUMMY1  21;                *
               UCTALT  1;                 * Place 22, UCTRN up to CPG 2.0
               DUMMY2  4;                 *
               UCTR  1;                   * Place 27, expanded UCTRN
    information
               ORG;                       * End of the redefinition
    -C;
            COMRG SYSINF;                 * Finds out the UCTRAN-status
            UCTRN OFF;                    * Basicly: UCTRN OFF
                                          * ----------------------------*
            IF UCTR >< UCTALT AND;        * Place 27 with COMRG will be
            IF UCTR >< 'T';               * filled from CICS 2.2. Therefore
              UCTR = UCTALT               * the place 22 should be con-
            ENDIF                         * sidered.
                                          *
                                          * ----------------------------*
            STATUS = UCTR;                * Transfer the UCTRAN-status to
                                          * the channel field
```

Explanations:

Since CPG release 2.1 the UCTRN status is in two places in the COMRG area: In place 22 (U or N) and in place 27 (U, N or T). The entry T is for UCTRN Transaction. Place 27 is filled by the CPG, if a CICS release 2.2 or higher is in use.

The following HL1 module resets the UCTRN status to the status before the call of the program.

```
    OPTIONS PHASE PENDE;
    -D;
            STATUS  1;
    -C;
            IF STATUS = 'U';              * If the status was 'U', then set
              UCTRN ON;                   *   UCTRN on again
            ELSE;
              IF STATUS = 'T';            * If the status was 'T', then set
                UCTRN ON TRANSACTION;     *   UCTRN transaction on again
              ENDIF
            ENDIF
```

Example B01 : Reader, Printer and overflow control 8600

```
 1  - OPTIONS  BATCH  PHASE BSPB01;

 2  - FILE READER  INPUT  FIX  80   READER
 3  - FILE PRINTER OUTPUT FIX 132   PRINTER

 4  - FORMS PRINTER  LENGTH 72  LINE 1 CHANNEL 1  LINE 66 CHANNEL 12

    - -I
 5  -     FILE READER
 6  -        1 80 SATZ

    - -C
 7  -     EXCPT  KOPF
 8  -     DO UNTIL CPGFRC >< '  '
 9  -       READ READER
10  -       IF CPGFRC >< 'EF'
11  -         IF CONDITION OF;        * Overflow
11  -           EXCPT  SEITENWECHSEL
11  -         ENDIF
12  -         EXCPT  DATEN
    -       ENDIF
13  -     ENDDO

    - -O
15  - FILE PRINTER  SPACE # 2  SKIP 01  KOPF
16  -               24 'First Page'
17  -     UDATE     70
18  -     UTIME     80
19  - FILE PRINTER  SPACE # 2  SKIP 01  SEITENWECHSEL
20  -               24 'Following Page'
21  - FILE PRINTER  SPACE # 1           DATEN
22  -     SATZ      80
```

Explanations:

1 Option BATch must be indicated for batch programs.

2 A file 'READER' is defined as input file (INPUT). The record length is (FIX) 80 bytes. The unit is READER (SYSIPT).

3 A printer file 'PRINTER' is defined as output file (OUTPUT). The record length is (FIX) 132 bytes. The unit is 'PRINTER'. (SYSLST).

4 The forms length is defined here with 72 lines. The first line will be defined with channel 01 and line 66 with channel 12. (Channel 12 describes the line, after which the overflow switch 'OF' is set).

The FORMS entries can be completely omitted. In this case the defaults apply: Channel 01 = line 6, channel 12 = line 66.

5 In the Input Division the reading regulation for the file READER is described.

6 From the READER the places 1 to 80 are read into the field SATZ.

7 Execution of the output. All record specifications not locked with name or switches will be executed and the output, whose record specification has the name KOPF. Because all outputs are locked, this works like a direct branch to the output with the name KOPF.

8 Beginning of a loop, which is terminated, if the File Return Code is not blank any more (at End of File).

9 The input file 'READER' is read.

10 'End of File' is queried in the File Return Code (EF in CPGFRC).

11 The overflow switch is queried; if it is set, the output SEITENWECHSEL is executed.

12 Execution of the output with the name DATEN.

13 Branches back to the start of the loop (statement 8).

15 This output is addressed with EXCPT KOPF (see 7).

Before printing there is a forms feed to channel 01, that means to the beginning of the next page (SKIP 01). Before printing no line feed takesplace, after printing two line feeds are made. (SPACE # 2).

16 Output of the heading, date and time.

19 Forms feed to channel 01 before printing, after printing 2 line feeds.

20 Output of a literal.

21 A line feed after output of the field SATZ, if EXCPT DATEN is executed.

Example B02 : Program catalogue CPGWRK 8610

```
 1  - OPTIONS  BATCH  TITEL Program catalogue PHASE BSPB02;

 2  - FILE PRINTER
 3  - FILE CPGWRK  INPUT

    - -D;
 4  - KEY    14
 5  - STRICH 80

    - -I;
 7  - FILE CPGWRK;
 8  -     1   2   SA
 9  -     3  10   PNAME
10  -    11  14   TRANID
11  -    15  24   PROT
12  -    25  29 0 TWA
13  -    30  49   TEXT
14  -    50  52   PKZ

    - -C;
13  -    FILL '-' STRICH
14  -    EXCPT  01
15  -    MOVEL '01 ' TO KEY
16  -    KEY  SETLL  CPGWRK
```

```
-    DO UNTIL CPGFRC = 'EF'
-       KEY  READ  CPGWRK
-       IF CPGFRC >< 'EF'  AND
-       IF SA <= KEY
-          EXCPT 02
-       ENDIF
-    ENDDO

- -O;
- FILE PRINTER  SPACE # 2  SKIP 01  ON 01
-                                    17 'PROGRAM CATALOGUE'
-                        UDATE      70
-                        UTIME      80
- FILE PRINTER  SPACE # 2           ON 01
-       STRICH  80
- FILE PRINTER  SPACE # 2           ON 01
-               22 'PROGRAMM   TRID   TEXT'
-               67 'PKZ     TWA   PROTECTION'
- FILE PRINTER  SPACE # 1           ON 02
-       PNAME    8
-       TRANID  15
-       TEXT    38
-       PKZ     46
-       TWA     54  EDITCODE J
-       PROT    67
```

Explanations:

In this example it is positioned with a key in the file 'CPGWRK' and a list (program catalogue) is given out on the printer. In case of 'End of File' (switch EF) or in case of a new record type the program is terminated.

Example B02, coded in a move up-to-date style:

By application of the CPG4 tools the coding can be shortened substantially. The printing output is program externally developed and maintained, file in and output structures are inserted at compilation time from the Data Dictionary.

The remaining coding:

```
- OPTIONS  BATCH  TITEL Programmkatalog  PHASE BSPB02;
- FILE PRINTER
- FILE CPGWRK  INPUT
- -D;
-    KEY   14
- -I;
-    FILE CPGWRK DD
- -C;
-    LIST BSPB02 SECTION KOPF
-    KEY = '01 '
-    DO UNTIL CPGFRC = 'EF'
-       KEY READ CPGWRK
-       IF CPGFRC >< 'EF' AND SA <= KEY
-          LIST BSPB02 SECTION DATEN
-       ENDIF
-    ENDDO
```

Example B03 : Loading records from the reader in a KSDS file.   8620

```
- OPTIONS BATCH TITEL Load#CPGKSD PHASE BSPB03;

- FILES READER PRINTER
- FILE CPGKSD OUTPUT

- -D
-    STRICH  80

- -I
-    FILE READER
-       1 20 KEY
-      21 80 DATEN

- -C
-    FILL '-' STRICH
-    EXCPT KOPF
-    DO UNTIL CPGFRC = 'EF'
-       READ READER
-       IF CPGFRC >< 'EF'
-          EXCPT SAETZE
-       ENDIF
-    ENDDO

- -O
- FILE PRINTER  SPACE # 2  SKIP 01   KOPF
-             18 'CPGKSD NEW RECORDS'
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 2            KOPF
-      STRICH 80
- FILE PRINTER  SPACE # 2            KOPF
-              3 'KEY'
-             26 'DATA'
- FILE PRINTER  SPACE # 1            SAETZE
-      KEY    20
-      DATEN  81
- FILE CPGKSD   ADD                  SAETZE
-      DATEN  80
-      KEY    20
```

Explanations:

In the example above, cards are read from a reader and transferred in the KSDS file 'CPGKSD'. The read records are added sequentially with 'ADD', that means if records exist in the file, new records are inserted into the file.

FILE CPGKSD OUTPUT in the Files Division declares the file as output file. The output is sequential. With the sequential adding the records must exist in ascending record sequence. While the loading of the file a log is printed on the file 'PRINTER' at the same time.

Example B04 : Copy records from the CPGWRK into a KSDS file. 8630

```
- OPTIONS  PHASE BSPB04
-          TITEL Copy#KSDS#to#KSDS
-          BATCH
-          END

- FILES READER PRINTER
- FILE CPGWRK  INPUT
- FILE CPGKSD  OUTPUT

- -D
-    EC      2
-    STRICH 80

- -I
-    FILE READER
-       1   2 SA
-       1  14 KEYVL
-    FILE CPGWRK
-       1  14 KEY
-      15 100 SATZ

- -C
-    FILL '-' STRICH
-    DO WHILE EC >< 'EF'
-       EC = '  '
-       READ READER
-       IF CONDITION EOF
-          EC = 'EF'
-       ELSE
-          EXCPT  KOPF
-          KEYVL SETLL CPGWRK
-          DO WHILE EC = '  '
-             KEYVL READ CPGWRK
-             IF CONDITION EOF
-                EC = 'EF'
-             ELSE
-                IF KEY = SA
-                   EXCPT  DATEN
-                ELSE
-                   EC = 'SA'
-                ENDIF
-             ENDIF
-          ENDDO
-       ENDIF
-    ENDDO

- -O
- FILE PRINTER  SPACE # 2  SKIP 01   KOPF
-             12 'COPY RECORDS'
-             34 'FROM CPGWRK TO CPGKSD'
-             46 'RECORD TYPE'
-      SA     53
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 2            KOPF
-      STRICH 80
- FILE PRINTER  SPACE # 2            KOPF
-              3 'KEY'
-             25 'SATZ'
- FILE PRINTER  SPACE # 1            DATEN
-      KEY    14
-      SATZ  107
- FILE CPGKSD   ADD                  DATEN
-      KEY    14
-      SATZ  106
```

Explanations:

In the example above, records from the 'CPGWRK' are copied sequentially into a KSDS file. The records are transferred with 'ADD' into the KSDS file. From the file 'READER', header cards are read in, which indicate the record types to be copied. On the file 'PRINTER' a log is printed.

Example B05 : Direct update of records in a KSDS file. 8640

```
        - OPTIONS  BATCH  TITEL Direct#Update  PHASE BSPB05;

        - FILES READER PRINTER CPGKSD

        - -I
        -    FILE READER
        -       1  20 KEY
        -      21  21 UPDKZ
        -    FILE CPGKSD
        -      21  39 NAME

        - -C
        -     EXCPT KOPF
        -     DO UNTIL CPGFRC = 'EF'
        -        READ READER
        -        IF CPGFRC >< 'EF'
     ** -           KEY CHAIN CPGKSD  99
        -           EXCPT DATEN
        -        ENDIF
        -     ENDDO

        - -O
        - FILE PRINTER  SPACE # 2  SKIP 01          KOPF
        -             24 'Modify records in CPGKSD'
        -      UDATE  70
        -      UTIME  80
        - FILE PRINTER  SPACE # 2                   KOPF
        -              3 'KEY'
        -             25 'NAME'
        -             42 'X'
        - FILE PRINTER  SPACE # 1                   DATEN
        -      KEY    20
        -      NAME   40
        -      UPDKZ  42
        -   ON 99  #  43 '*** not found      ***'
        - FILE CPGKSD   ON NOT 99                   DATEN
        -      NAME   59
        -      UPDKZ 106
```

Explanations:

In this example records, which were read in from a card reader, are updated directly in a KSDS file. A log is printed out on the printer.

**** consider:** A switch with the CHAIN must be indicated, if the internal field CPGFRC for the file Return code is not queried in the program.

Example B06 : Sequential update of records in a KSDS file 8650

```
- OPTIONS  BATCH  TITEL sequential#Update PHASE BSPB06;

- FILE LISTE OUTPUT FIX 132 PRINTER
- FILE CPGKSD

- -I; FILE CPGKSD
-     1  20  KEY
-     1  40  SATZ

- -C;
-     EXCPT KOPF
-     MOVEL '11' TO KEY
-     DO WHILE KEY = '11'
-        KEY READ CPGKSD
-        IF KEY = '11'
-           EXCPT SEQUPD
-        ENDIF
-     ENDDO

- -O;
- FILE LISTE    SPACE # 2  SKIP 01  KOPF
-              17 'SEQUENTIAL UPDATE'
-              38 'OF RECORDS IN CPGKSD'
-      UDATE  70
-      UTIME  80

- FILE LISTE    SPACE # 1           SEQUPD
-      SATZ   40
-              40 '*'

- FILE CPGKSD                       SEQUPD
-              40 '*'
```

Explanations:

In this example records are sequentially updated in a KSDS file and given out at the same time as list on the printer.

Only the records are updated, which have the key '11'.

Example B07 : Delete records in a KSDS file sequentially 8660

```
- OPTIONS  BATCH  TITEL sequential#delete PHASE BSPB07;

- FILES READER PRINTER CPGKSD

- -D
-    EOF  2
- -I
-    FILE READER
-       1   2  SA
-    FILE CPGKSD
-       1  20  KEY
-      21 100  SATZ
- -C
-    DO WHILE EOF >< 'ER';                  * EOF READER
-       READ READER
-       IF CPGFRC = 'EF'
-          EOF = 'ER'
-       ELSE
-          EXCPT KOPF
-          DO WHILE EOF >< 'ED';            * EOF CPGKSD  or
-             SA READ CPGKSD;               * end of the record key
-             IF CPGFRC = 'EF'
-                RANDOM CPGKSD
-                EOF = 'ED'
-             ELSE
-                IF KEY = SA
-                   EXCPT DELETE
-                ELSE
-                   RANDOM CPGKSD
-                   EOF = 'ED'
-                ENDIF
-             ENDIF
-          ENDDO
-          FILL ' ' EOF
-       ENDIF
-    ENDDO
- -O
- FILE PRINTER  SPACE # 2  SKIP 01          KOPF
-             24 'Delete Records in CPGKSD'
-             36 'RECORD TYPE'
-      SA     39
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 2                   KOPF
-              3 'KEY'
-             25 'DATA'
- FILE PRINTER  SPACE # 1                   DELETE
-      KEY    20
-      SATZ  101
- FILE CPGKSD   DEL                         DELETE
```

Explanations:

In this example records are sequentially deleted in a KSDS file and a log is printed on the printer.

Example B08 : Direct delete of records in a KSDS file 8670

```
-   OPTIONS  BATCH  TITEL direct#delete PHASE BSPB08;

-   FILES READER PRINTER CPGKSD

- -D
-   EOF  2

- -I
-   FILE READER
-               1  20 KEY
-   FILE CPGKSD
-              21  39 NAME

- -C
-    EXCPT KOPF
-    DO WHILE CPGFRC = '  '
-       READ READER
-       IF CPGFRC = '  '
-          KEY CHAIN CPGKSD  99
-          EXCPT DELETE
-       ENDIF
-    ENDDO

- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             16 'Direct delete of'
-             34 'records in CPGKSD'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 2           KOPF
-              3 'KEY'
-             25 'NAME'

- FILE PRINTER  SPACE # 1           DELETE
-      KEY    20
-      NAME   40
-    ON 99  # 43 '*** not found ***'

- FILE CPGKSD DELETE      ON NOT 99 DELETE
```

Explanations:

Records in a KSDS file are deleted directly. A log is output on the printer.

Example B09 : List program catalogue backwards 8680

```
- OPTIONS  BATCH TITEL catalogue#backwards PHASE BSPB09;
- FILES PRINTER CPGWRK
- -D; OC       2
-     STRICH  80
- -I; FILE CPGWRK;    * DD - take fields from the Data Dictionary
-                     *
-                                         1   2 SA
-                                         3  10 PNAME
-                                        11  14 TRANID
-                                        15  24 PROT
-                                        25  290TWA
-                                        30  49 TEXT
-                                        50  52 PKZ
- -C; FILL '-' STRICH
-     EXCPT KOPF
-     '02' READB CPGWRK;           * The record must exist!
-     DO WHILE OC = '  '
-        '01' READ-BACK CPGWRK;    * Last 01 record is read
-         IF CPGFRC = 'EF'
-            OC = 'EF'
-         ELSE
-            IF SA = '01'
-               EXCPT  RECORD
-            ELSE
-               OC = 'ED'
-            ENDIF
-         ENDIF
-      ENDDO
- -O
- FILE PRINTER  SPACE # 1  SKIP 01  KOPF
-              17 'Program catalogue'
-              27 'backwards'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 2           KOPF
-      STRICH 80

- FILE PRINTER  SPACE # 2           KOPF
-              22 'Programm   TRID   Text'
-              67 'PKZ     TWA   Protection'

- FILE PRINTER  SPACE # 1           RECORD
-      PNAME   8
-      TRANID 15
-      TEXT   38
-      PKZ    46
-      TWA    54  EDIT J
-      PROT   67
```

Explanations:

In a program catalogue it will be read backwards (READB) and a log on the printer is output.

Example B10 : Copy records from the card reader into an ESDS file 8690

```
- OPTIONS  BATCH  TITEL Copy#Reader#->#ESDS  PHASE BSPB10;

- FILES READER CPGESD

- -I; FILE READER;  1 80 SATZ

- -C; DO LOOP
-        READ READER
-        ON EOF  BREAK
-        EXCPT
-     ENDDO

- -O; FILE CPGESD  ADD;  SATZ 80
```

Explanations:

In this example records are copied from a card reader into a ESDS file. The records are added with 'ADD' to the ESDS file.

If records already exist in the ESDS file, then the new records are added at the end of the file.

Example B11 : Printing of an ESDS file 8700

```
- OPTIONS  BATCH  TITEL ESDS#print  PHASE BSPB11;

- FILES CPGESD PRINTER

- -D; COUNT  5 0
-     KEY    4
-     OC     2

- -I; FILE CPGESD
-        1  80  SATZ

- -C; EXCPT KOPF
-     FILL  X'00'  KEY
-     DO WHILE CPGFRC = '  '
-        KEY READ CPGESD
-        IF CPGFRC = 'EF'
-           EXCPT SUM
-        ELSE
-           COUNT = COUNT + 1
-           EXCPT RECORDS
-        ENDIF
-     ENDDO
- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             19 'Records from CPGESD'
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 1           RECORDS
-      SATZ   80
- FILE PRINTER  SPACE 2 1           SUM
-             21 '*** Number of records'
-      COUNT  23 EDITCODE Z
```

Explanations:

In the example above an ESDS file is read and output on the printer.

The processing begins with the first record of the file (FILL X'00' to KEY).

Example B12 : Sequential update of an ESDS file 8710

```
- OPTIONS  BATCH  TITEL ESDS#seq#update  PHASE BSPB12;

- FILES CPGESD PRINTER

- -D; COUNT  5 0
-     KEY  4
-     OC  2

- -I; FILE CPGESD
-        1  80  SATZ

- -C; EXCPT KOPF
-     FILL  X'00'  KEY
-     DO WHILE CPGFRC = '  '
-        KEY READ CPGESD
-        IF CPGFRC = 'EF'
-           EXCPT SUM
-        ELSE
-           COUNT = COUNT + 1
-           EXCPT RECORDS
-        ENDIF
-     ENDDO

- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             17 'Records from CPGESD'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 1           RECORDS
-      SATZ   80

- FILE PRINTER  SPACE 2 1           SUM
-             17 '*** Number of records'
-      COUNT  23 EDITCODE Z

- FILE CPGESD                       RECORDS
-             40 '<--- SEQ.UPD --->'
```

Explanations:

In the example above records are updated sequentially in an ESDS file and output as list on the printer at the same time.
As unit of the file 'ESDS' must be entered in the Data Dictionary.
The processing begins with the first record of the file.

Example B13 : Direct and sequential processing of an ESDS File 8720

```
- OPTIONS  BATCH  TITEL ESDS processing  PHASE BSPB13;
- FILES CPGESD PRINTER
- -D
-    COUNT  5 0;                   * Counter
-    KEY  4;                       * Key (RBA)
-    KEYS  99 * 4;                 * Key-(RBA)-Array
-    OC  2;                        * Operation code
-    X  3 0;                       * Array index
- -I
-    FILE CPGESD
-       1  80   SATZ
- -C
-    EXCPT  KOPF
-    FILL  X'00'  KEY
-    DO UNTIL X =  99  OR
-       UNTIL CPGFRC = 'EF'
-       KEY READ CPGESD
-       IF CPGFRC >< 'EF'
-         X = X + 1
-         KEYS(X) = CPGK01
-         EXCPT RECORDS
-       ENDIF
-    ENDDO
-    EXCPT SUMME
-    RANDOM CPGESD
-    DO 5 TIMES WITH X
-       KEY = KEYS(X)
-       KEY CHAIN CPGESD   11
-       EXCPT UPDESD
-    END
- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             17 'Records from CPGESD'
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 1          RECORDS
-      SATZ   80
-             93 'sequential'
- FILE PRINTER  SPACE 2 1          SUM
-             17 '*** Number of records'
-      X      23 EDIT  Z
- FILE PRINTER  SPACE # 1          UPDESD
-      SATZ   80
-             93 'direct '
-    ON 11 # 110 'not found     '
- FILE CPGESD                      UPDESD
-             20 '<<< direct >>> '
```

Explanations:

Here an ESDS file is processed both directly ('CHAIN') and sequentially. ('READ'). By the instruction 'RANDOM' it is switched from the sequentially processing to the direct processing, at the same time a log on the file 'PRINTER' is printed. The direct access may only take place with a valid relative byte address. In this example the relative byte addresses were stored with seq. Read in the array KEYS.

Example B14 : Printing of a RRDS file 8730

```
- OPTIONS  BATCH  TITEL RRDS#print PHASE BSPB14;

- FILES CPGRRT PRINTER

- -D
-   COUNT  5 0
-   KEYN  9 0
-   OC  2

- -I
-   FILE CPGRRT
-      1  80  SATZ

- -C
-   EXCPT KOPF
-   KEYN = 1
-   DO WHILE CPGFRC = '  '
-      KEYN READ CPGRRT
-      IF CPGFRC = '  '
-         COUNT = COUNT + 1
-         EXCPT RECORDS
-      ENDIF
-   ENDDO
-   EXCPT SUM

- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             17 'Records from CPGRRT'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 1           RECORDS
-      SATZ   80

- FILE PRINTER  SPACE 2 1           SUM
-             17 '*** Number of records'
-      COUNT  23
```

Explanations:

In the example above a RRDS file is read and output on the printer.

The processing begins with the first record of the file.

Example B15 : Processing a RRDS file directly 8740

```
- OPTIONS  BATCH  TITEL RRDS#direct  PHASE BSPB15;

- FILES READER PRINTER CPGRRT

- DATA DIVISION
-    COUNT  5 0;                 * Counter
-    INFO  14;                   * Info for Duplicate Record
-    SERVICE  7;                 * Text field for service

- INPUT DIVISION
-    FILE READER
-       1   5 0  X;              * Relative record number
-       6   6    OC;             * Operation code
-       1  20    FELD
-    FILE CPGRRT
-       1  80    SATZ

- PROCEDURE DIVISION
-    EXCPT KOPF
-    DO UNTIL OC = 'E';          * Up to End of File
-       READ READER
-       IF CPGFRC = 'EF'
-          MOVE 'E' TO OC;       * Termination criterium of the loop
-       ELSE
-          X  CHAIN CPGRRT  11
-          COUNT = COUNT + 1
-          MOVEL X  TO  SATZ
-          MOVE  FELD  TO  SATZ
-          EVALUATE
-            WHEN OC = '*';      *------ Service: Modify  -----*
-              SERVICE = 'UPDATE'
-              EXCPT  RRTUPD
-            WHEN OC = '+';      *------ Service: Add  -*
-              SERVICE = 'WRITE '
-              EXCPT RRTADD
-            WHEN OC = '-';      *------ Service: Delete   ----*
-              SERVICE = 'DELETE'
-              EXCPT RRTDEL
-          END-EVALUATE
-          IF CPGFRC = 'D'
-             INFO = 'double record'
-          ENDIF
-          EXCPT  RECORDS
-       ENDIF
-    ENDDO
-    EXCPT SUM

- -O
- FILE PRINTER  SPACE # 2  SKIP 01          KOPF
-             21 'CPGRRT DIRECT CHAIN, '
-             42 'UPDAT, WRITE, DELET  '
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 1                   RECORDS
-      SATZ     80
-      ON 11 #  90 'not found'
-      SERVICE 100

-      INFO    120 BLANK-AFTER-OUTPUT
-      X       125 EDITCODE Z

- FILE PRINTER  SPACE 2 1                  SUM
-               21 '*** Number of records'
-      COUNT    23 EDIT  Z

- FILE CPGRRT             ON NOT 11        RRTUPD
-      SATZ     80

- FILE CPGRRT        ADD                   RRTADD
-      SATZ     80

- FILE CPGRRT        DEL  ON NOT 11        RRTDEL
```

Example B16 : Load an RRDS file from the card reader 8750

```
- OPTIONS  BATCH  TITEL Load#RRDS#from#card#reader  PHASE BSPB16;

- FILES READER PRINTER CPGRRT

- INPUT DIVISION

-    FILE READER
-       1  80  SATZ

- PROCEDURE DIVISION

-    EXCPT  KOPF
-    DO LOOP
-       READ READER
-       ON EOF  BREAK
-       EXCPT RECORDS
-    ENDDO
-    EXCPT ENDE

- OUTPUT DIVISION

- FILE PRINTER  SPACE # 2  SKIP 01        KOPF
-             23 'Load CPGRRT from Reader'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 1                 RECORDS
-      SATZ   80

- FILE PRINTER  SPACE 2 1                 END
-             17 '*** End  ***      '

- FILE CPGRRT   ADD                       RECORDS
-      SATZ   80
```

Explanations:

In the example above cards are read from a card reader and transferred into the RRDS file 'CPGRRT'. The read records are added with 'ADD', that means if already records are existing on the file, new records are added at the end of the file. With the loading of the file a log is printed at the same time on the file 'PRINTER'.

Example B17 : Sequential update of a RRDS file 8760

```
- OPTIONS  BATCH  TITEL sequential#UPD  PHASE BSPB17;

- FILES CPGRRT PRINTER

- -D;
-     COUNT  5 0
-     OC  2
-     RECN  3 0
- -I;
-     FILE CPGRRT
-        1 80  SATZ

- -C;
-     EXCPT KOPF
-     RECN = 1
-     DO WHILE CPGFRC = '  '
-        RECN  READ  CPGRRT
-        IF CPGFRC = '  '
-           COUNT = COUNT + 1
-           EXCPT RECORDS
-        ENDIF
-     ENDDO
-     EXCPT  SUM

- -O;
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             19 'Records from CPGRRT'
-      UDATE  70
-      UTIME  80

- FILE PRINTER  SPACE # 1           RECORDS
-      SATZ   80

- FILE PRINTER  SPACE 2 1           SUM
-             21 '*** Number of records'
-      COUNT  23 EDIT  Z

- FILE CPGRRT                       RECORDS
-             40 '<--- SEQ.UPD --->'
```

Explanations:

In the example above records are sequentially updated in a RRDS file and output as list on the printer at the same time.

The processing begins with the first record of the file.

Example B18 : File with OPEN in the program 8770

```
- OPTIONS  BATCH  TITEL Entry#No#open  PHASE BSPB18;

- FILE CPGESD  INP  VAR  8100  ESDS   NO OPEN
- FILE PRINTER

- -D
-    COUNT  5 0
-    KEY  4
-    OC  2
- -I
-    FILE CPGESD
-       1 80  SATZ

- -C
-    IF CONDITION U1;             * UPSI 1
-       OPEN CPGESD
-    ENDIF
-    IF CPGFRC =  '  ';           * File Return Code for OPEN
-       EXCPT KOPF
-       FILL X'00' KEY
-       DO WHILE CPGFRC = '  '
-          KEY READ CPGESD
-          IF CPGFRC = '  '
-             COUNT = COUNT + 1
-             EXCPT RECORDS
-          ENDIF
-       ENDDO
-       IF CONDITION U1;          * UPSI 1
-          CLOSE CPGESD
-       ENDIF
-       EXCPT  SUM
-    ELSE;                        * Error with OPEN
-       IF CPGFRC = 'EF';         * Empty file
-          EXCPT EMPTY-FILE;      * Message on the printer
-       ELSE
-          EXCPT OPEN-ERROR;      * Message on the printer
-       END
-    ENDIF
- -O
- FILE PRINTER  SPACE # 2  SKIP 01  KOPF
-             19 'Records from CPGESD'
-      UDATE  70
-      UTIME  80
- FILE PRINTER  SPACE # 2  SKIP 01  EMPTY-FILE
-             20 'File CPGESD is empty'
- FILE PRINTER  SPACE # 2  SKIP 01  OPEN-ERROR
-             24 'OPEN-error with CPGESD:'
-      CPGFRC 27
- FILE PRINTER  SPACE # 1           RECORDS
-      SATZ   80
- FILE PRINTER  SPACE 2 1           SUM
-             21 '*** Number of records'
-      COUNT  23 EDITCODE  Z
```

Explanations:

The file CPGESD is declared with NO OPEN in the Files Division.
Thus the file will not be automatically opened, but can be directly opened with the instruction OPEN. In this example the OPEN is executed only if the switch U1 (= UPSI 1) is set. If necessary, OPEN sets a switch and the File Return Code CPGFRC like all file operations, even if the file is empty. (Then the switch EF is set additionally). With the instruction CLOSE the file can be closed at the end of the processing.

Example B19 : Output on puncher 8780

```
- OPTIONS  BATCH  TITEL Output#on#Puncher  PHASE BSPB19;

- FILES READER PUNCHER

- -I
-    FILE READER
-       1  80  SATZ

- -C
-    DO LOOP
-       READ READER
-       ON EOF  BREAK
-       EXCPT
-    ENDDO

- -O
-    FILE PUNCHER
-       SATZ     80
```

Example B20 : Copy disk tape 8790

```
- OPTIONS  BATCH  TITEL Copy#Disk tape  PHASE BSPB20;

- FILE IJSYS04 INP FIX   #   80  DISK
- FILE TAPE    OUT FIX 1600  80  TAPE

- INPUT DIVISION
-  FILE IJSYS04
-     1 80 SATZ

- PROCEDURE DIVISION
-  DO LOOP
-     READ IJSYS04
-     ON EOF  BREAK
-     EXCPT
-  ENDDO

- OUTPUT DIVISION
-  FILE TAPE
-     SATZ  80
```

The '#' has to be entered for block size in the file description, if a disk file is not blocked.

Chapter Index