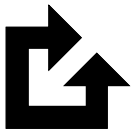
SQT1-22911-00



QPAC for Batch

Reference Manual

Version 9 Release 11

First Edition (December 2023)

This edition applies to version 9 release 11 of the Osys Software AG licensed program QPAC for Batch, program number 8050-QP-911-T10 and to all subsequent versions, releases and modifications until otherwise indicated in new editions.
Consult Osys Software AG for current information on this product.

Order publications or address your comments to the following address

Osys Software AG
Muellerenstrasse 3, CH-8604 Volketswil/Switzerland
E-Mail qpac@osys.ch

Special Notices

The terms listed below are used in this publication and are trademarks or service marks of the following companies:

Osys Software AG Zurich, Switzerland:
Osys-QPAC

International Business Machines:
z/OS
CICS
VSAM
DL/I
DB2
RACF
MQSeries
ISPF/PDF-TSO

Contents

Figures

Chapter 1. Introduction

System Overview

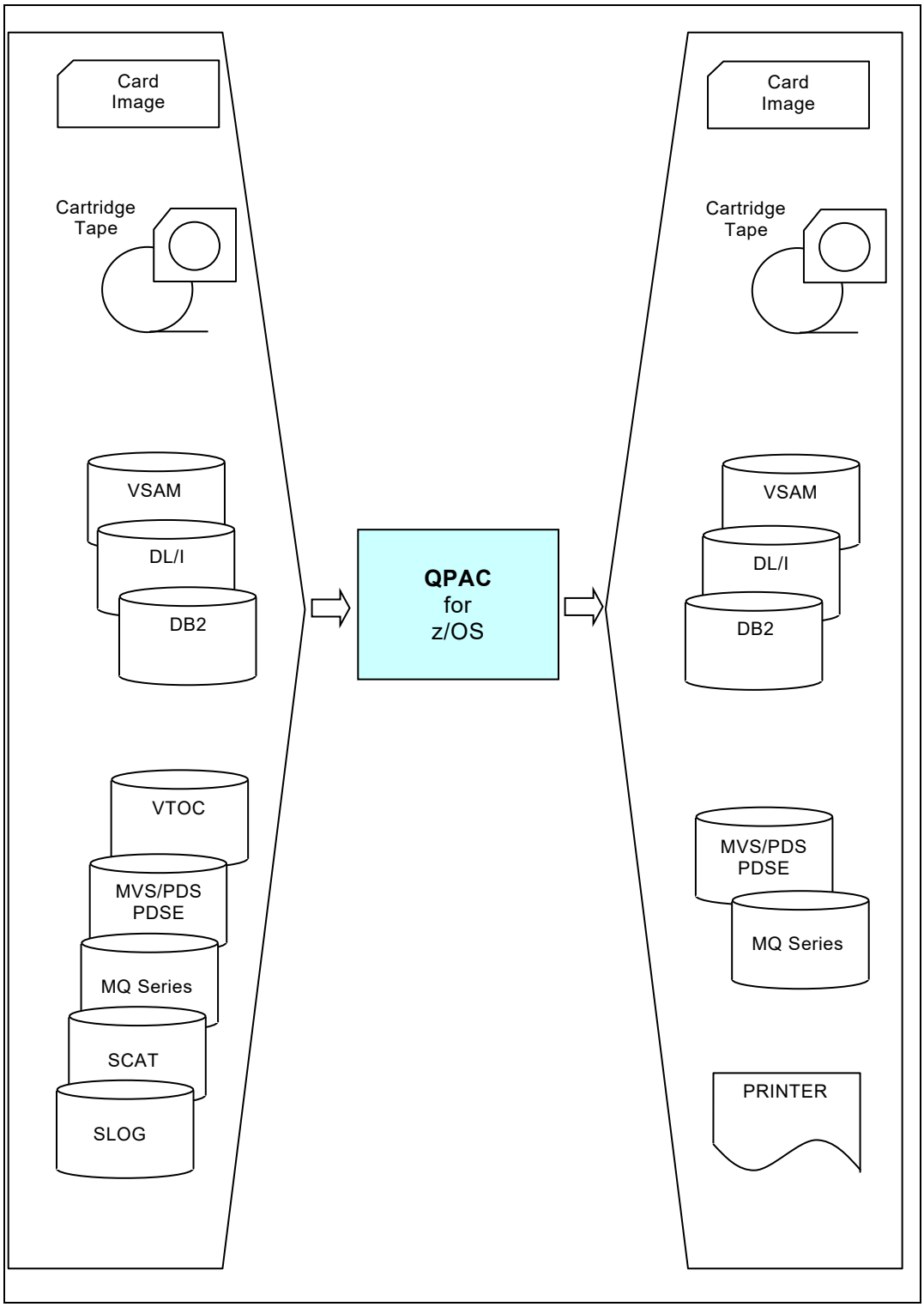


Fig. 1: QPAC-Batch system overview

Purpose of QPAC

The QPAC utility, a multi-function programming system of the 4th generation, enables the solution of problems within an environment of commonly known file organizations, in an easy, logical, and programmable form.

Especially (but not exclusively) for the following tasks QPAC is qualified:

- all kind of ad-hoc tasks
- create reports
- create, read, print files and databases
- analyze, reorganize, modify, correct files and databases
- process records individually according to logical decisions
- process a file **without** needing to take its background into account

All problems are freely programmable, following logical principles without complicated file definitions, and without special knowledge.

Possibilities

All standard IBM file organizations may be processed:

- all SAM org. on Disk/Tape input and output with fixed or variable record length, blocked or unblocked
- all VSAM org. input and output
- DL/I databases (DL/I support feature)
- DB2 Databases (DB2 Support Feature)
- z/OS PDS/PDSE system libraries
- z/OS System Catalog
- z/OS System Logger
- VTOC
- MQSeries
- ISPF/PDF-TSO
- WEB CICS support

The QPAC Program

The utility consists of several overlays.

It is stored under the name **QPAC**, and the utility is 'called' using that name.
It occupies a main storage region of at least 1MB. For each file definition, 4K plus the I/O area must be added. In the remaining main storage area, the processing instructions will be built up from translation of the QPAC statements.
The size of a program to be written in QPAC thereby depends mainly on the main storage size.
If no symbolic addressing is used the maximum addressable logical record length of records processed directly in the I/O areas is 4096 bytes. Records themselves may be larger.
The processable block length is device dependent.

After submission the QPAC source statements are read in by QPACIN,or the input medium defined by the EXEC-PARM, and then syntax checked by the QPAC translator. If no syntax errors are found, machine code is produced and immediately executed.

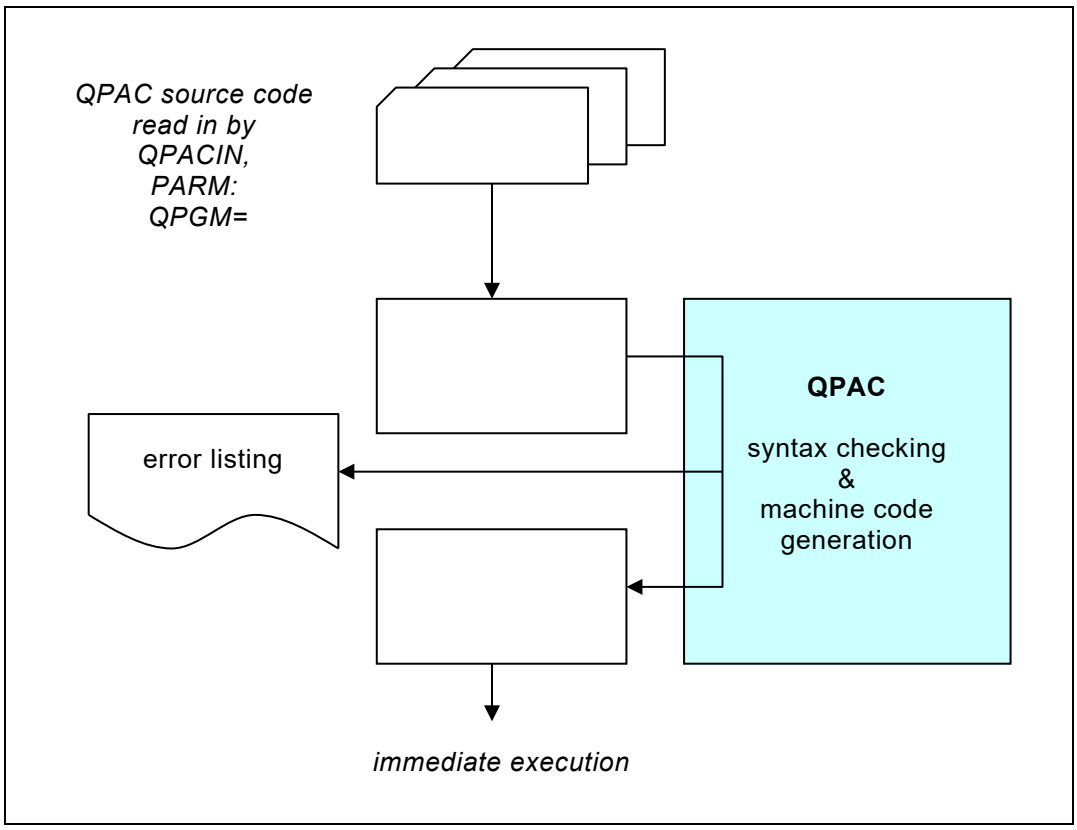


Fig. 2: Syntax checking and machine code generation

Job Control

The utility statements are read in following the DD statement QPACIN (//QPACIN DD*).

A list of statements read in, and any error messages, appear in the DD statement QPACLIST (//QPACLIST DD SYSOUT=A).

Normally the DD names of the input or output files to be processed are their file identifications (//IPF DD .., //OPF DD ..,//UPF1 ... etc.). It is possible, however, to give different names explicitly in the file definition; e.g. `IPF=*FILEIN,SQ` (//FILEIN DD .... DCB=DSORG=PS).

With `PARM` options, console log information can be suppressed or printed on QPACLIST.

```
//EXEC  PGM=QPAC,PARM='NOLOG'
```

Fig. 3: QPAC PARM Option

Any error messages will appear on the console and cannot be suppressed. The console start and end message may be suppressed by using the `NOLOGTIT` option. Please refer to "The PARM Control Statement" for further information.

```
//SAMPLE   JOB  ....
//         EXEC PGM=QPAC,PARM='NOLOG'
//QPACLIST DD   SYSOUT=*
//IPF      DD   DSN=
//OPF      DD   DSN=
//QPACIN   DD   *
IPF=SQ
OPF=SQ
...
...
...
END
```

Fig. 4: Sample JCL for z/OS

The QPAC definition statements may also be directly read in from a PDS. In this case the DD name is QPACPGM. The member name is defined by the EXEC-PARM.

```
//SAMPLE   JOB  ....
//         EXEC PGM=QPAC,PARM='NOLOG,QPGM=SAMPLE'
//QPACLIST DD   SYSOUT=*
//QPACPGM  DD   DSN=
//IPF      DD   DSN=
//OPF      DD   DSN=
```

Fig. 5: Sample JCL for z/OS with PARM QPGM=

Format of User Statements

QPAC statements are written in free format within columns 1 to 71. Column 72 must be left clear. Columns 73-80 can be used for any identification or sequence number, which will not influence processing.

Each statement must be fully contained within one record. The number of blanks between two statements is not limited. Within a statement there are logically no blanks.

A blank is signifying the end of the statement.

The number of statements is not limited. Their contents are processed in the presented procedural sequence (left - right - top - down) and converted into machine code for execution.

Comments

An asterisk (∗) within a statement record signifies that the part to the right of the ∗ is a comment.

Where it is possible to confuse the asterisk with the operations sign for multiplication (∗), an asterisk followed by a non-blank indicates a comment (e.g. ∗.).
A double asterisk (∗∗) is not allowed.

An asterisk (∗) in column 1 of a statement record signifies that the part to the right of the ∗ is always a comment.

QPAC Listing Control

With some control commands the QPAC listing can be controlled. They begin in column 1 with an asterisk:

`*SK`	skip to new page
`*SP`	space 1 line
`*SP`*n*	space *n* lines
`*POFF`	print off
`*PON`	print on

Fig. 6: Control of the QPAC listing

The PARM Control Statement

The first QPAC statement can be one that defines various options, additional to, and partially independent, of the equivalent job control definitions.

If used, it must contain the characters **PARM=** in positions 1-5.

Following the fixed character string, the options can be defined in any sequence, separated by commas.

This statement is compulsory in, for example, DB applications on z/OS systems, since EXEC parameters are not possible in that environment.

Format

```
PARM=Option,Option, ...
```

The following PARM options are possible:

CALL=SUB MAIN	COBOL or PL/I programs are internally called via LE (Language Environment). There exist two initialization functions SUB routine and MAIN program. As default SUB is assumed. The differences are described with the CALL command in more details.
COPYL | NOCOPYL	List copy-books.
DUMP	If the QPAC ends abnormally, an additional system dump is issued (needed in inexplicable situations).
EPARM='....'	System parameter communication. The character string between apostrophes is communicated as a parameter to a program called by QPAC when using CALL-'*program*',*parm* or when the EPARM reserved symbol is used and is stored in the internal work area. The reserved symbol EPARM is allocated with the length of the defined parameter. Additionally, the length of the parameter is presented in the field EPARML.
HSPACE=*nn*M	Size of the Hiper space. This area is defined in megabytes. Its maximum size is depending on the region size. For its addressing implicit position symbols HPOS*nnnn* are available. Index registers or BASED structures may also be used for addressing. Individual field symbols may be assigned to this area by inserting the letter "H“ between the position specification and the equals sign: e.g. 100H=FIELDSYMBOL,CL20.
LCT=*nn* 60 (z/OS)	Line counter (page length) - overrides temporarily the value given by the JCL.
LIST | NOLIST	Print list / no list information - overrides temporarily the value given by the JCL.
LISTL=*nnn*	Width of the line for the system printer. Printer output (e.g. from WPOS5200 in the work area),

	output by PUTLST (QPACLIST), normally has a line width of 121 characters (including control character). This PARM definition can increase the width of the line up to 250 bytes. This is meaningful when working with a laser printer.
LOG | NOLOG	Print log / no log information - overrides temporarily the value given by the JCL.
LOGTIT |NOLOGTIT	The start and the end message on the system log are suppressed.
PLIST | NOPLIST | NOPLIST=SAVE	The program statements are listed or suppressed. This way only the statistics may be listed. NOPLIST forces NOXREF. With NOPLIST=SAVE the program statements are only listed in case of an error. Because the program statements are internally stored above the 2 GB line a MEMLIMIT size of 1 MB must be defined.
QMOD=*loadmodule*	Load module name of the QPAC program code. Eliminates the need for a QPACIN statement.
QPGM=*member*	QPAC program as PDS member which may be directly read in by a //QPACPGM DD statement. The QPACPGM data sets with a record length of 80 bytes may be concatenated. The //QPACIN DD is now obsolete.
SYNTAX	Only a syntax analysis of the QPAC statements is desired. The statements are not executed.
STRUCT	Only a structure test is desired. The statements are not executed.
TRACE=ON OFF	If TRACE=ON is defined QPAC will in case of an abnormal end list the last used statement sequence.
WORK=*nnnnn* WORK=*nn*M	The size of the internal working storage area can be enlarged with this definition. The value may not be less than 12288, and the upper boundary is limited by main storage size, maximum is 16 MB. The defined size is the number of bytes and determines the address positions of the internal working storage. WORK=80000 results in address positions for the internal working storage from 1 to 80000. For its addressing implicit position symbols WPOS*nnnn* are available. Index registers or BASED structures may also be used for addressing. Individual field symbols may be assigned to this area by inserting the letter "W“ between the position specification and the equals sign: e.g. 100W=FIELDSYMBOL,CL20. Positions 1-4999 are only addressable with the letter "W“. WORK= definitions with a size up to 1 MB are loaded below the 16 MB line. Larger working storage areas are placed above the line.
XREF | NOXREF | FXREF	Cross-reference list yes, no or full (also the unused symbols are listed).

NOLSR | LSR	For VSAM file definitions the default operand should be set to LSR or NOLSR (default) respectively.
PLAN=*qpacplan*	See under Chapter 12: DB2 Feature
DB2ID=*sysid*	See under Chapter 12: DB2 Feature
GROUPID=*xx*	An identification unit of up to 2 characters in EBCDIC (i.e. also multi-punch) format can be defined, with a space or comma being treated as a delimiter.
PASSWORD=*xxxxxx*	An identification unit of up to 6 characters in EBCDIC format can be defined, with a space or comma being treated as a delimiter.
USERID=*xxxx*	An identification unit of up to 4 characters in EBCDIC format can be defined, with a space or comma being treated as a delimiter.
STAB | NOSTAB	Set abend option Abends not caused by a P-Check situation are handled and listed on QPACLIST for documentation.

PARM Option User Defaults

PARM Options that are to be used as standard can be stored as default in a load module (phase in VSE) with name QPACBOPT.

QPAC looks for and processes PARM options in the following sequence:

1. PARM options in the load module QPACBOPT if existing
2. PARM options from the EXEC parm
3. PARM options at the beginning of the program code

The load module has the following Assembler format:

```
QPACBOPT CSECT
  DC     CL80'PARM=.....'
  DC     CL80'PARM=.....'
  END
```

Fig. 7: Format PARM defaults load module QPACBOPT

COPY Statement (IBM z/OS)

In the z/OS version, the source statement library is referenced by the DD statement SYSLIB.

membername is a name of 1 to 8 characters.

All forms of definition, except for other `COPYs`, may be used within a COPY book.

```
COPY-membername
```

Fig. 8: Format (z/OS) COPY statement

Load Module Code Mode

This extension allows the user to load his QPAC definitions as a load module, e.g. static QPAC programs can be executed without using QPACIN (z/OS).

The first 2 characters of the name must contain 'QP'.

```
//EXEC PGM=QPAC,PARM='QMOD=LOADMOD'
```

Fig. 9: Execution under z/OS

The load module itself contains only QPAC definitions. It is assembled and linked as code of 80 byte constants, as for example:

```
QPACLM01 START 0
         DC    CL80'PARM=WORK=20000         '
         DC    CL80'IPF=SQ                  '
         DC    CL80'OPF=PR                  '
         DC    CL80'*. BEISPIEL             '
         DC    CL80'SET OPOS1 = IPOS1,CL120 '
         DC    CL80'END                     '
         END
/*
```

Fig. 10: QPAC load module code

It is possible to define a `PARM=` statement as the first code line.

Data Security Extensions

QPAC provides the user with an EXIT in which he can define the **authorization rules** for access to protected data sets. This USER-EXIT is a phase or load module with the name **QPACUSER**. This EXIT is taken for each disk or tape file definition (only tapes with labels) and checks the file ID. Up to 3 additional control keys can be defined using `PARM` options, allowing a 3 level security key or password scheme.

`GROUPID=`*`xx`*	2 characters
`USERID=`*`xxxx`*	4 characters
`PASSWORD=`*`xxxxxx`*	6 characters

The processing of the security key information, the coding of the necessary checks on the file identification (file name in the DLBL statement or data set name in the DD statement), is entirely up to the user; QPAC only puts this information, if available, at the disposal of the user.

The EXIT must be coded according to the official linkage conventions, i.e. according to the principle of CALL-SAVE-RETURN.

The RETURN code (R15) on entering QPAC again, must contain either 0, meaning file access is authorized, or 4 when it is not authorized.

```
QPACUSER  START 0
          USING *,15
          USING DEVDSECT,2
 *
          DC    XL256'00'           MUST BE X'00'
 *
 ENTRY1   B     GOENTRY1            POSITION X'100'
          B     *                   RESERVED
          B     *                   RESERVED
          B     *                   RESERVED
 *
 GOENTRY1 SAVE  (14,12)
          L     2,0(1)              LOAD DSECT ADDRESS
          ST    13,SAVEAREA+4
          . (USER CODE)
 *
 EXITOK1  L     13,SAVEAREA+4
          LA    15,0                RC=0
          ST    15,15(13)
          RETURN (14,12)
 *
 EXITNOK1 L     13,SAVEAREA+4
          LA    15,4                RC=4
          ST    15,16(13)
          RETURN (14,12)
 *
 SAVEAREA DS    18F
 *
 DEVDSECT DSECT
 DEVUSARE DS    0XL100
 DEVUSAR1 DS    0XL16               CONTROL FIELD
 DEVUSGID DS    CL2                 --GROUPID  OR X'00'
 DEVUSUID DS    CL4                 --USERID   OR X'00'
 DEVUSPSW DS    CL6                 --PASSWORD OR X'00'
          DS    CL4                 RESERVED
 DEVUSAR2 DS    CL3                 FILE ID OF QPAC DEF
 DEVUSAR3 DS    CL2                 FILE ORG OF QPAC DEF
 DEVUSAR4 DS    CL3                 RESERVED
 DEVUSAR5 DS    CL8                 DDNAME
 DEVUSAR6 DS    CL44                DSN
 DEVUSAR7 DS    CL24                RESERVED
 *
          END
```

Fig. 11: Skeleton of routine QPACUSER

Chapter 2. Input/Output Definitions

File Definitions (Fixed Length Records IBM z/OS)

General Format of the File Definitions

```
SAM:    IPF[n]=
        UPF[n]=[*DDname,] og [,rl,bl,opt ...]
        OPF[n]=

VSAM:   IPF[n]=
        UPF[n]= [*DDname,] VSAM [,rl,opt ...]
        OPF[n]=
```

IPF	=	input file definition implicit form
IPF*n*	=	input file definition explicit form
UPF	=	update file definition implicit form
UPF*n*	=	update file definition explicit form
OPF	=	output file definition implicit form
OPF*n*	=	output file definition explicit form
**DDname*		explicitly defined DD name, if missing, IPF/UPF/OPF is taken as DD name.
og	=	organization definition
rl	=	record length (as positional operand)
bl	=	block length (as positional operand)
opt	=	options

Additional Parameters for File Definitions (Fixed Length)

File Organizations

og	SQ SAM	general sequential (device independent). The storage medium assigned to this file is defined through the appropriate SYS-number or DD statement. No particular medium is assumed. This definition cannot be used for printer files, since SQ does not support additional functions such as page control.
	SD DISK	sequential disk file (type independent). A disk unit is assumed as the storage medium, if not, an error message will occur.
	MT TAPE	sequential magnetic tape file. A tape unit is assumed as the storage medium, if not, an error message will occur.
	PR PRINT	printer output (type independent). A printer unit is assumed as the output medium. This definition contains additional functions such as line and page control, and title lines.
	CARD	sequential card file (type independent). A physical/virtual card reader/puncher is assumed as the storage medium.
	VS VSAM KSDS ESDS RRDS	VSAM (virtual storage access method) is assumed as the access method.
	PDS PDSE	partitioned data sets (see chapter 10)
	VTOC	VTOC format 1 data (see chapter 10)
	SCAT	System or user catalog (see chapter 10)
	SLOG	System logger (see chapter 10)

Options for General Definitions

opt various options giving additional information to the file definition are available.
They are **not positional operands**, their sequence is therefore irrelevant. They are coded after the last positional operand, separated by a comma:

e.g.
```
IPF=TAPE,RL=120,BL=2400,WP=5001
```

BL=
BLKSIZE= block length (is ignored under z/OS).
For output files always defined in the DD statement.

RL=
LRECL= record length.
This defines an internal buffer size (default 32k).

CLR=C'*x*'
CLR=X'*xx*'
CLR=NO output area clear character
After each output from an output area, QPAC clears the area to X'00', with the exception of CD and PR files where it is cleared to X'40' (space).
CLR= can be used to define a different clear value in character or hexadecimal format.
CLR=NO causes the output area not to be cleared after each output.

CLE=C'*x*'
CLE=X'*xx*' input area clear character at EOF time.
QPAC clears input areas at EOF to X'FF'. With CLE= a different value can be defined in character or hexadecimal format.

DESC='' file description.
This option allows a 44 bytes file description to be associated with the file. It is shown in console messages and file statistics.

DSN= Data set name (z/OS)
May be directly specified within the file definition and makes a corresponding DD statement obsolete.

e.g.
```
IPF=SAM,DSN=TEST.FILE
```

WP= work area position.
This definition states that the record will be written into, or read from, the general work area. The definition refers to the location within the work area to be occupied by the record to be processed.
When WP= is defined, a **dynamic record area does not exist** for the relevant file definition. In addition to the implicit position symbols (IPOS*n*, OPOS*n* ...), WPOS*nnn* can be used for addressing.

FCA= file communication area.
The FCA defines the location within the general work area where information exchange should take place. One example of an FCA function is the communication of record lengths. It may also be used to pass on keys when, for example, the 'set generic key' function is being employed.

If the FCA is not defined, it will by default be dynamically defined and can only be addressed by attached symbol names. The FCA has a length of 256 bytes.

```
COBREC=bookname[/bookname] [,PRFX=YES]
PLIREC=bookname[/bookname] [,PRFX=YES]
```

Cataloged COBOL and PL/I record structures can be loaded from a source library and be associated with a file definition. The field names are converted to QPAC symbol names (- signs are converted to _ signs). Initial values and edit masks are ignored.
Multiple structure names can be specified separated by a / sign. If there is not room for all the names within the statement, continuation on the following line is achieved by the definition of a slash followed by a space in the current statement. The next structure name is then defined in the following statement. Leading blanks are allowed.
If `PRFX=YES` is specified then every symbol name will be prefixed by the short form of the file identification.
Based definitions are translated as QPAC based structures. The based pointer must be loaded with the correct address within the QPAC program. In z/OS these copy books are read in by use of the PDS DD name QPACCOPY (QPACCOPY DD ...).

Options for Tape File Definitions

opt `BWD` — read tape files with fixed record length or “undefined” backwards

Options for VSAM-File Definitions

opt NRS — no reset to empty state.
An output file will by default be treated as if having the RST attribute, i.e. the file will be considered to be empty and will be newly created.

The NRS option should be defined when the new output records are to be appended to the existing records, i.e. the output file (OPF) should not be treated as if it were empty.

ESD — entry sequenced dataset.
QPAC will find out for itself whether it is dealing with ESDS, RSDS or KSDS. The system message ERROR X'A0' occurs if only VSAM is specified as file organization, which is to be ignored. In order to suppress this message, the ESD option can be explicitly defined when dealing with an ESDS.

LSR
NOLSR — local shared resource
VSAM must or must not use a local shared resource pool

RLS — the VSAM file is classified as record level shared.

FCA= — file communication area.
The FCA enables an exchange of necessary information between the user and QPAC VSAM. At present, this information consists of return codes, the record lengths or key information.

FCA= defines the location within the general work area to be used for the information exchange. **A default of dynamic allocated FCA will be assumed should the FCA not be explicitly defined, and it may only be accessed by the attached symbol names.**
The FCA has a length of 256 bytes.

PSW= — password.
This option enables a password to be defined when a dataset is to be password protected. The password may consist of between 1 and 8 alphanumeric characters and does not appear on the QPAC listing.

RC=YES — return code/feedback code.
The VSAM return and feedback code is returned in the FCA without QPAC being terminated in case of an error. For random commands (READ, READGE ...) this option is the default.
The FCA field RC1 contains the binary return code, the FCA field RC2 contains the binary feedback code. If no error occurred, the FCA field RC contains X'0000'.
At EOF X'0804' is returned.
After SETGK X'0804' is returned (if exists).
After SETEK X'0810' is returned (if exists).

BWD — Read backwards.
The VSAM file is read backwards. This option is valid for ESDS with fixed record length and KSDS.

Options for Print File Definitions

opt RL=
LRECL=
record length.
The line width for a laser printer can be extended to 500 bytes by explicitly defining the record length. The default is 132 bytes.

LCT=
line count.
A line count independent of the system line count may be defined in the relevant print file definition. This line count only affects the print file it is defined under.

IPC
ignore page control by print output.
Automatic page control according to the line counter is suppressed with this definition. Also the print out of title lines because of an existing HDR= or TITLE subroutine are suppressed.

e.g.

```
z/OS:     OPF=PR,IPC
```

ASA
ASA control character for print output.
The ASA control character set will be used for the print output.

CCH
control character.
The first position in the print line contains the control character. If ASA is defined, an ASA format will be expected, if not, machine format is expected.

CLASS=
The SYSOUT class may be specified within the file definition. The definition of a separate DD SYSOUT statement is then obsolete:

e.g.

```
OPF=PR,CLASS=T
```

File Definitions (Variable Length Records)

Three types of file definitions can be used:

- standard variable mode blocked or unblocked
- variable spanned mode blocked or unblocked
- undefined mode, unblocked only

General Format of File Definitions

```
z/OS: IPF[n]=[*DDname,] og-VAR [,rl,bl,opt ...]
      UPF[n]=           og-SPN
      OPF[n]=           og-UND
```

Sequential files can be defined as being on magnetic tape or disk:

SQ-*type* SAM-*type*	sequential, device independent
SD-*type* DISK-*type*	sequential disk
MT-*type* TAPE-*type*	sequential tape
PR-*type* PRINT-*type*	printer (only *og*-VAR possible)
PDS-*type*	partitioned data set (z/OS)

The distinction between the three organizations is made by an additional definition:

og-VAR	standard variable mode. A record is never longer than the block.
og-SPN	variable spanned mode. The logical record can be longer than the block. It can extend over more than one block. Not valid for PDS.
og-UND	undefined mode. The physical record is read and made available.

Additional Parameters for File Definitions (Variable Length)

Variable Record Lengths and Block Lengths

rl

record length:
This is the longest possible record length. With VAR and SPN the first 4 bytes of each record contain the record descriptor word according to standard convention:

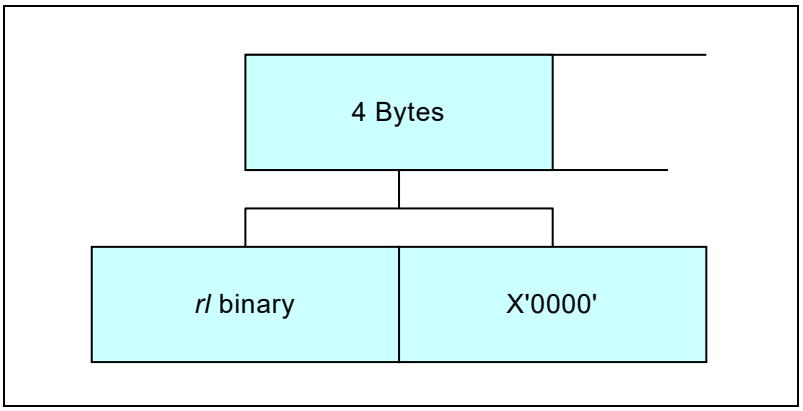


Fig. 12: The first 4 bytes: the record description word

The first two bytes contain the record length in binary form, the second two bytes are used by the IOCS.
The record length specification must include these 4 bytes.

bl

block length:
i.e. the longest possible block length (physical record). It is ignored in the UNDefined mode. In SPN output it is the length of the physical records to be written.

If the block length, under VAR, is equal to or not more than 4 bytes larger than the record length, it will always be rounded up to record length plus 4 bytes. In this case the file is considered to be unblocked. SPN makes no distinction between blocked or unblocked.

Options with Special Importance on Undefined Length Records

opt

In addition to the options described under fixed record lengths, the FCA definition (file communication area) in conjunction with UNDefined files is of particular importance.

FCA=

The FCA is used to interchange the actual record length between the user and QPAC. FCA= is used to define the working storage position, within the general QPAC work area, which serves as an information interchange for the appropriate file. **A default of dynamic allocated FCA will be assumed should the FCA not be explicitly defined and it may only be accessed by the attached symbol names:**

```
e.g.  OPF=SQ-UND,RL=4096,FCA=8000
```

For output records the record length in binary form must be put into the field ..LENG at displacement 12 (4 bytes), before the record is written. For input records, QPAC fills the field ..LENG with the record length (binary) of the record just made available.
(The FCA is **meaningless** for VAR and SPN, since the record length is part of the logical record in these cases).

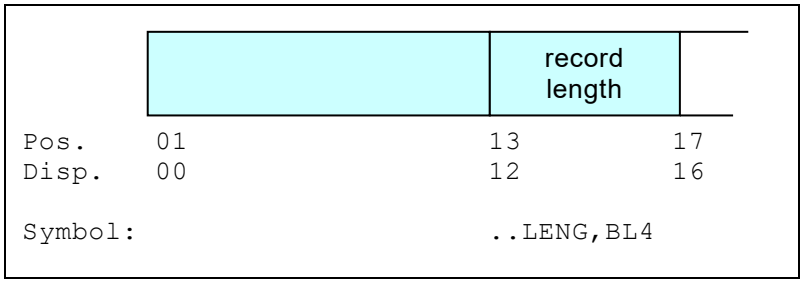


Fig. 13: File Communication Area (FCA)

The symbol ..LENG is attached to the length field in the FCA.

General Hints on File Definitions

An update file definition is the equivalent of an input and an output file definition. The file id number assigned to such a file may therefore not be given to any other file.

A DUMMY DD statement in z/OS job control is accepted by QPAC. It is also possible to define a DUMMY in QPAC as follows:

e.g.

```
IPF=DUMMY
OPF=DUMMY
```

These statements ensure the building up of the normal logical file control mechanism as would be the case if a physical file were present.
`IPF=DUMMY` does not result in the same conditions as under z/OS job control.
`IPF=DUMMY` never produces an EOF status. An EOF status can be achieved by adding a `LIPF=` statement to this file, or by giving a `CLOSE-I` instruction. The input area assigned is then cleared to X'FF' or to the value specified in `CLE=`.

Under z/OS, record length and block length can be defined by QPAC file definitions, but the block length is ignored. The record length is used for the size of the internal buffer. Job control specifications are decisive for the dataset.
If record lengths are defined through both mediums, the QPAC length must not be less than the JCL length, since QPAC has already built up an internal I/O area based on its file definitions. If the record length is missing in the QPAC file definition, an I/O area of 32K is reserved.

After each `PUT` the output area is normally cleared, whether the output records are processed in a dynamically allocated I/O area or in a working storage area defined by `WP=`. The clearing of the output area can be suppressed by the `CLR=NO` option.

General Points on VSAM

1. In the file definition, no distinction is made between ESDS, KSDS and RRDS. This information, together with any key length and key position information, is obtained by QPAC from the cluster definition in the VSAM catalog.

2. It is possible to process VSAM files with variable record lengths. The necessary communication is established through the FCA. The storage position of the FCAs can be defined as an option to the file definition, otherwise it will dynamically be allocated:

e.g.

```
IPF=VS,FCA=9000
OPF=VSAM
```

   - After each `GET` instruction (for output), the actual logical record length of the record supplied to the user is stored in binary form in the 4 bytes of the FCA, reserved for *`rl`*.
     When updating, this length should not be modified.
     The symbol `..LENG` is attached to the length field in the FCA.

   - Before a `PUT` instruction, the actual logical record length of the record to be written can be placed in binary form into the 4 bytes of the FCA (*`rl`* positions). If the FCA contains binary zeros, the maximum length is taken from the cluster definition.
     The symbol `..LENG` is attached to the length field in the FCA.

3. The *`rl`* definition within the file definition is only used for the internal reservation of a buffer area. If the storage partition is large enough, the *`rl`* definition may be omitted, in which case an area of 32760 bytes is reserved. For information purposes, QPAC states the largest encountered record length at the end of processing.

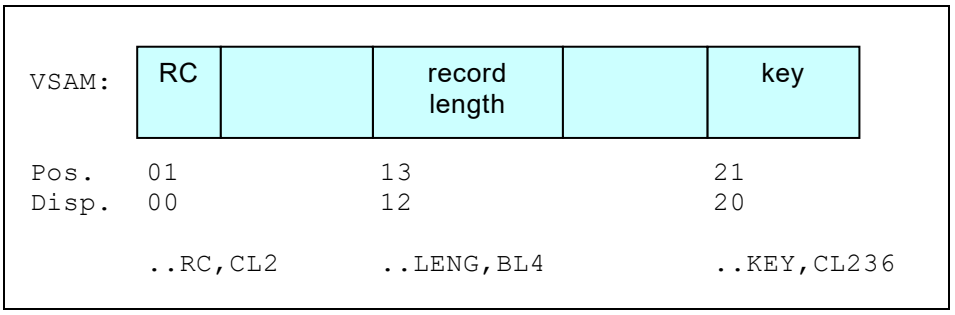


Fig. 14: File Communication Area (FCA)

General Points on Variable Record Lengths

1. Updating of VAR, SPN or UND disk files is supported. The actual record length may not be altered.

2. In processing, there is no difference between VAR and SPN. The only difference is in the file definitions, which are based on different forms of physical storage.

3. Under z/OS, the record format 'variable' 'spanned' or 'undefined' is taken from the job control and need not be given in the file definition. Where the record format is given in the file definition, this format will be the one used, and will not be overwritten. The default is 'fixed'.

4. SPN 'spanned' is not supported for z/OS partitioned data sets.

Dynamic File Allocation for z/OS

For the z/OS environment QPAC additionally supports the . Three different definition types and one additional command category are supported:

1. JCL static allocation
2. JCL dynamic allocation
3. Full dynamic allocation
4. Data set only commands

With **JCL static allocation** the DD statement is dynamically created.
The corresponding DSN and the DD name are a predefined part of the QPAC file definition.
JCL static allocation allows **implicit and explicit file definitions**.

e.g.

```
IPF=PDS,DSN=INPUT.DATASET
```

Fig. 15: JCL static allocation

With **JCL dynamic allocation** the DD statement is dynamically created.
The file organization and certain options are defined with the file definition but not the DSN and if required the DD name.
JCL dynamic allocation only allows the **explicit file definition**.

e.g.

```
IPF1=VSAM,DYNAMIC
IPF1=VSAM,DSN=DYNAMIC
```

Fig. 16: JCL dynamic allocation

With **full dynamic allocation** only a file definition skeleton is defined in the QPAC program. The completion of the file definition is done dynamically during execution time. Therefore, the file organization and all necessary additional information such as record length, block length, storage medium etc. in addition to DSN and DD name (if required) must be specified.

e.g.

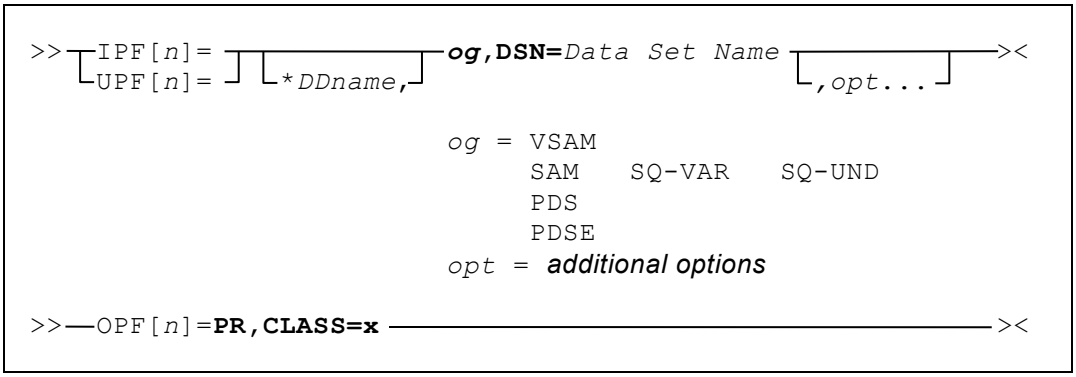
```
OPF1=DYNAMIC
```

Fig. 17: Full dynamic allocation

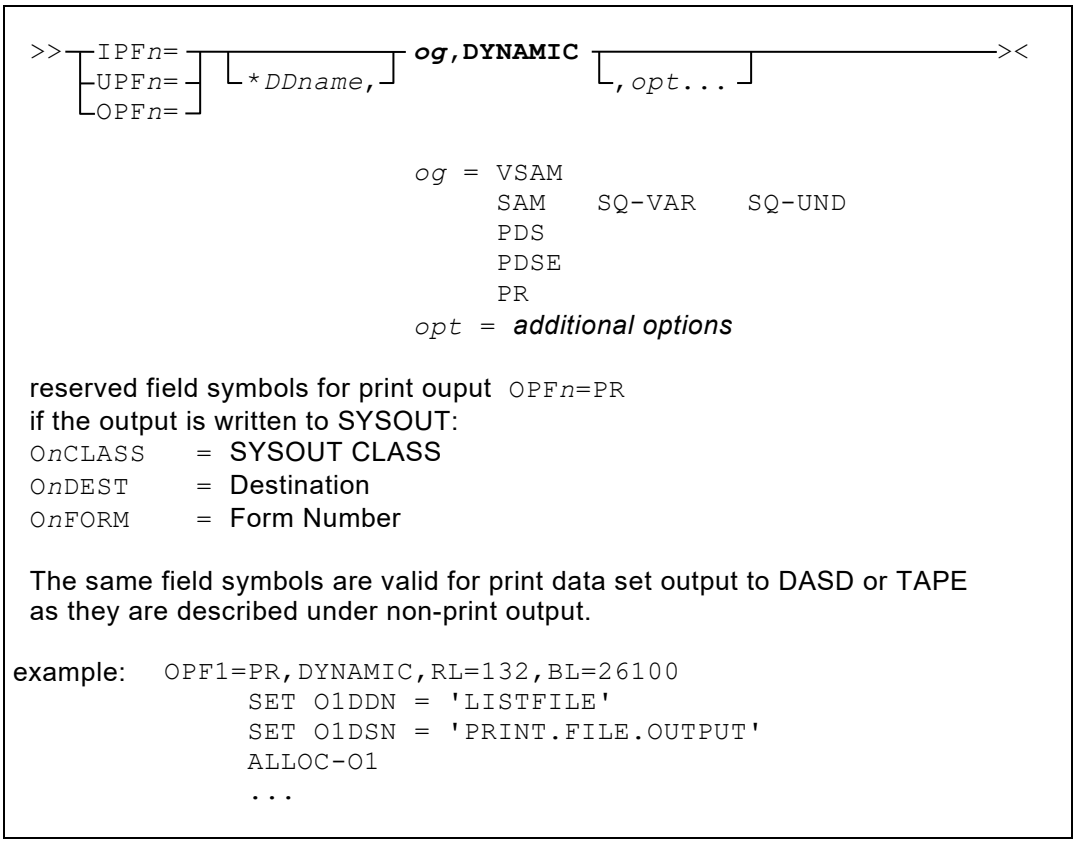
With print files a further differentiation exists. If a DSN is defined the output is written to a data set on DASD or tape, according to the medium specified. Default is DASD. But if a class (..CLASS =) is defined a SYSOUT DD statement is allocated. If both DSN and CLASS are present SYSOUT is prefered.

Data Set only Commands belong to a command category which allows to get information about a data set before allocation effectively is done. A group of reserved field symbols is associated with it whose names are all beginning with the prefix `ANY...`

Basic Format of File Definition for JCL Static Allocation



Basic Format of File Definition for JCL Dynamic Allocation



reserved field symbols for print ouput `OPFn=PR`
if the output is written to SYSOUT:

`OnCLASS` = SYSOUT CLASS
`OnDEST` = Destination
`OnFORM` = Form Number

The same field symbols are valid for print data set output to DASD or TAPE as they are described under non-print output.

example:
```
OPF1=PR,DYNAMIC,RL=132,BL=26100
     SET O1DDN = 'LISTFILE'
     SET O1DSN = 'PRINT.FILE.OUTPUT'
     ALLOC-O1
     ...
```

Basic Format of File Definition for Full Dynamic Allocation

```
>>─┬─IPFn=─┬─ DYNAMIC ──────────────────────────────────────────────><
   ├─UPFn=─┤            └─,opt...─┘
   └─OPFn=─┘
```

the file organization must be filled into the field ..DSORG:
PS or SQ, PO, PE or VS

reserved field symbols for full dynamic:
..DSORG = data set organization:
PS = SAM, PO = PDS, PE = PDSE, VS = VSAM
..VOLID = volume id: e.g. for tape, 'SCRTCH' is valid for OPFn=

full dynamic allocation for data sets whose organization is not yet predefined can only be used with explicit file definitions.

The definitive organization is allocated with the ALLOC-.. command. Therefore, the reserved filed symbols shown in the following examples must be filled in advance.

examples:

```
IPF1=DYNAMIC                OPF1=DYNAMIC
      SET I1DSN = ...             SET O1DSN = ...
      SET I1DSORG = ...           SET O1DSORG = ...
      SET I1RECFM = ...           SET O1SDISP = ...
      SET I1RL = ...              SET O1NDISP = ...
      SET I1BL = ...              SET O1CDISP = ...
      ALLOC-I1                    SET O1RL = ...
                                  SET O1BL = ...
                                  SET O1UNIT = ...
                                  SET O1RECFM = ...
                                  SET O1LABEL = ...
                                  ALLOC-O1
```

Additional Commands for JCL Dynamic and Full Dynamic Allocation

ALLOC-I*n* -U*n* -O*n*	Allocate Data Set
UNALLOC-I*n* -U*n* -O*n*	Unallocate Data Set

The ALLOC command must be executed **before** the OPEN command. The UNALLOC command must be executed **after** the CLOSE command. Both commands are using the SVC 99.
In case of any error the SVC 99 return code is returned and can be examined if RC=YES has been defined. Otherwise, execution is terminated.

e.g.

```
IF I1RC NOT = X'0000' THEN not ok
```

Reserved Field Symbols for JCL Dynamic and Full Dynamic Allocation

reserved field symbols for **IPF**:

I*n*DSN	=	'Data set name'
I*n*DDN	=	'DDname' Default is IPF*n*
I*n*SDISP	=	Status disposition: 'S' = SHR, 'O'=OLD
I*n*NDISP	=	Normal disposition: 'C' = CATLG, 'K' = KEEP, 'D'=DELETE, 'U' = UNCATLG
I*n*CDISP	=	Cancel disposition: 'D' = DELETE, 'K' = KEEP, 'C' = CATLG
I*n*MN	=	Member name, also possible in generic format (PDS only)
I*n*MEMNM	=	Member name, full name from PDS as SAM file

reserved field symbols for **UPF**:

U*n*DSN	=	'Data set name'
U*n*DDN	=	'DDname' Default is UPF*n*
U*n*SDISP	=	Status disposition: 'S' = SHR, 'O'=OLD
U*n*NDISP	=	Normal disposition: 'C' = CATLG, 'K' = KEEP, 'D'=DELETE, 'U' = UNCATLG
U*n*CDISP	=	Cancel disposition: 'D' = DELETE, 'K' = KEEP, 'C' = CATLG

reserved field symbols for **OPF** (not for print output):

O*n*DSN	=	'Data set name'
O*n*DDN	=	'DDname' Default is OPF*n*
O*n*RL	=	Record length
O*n*BL	=	Block length
O*n*SDISP	=	Status disposition: 'N' = NEW, 'M' = MOD, 'O'=OLD
O*n*NDISP	=	Normal disposition: 'C' = CATLG, 'K' = KEEP, 'D'=DELETE
O*n*CDISP	=	Cancel disposition: 'D' = DELETE, 'K' = KEEP, 'C' = CATLG
O*n*TYPSP	=	Type of space: 'C' = CYL, 'T'=TRK, 'B' = BLK, 'K' = KB, 'M' = MB
O*n*PRISP	=	Primary space: 3 or *nnn*
O*n*SECSP	=	Secondary space: 3 or *nnn*
O*n*DIRBL	=	Directory blocks for PDS: 32 oder *nnn*
O*n*RLSE	=	Release space: 'N' = NO or 'Y' = YES
O*n*UNIT	=	Storage medium: 'SYSALLDA' or '*xxxxxxxx*'
O*n*DCLAS	=	Data class name: '*xxxxxxxx*'
O*n*MCLAS	=	Mgment class name: '*xxxxxxxx*'
O*n*SCLAS	=	Storage class name: '*xxxxxxxx*'

Tape Full Dynamic Allocation

If dynamic allocation for tape output is used a DD statement must be present that will cause the tape units to be reserved. The DD statement must contain the UNIT parameter, the DISP parameter and if necessary the VOL parameter.

e.g.

```
//OPF1     DD  UNIT=(TAPE,2,DEFER),DISP=(MOD,KEEP,KEEP),
//             VOL=(,,,20)
```

Data Set Only Commands

The following commands are related to the processing of data sets without a necessary corresponding file defintion. They use a group of reserved field symbols with names beginning with the prefix ANY... These predefined fields are necessary for the exchange of information. This command category allows to get information about a data set before allocation effectively is done. For example, it allows testing whether the data set already exists and must be deleted in advance.

ANYDSN	CL22	Data Set Name
ANYDDN	CL8	DD Name
ANYDIRBL	BL2	Directory Blocks
ANYDSORG	CL2	Data Set Organization PS, PO, VS
ANYLABEL	CL2	Tape Label NL,SL
ANYPRISP	BL2	Primary Space
ANYSECSP	BL2	Secondary Space
ANYSPTYP	CL1	Type of Space in Tracks (T)
ANYRECFM	CL3	Record format
ANYCREDT	CL8	Creation date
ANYREFDT	CL8	Last referenced date
ANYEXPDT	CL8	Expiration date
ANYBL	BL4	Block length
ANYRL	BL4	Record length
ANYUNIT	CL8	Unit type SYSDA or TAPE
ANYRC	CL2	Return code
ANYVOLID	CL6	Volume serial identification
ANYDCLAS	CL8	Data class name
ANYMCLAS	CL8	Management class name
ANYSCLAS	CL8	Storage class name

Neutral Commands

INQDSN-ANY	Inquire Data Set

This command allows the inquiry of a data set.
The field ANYDSN must be filled with the name of the data set in advance. As a result, the fields ANYVOLID, ANYRECFM, ANYDSORG, ANYBL, ANYRL, ANYUNIT, ANYLABEL, ANYPRISP, ANYSPTYP are returned, if the return code ANYRC contains X'0000'.
In case of an error the field ANYRC contains the return code (see the following list).

INQDDN-ANY	Inquire Data Set via DD-Name of DD Statement

This command allows the inquiry of a data set via DD name.
The field ANYDDN must be filled with the name of the DD statement in advance.
As a result the fields ANYDSN, ANYVOLID, ANYRECFM, ANYDSORG, ANYBL, ANYRL, ANYUNIT, ANYLABEL, ... are returned, if the return code ANYRC contains X'0000'.
In case of an error the field ANYRC contains the return code (see the following list).

UNCDSN-ANY	Uncatalog Data Set

This command allows any data set to be uncataloged. The VTOC will still contain the file. The field `ANYDSN` must be filled with the name of the data set in advance.
SMS controlled data sets will not be affected by this command and the return code is 0.
In case of an error the field `ANYRC` contains the return code (see the following list).

DELDSN-ANY	Delete Data Set

This command allows any data set to be removed from the catalog and to be deleted in the VTOC. Tape data sets are only removed from the catalog.
The field `ANYDSN` must be filled with the name of the data set in advance.
In case of an error the field `ANYRC` contains the return code (see the following list).

ANYRC or ..RC Return Codes

`X'....' =`	SVC 99 return codes
`X'2560' =`	definition error
`X'2222' =`	technical cancel error
`X'9999' =`	function code error

A PDS member selection can be dynamically defined. There are two ways to accomplish this:

1. A single member is read as a SAM file, after the JCL priciple DSN=fileid(member), e.g. `SET I1MEMNM = 'MEMNAME'`

2. A generic member selection can be defined that is sequentially read from the PDS file, e.g. `SET I1MN = 'ABC*'`
   All the members that are beginning with ABC are selected. The rule for the generic definition format is the same as described under the static PDS file definition.

Chapter 3. Input/Output Instructions

Instructions Overview

System related Instructions

GETIN	read from the system reader
PUTPCH	punch on to the system puncher
PUTLST	write on the system printer

Fig. 18: Overview of system related instructions

File related Instructions

OPEN	-I	open dataset
	-O	
	-U	
CLOSE	-I	close dataset
	-O	
	-U	

Fig. 19: Overview of file related instructions

I/O Instructions for Sequential Processing

GET	-I	read next record
	-U	
PUT	-O	write new record
	-U	rewrite record
PUTA	-U	insert record
PUTD	-U	delete record
SETGK	-I	set key greater or equal to
	-U	
SETEK	-I	set key equal to
	-U	

Fig. 20: Overview of I/O instructions for sequential processing

Random Instructions for VSAM Files

READ	-I*n* -U*n*	read random with key equal or relativ record number
READGE RDGE	-I*n* -U*n*	read random with key greater or equal or relative record number (range ..RRN+100)
READUP RDUP	-U*n*	read for update random with key equal or relative record number
REWRITE RWRT	-U*n*	rewrite record previously read by READUP
INSERT ISRT	-U*n*	add a new record
DELETE DLET	-U*n*	delete a record

Fig. 21: Overview of random instructions for VSAM

General Format

OPERATION-*file identification*

OPEN-IPF	OPEN-I	OPEN-O	OPEN-O3
CLOSE-OPF9	CLOSE-O9	CLOSE-I	CLOSE-O
GET-IPF8	GET-I8	GET-U2	GET
PUT-OPF1	PUT-O1	PUT-U1	PUT

Fig. 22: Sample I/O instructions

The file identification can be omitted if '**implicit processing logic**' is used, i.e. file identifications have no numbers. The file identification can be coded in short form (GET-I7).

File Related Instructions

The OPEN Instruction

```
OPEN  -I[n]
      -O[n]
      -U[n]
```

An OPEN command can be given at any time but is **not necessary** for the first opening of a file. A GET command automatically opens the addressed file; the GET command however, only if the file is not already opened. The re-opening of a file can only be accomplished by using the OPEN command.

A file identification must always be added to the OPEN function.

The CLOSE Instruction

```
CLOSE  -I[n]
       -O[n]
       -U[n]
```

A CLOSE command can be given at any time, but is **not necessary**, since automatic closing takes place for input files on EOF condition, and for output files at processing end.

Repeated reading of an input file is possible by using CLOSE-I OPEN-I, even if the file has not yet reached EOF.

A file identification must always be added to the CLOSE function.

ALLOC / UNALLOC for Dynamic File Allocation (z/OS)

```
ALLOC   -In              allocate data set
        -Un
        -On
```

```
UNALLOC   -In            unallocate data set
          -Un
          -On
```

Before the ALLOC instruction the provided reserved field symbols must be filled up with the corresponding values; with IPF at least InDSN.

```
IPF1=PDS,RL=80,DSN=DYNAMIC

SET I1DSN = 'SYSX.USER.LIB1'
ALLOC-I1
OPEN-I1
...
GET-I1
...
CLOSE-I1
UNALLOC-I1
SET I1DSN = 'SYSX.USER.LIB2'
ALLOC-I1
OPEN-I1

CLOSE-I1
END
```

Fig. 23: Sample command sequence for dynamic file allocation

I/O Instructions for Sequential Processing

The GET Instruction

```
GET [ -I[n] ]      [AT-EOF . . . ATEND]
    [ -U[n] ]
```

Through the GET command, a logical input record is made available to the user. All input addresses in the processing instructions following the GET refer automatically to the input area activated by that GET, but only in hierarchical order.

If the GET command refers to an update file, the following output addresses refer to the same area.

A GET command can immediately follow an EOF control block (AT-EOF), so that it will be executed when the EOF status is met. If the EOF control block is missing, the whole GET block is skipped on EOF status. If the EOF control block is present, it is executed when EOF occurs, and no GET block skip occurs. Normal processing is resumed after the ATEND definition.

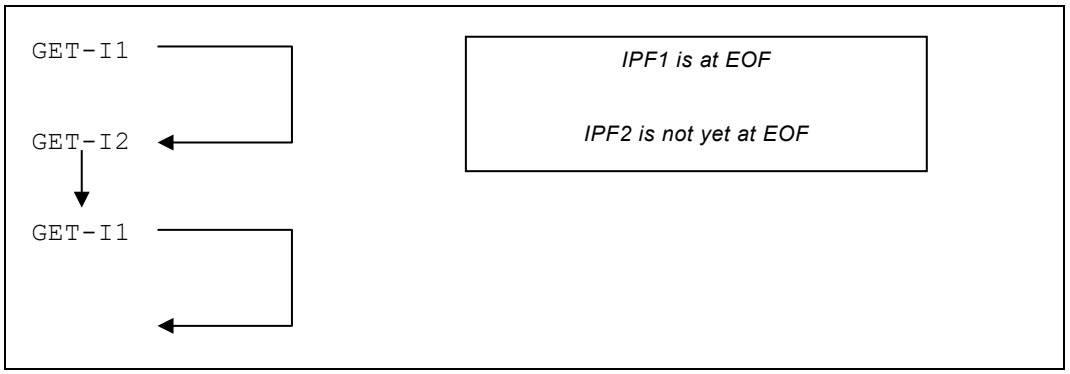


Fig. 24: Example of GET commands without EOF control block

Notice:
Any GET command on hierarchical level 0 marks the corresponding file definition as **leading file**.

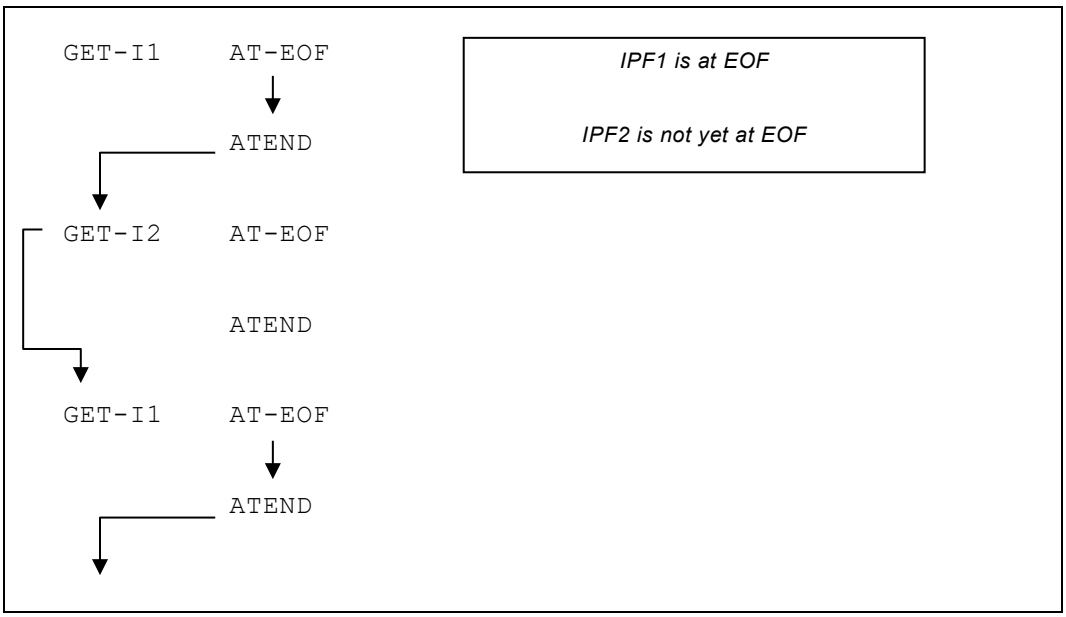


Fig. 25: Example of GET commands with EOF control block

The PUT Instruction

```
PUT  [ -O[n] ]
     [ -U[n] ]
```

The `PUT` command writes a logical record to an output file, or updates an 'update file'.
After execution of the command, the output area, **but not the update area**, is automatically cleared. (Clear character = X'00' resp. X'40' for card and printer output; or `CLR=X'..'`).

With VSAM KSDS, DB2 and DL/I databases, two additional I/O commands are supported.
The DB2 usage is described in the SQL/DB2 Support Feature, the usage of DL/I in the DL/I Support Feature.
The following information is concerned with VSAM datasets.

The PUTA (Put Addition) Instruction

```
PUTA [ -U[n] ]
```

This command can only be used for **update files**, and not where the file definition is for input only (IPF=) or for output only (OPF=).

The command causes the record currently in the I/O area to be written. The record length, if required, can be given in the FCA. If the length in the FCA is null, then the maximum length given in the cluster definition will be assumed. When processing RRDS files the actual relative record number must also be set into the FCA field ..RRN.

PUTA works with the RPL for sequential processing. The positioning will be controlled.

The PUTD (Put Delete) Instruction

```
PUTD [ -U[n] ]
```

This command can only be used for **update files**, and not where the file definition is for input only (IPF=) or for output only (OPF=).

The command causes the last record read by a GET to be deleted.

The FCA is not used for this command.

The SETGK and the SETEK Instructions

```
SETGK [ -I[n] ]
      [ -U[n] ]

SETEK [ -I[n] ]
      [ -U[n] ]
```

SETGK is an extended I/O operation for files built on keys, that allows processing to continue at, or after, a given generic key.

Processing continues with the **key equal to or greater than** the key given.

SETEK is an extended I/O operation for files built on keys, that allows processing to continue at a given equal key.

Processing continues with the **key equal to** the key given. The record with the same key must be present, otherwise a 'not found' condition appears.

Before execution of the SETGK or SETEK function, the key value must be moved to the FCA, starting at position 21 of the FCA (displacement 20). The use of positions 1 - 20 of the FCA varies according to the file organization.

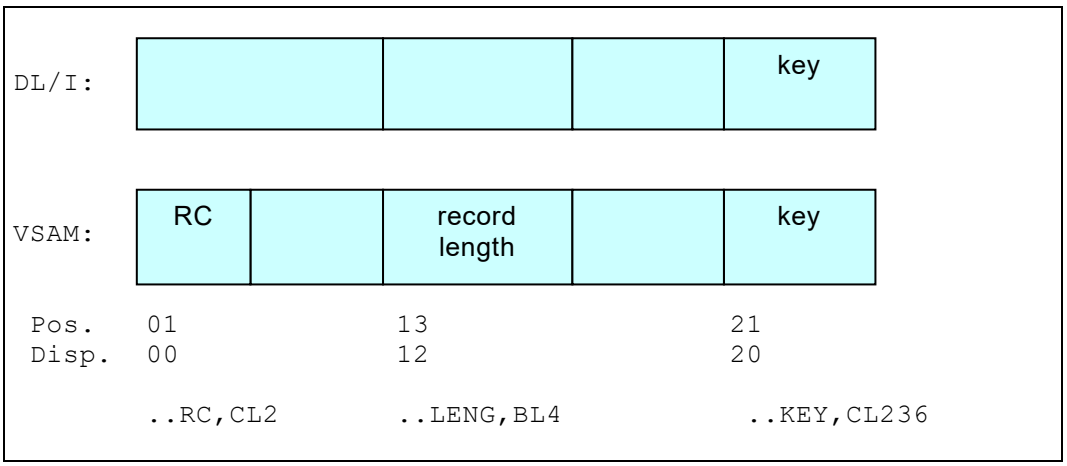


Fig. 26: File Communication Area (FCA)

The SETGK/SETEK contains an OPEN if the current file state is "closed".

Random Instructions for VSAM Files

The READ Instruction

```
READ-In
     -Un
```

Read random with key equal or relative record number.

a) VSAM KSDS
The record specified by the key in the FCA field ..KEY is read.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET I1KEY = '123456'
READ-I1
IF I1RC = X'0810' THEN ...    (no record found)
```

Fig. 27: Return code check after READ instruction

b) VSAM RRDS
The record specified by the relative record number in the FCA field ..RRN is read.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET I1RRN = 1000
READ-I1
```

Fig. 28: VSAM RRDS direct access with READ

The READGE Instruction

```
READGE-In
RDGE  -Un
```

Read random with key greater or equal or with relative record number (range ..RRN+100).

a) VSAM KSDS
The record specified by the key in the FCA field ..KEY or the next record in sequence is read.
In the FCA field ..RC the original VSAM return- and feedback code is returned:

Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET I1KEY = '123456'
READGE-I1
IF I1RC = X'0810' THEN ...    (no record found)
```

Fig. 29: Return code check after READGE instruction

b) VSAM RRDS
The record specified by the relative record number in the FCA field ..RRN or the next in sequence in the range of the next 100 slots is read.
In the FCA field ..RC the original VSAM return- and feedback code is returned:

Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET I1RRN = 1000
READGE-I1
```

Fig. 30: SAM RRDS direct access with READGE

The READUP Instruction

```
READUP-Un
RDUP
```

Read for update random with key equal or relative record number.

a) VSAM KSDS
This function is the same as explained under READ, but the file definition must be UPF*n*.

```
SET U1KEY = '123456'
READUP-U1
```

Fig. 31: Read for Update mit der READUP instruction

b) VSAM RRDS
This function is the same as explained under READ, but the file definition must be UPF*n*.

The REWRITE Instruction

```
REWRITE-Un
RWRT
```

Rewrite the record previously read by READUP.

a) VSAM KSDS
The record in the record area is written back (updated).
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET U1KEY = '123456'
READUP-U1
SET U1POS20,CL7 = 'NEUWERT'

REWRITE-U1
IF U1RC1 <> X'00' THEN ...    (any error condition occurred)
```

Fig. 32: Update a record with REWRITE after READUP

b) VSAM RRDS
The record in the record area is written back (updated).
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

The INSERT Instruction

```
INSERT-Un
ISRT
```

Insert a record.

a) VSAM KSDS:
The record in the record area will be inserted. The key within the record is relevant.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

b) VSAM RRDS:
The record in the record area will be inserted into the empty slot.
The relative record number (RRN) must be stored into the field ..RRN in advance.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET U1RRN = 5
INSERT-U1
IF U1RC1 <> X'00' THEN ...    (any error condition occurred)
```

Fig. 33: Insert a record with INSERT

The DELETE Instruction

```
DELETE-Un
DLET
```

Delete a record.

a) VSAM KSDS
The record whose key is stored in the field ..KEY will be deleted.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET U1KEY = '123456'
DELETE-U1
IF U1RC1 <> X'00' THEN ...    (any error condition occurred)
```

Fig. 34: Delete a record with DELETE

b) VSAM RRDS
The record whose relative record number is stored in the field ..RRN will be deleted.
In the FCA field ..RC the original VSAM return- and feedback code is returned:
Position 1 (..RC1) contains the return code X'00', X'04', X'08', X'0C' etc.
Position 2 (..RC2) contains the feedback code X'..'.

```
SET U1RRN = 5
DELETE-U1
IF U1RC1 <> X'00' THEN ...    (any error condition occurred)
```

Fig. 35: Delete a record with DELETE

Printer File related Instructions

As well as the `PUT` command, the following output commands can be used for printer files:

WNSP [-O[*n*]]	write no space	(ASA)
W	write and 1 space	|
WASP1	write and 1 space	|
WASP2	write and 2 spaces	|
WASP3	write and 3 spaces	|
WASK1	write and skip to channel 1	|
WASK2	write and skip to channel 2	|
WASK3	write and skip to channel 3	|
WASK4	write and skip to channel 4	|
WASK5	write and skip to channel 5	|
WASK6	write and skip to channel 6	|
WASK7	write and skip to channel 7	|
WASK8	write and skip to channel 8	|
WASK9	write and skip to channel 9	|
WASK10	write and skip to channel 10	|
WASK11	write and skip to channel 11	|
WASK12	write and skip to channel 12	(ASA)
SP1	immediate 1 space	
SP2	immediate 2 spaces	
SP3	immediate 3 spaces	
SK1	immediate skip to channel 1	
SK2	immediate skip to channel 2	
SK3	immediate skip to channel 3	
SK4	immediate skip to channel 4	
SK5	immediate skip to channel 5	
SK6	immediate skip to channel 6	
SK7	immediate skip to channel 7	
SK8	immediate skip to channel 8	
SK9	immediate skip to channel 9	
SK10	immediate skip to channel 10	
SK11	immediate skip to channel 11	
SK12	immediate skip to channel 12	

Fig. 36: Instructions overview for printer files

```
W-OPF1       SP1-O2
W-O1         WASK1-O3
SK1-O2
```

Fig. 37: Sample instructions for printer files

The output area is not cleared by an immediate command.

The special command explained below is available for cataloging member names into a PDS directory:

```
STOW   [ -U[n] ]
       [ -O[n] ]
```

The name of the member last created is taken from the FCA field `..MEMNM` and the member is cataloged into the PDS directory.
If the FCA field `..STOWID` contains an 'A' the member must not already exist.
If it contains an 'R' an existing member with the same name will be replaced.
If it contains no invalid code, then 'A' is assumed.

Additionally for updating the statistical records in the directory record the version modification and/or the user id can be specified. Therefore the following reserved field symbols are available: `..STOWVV`, `....STOWMM` and `....STOWUSER`.

System related Instructions

The GETIN Instruction

```
GETIN
```

This read command reads in a record from the in-stream file (QPACIN).

Data that follows the `END` statement of QPAC are read, without further file definition, into **working storage positions 5001 - 5080**.

At EOF, that area is set to high-value (X'FF').

The PUTLST Instruction

```
PUTLST
PUTLST-'literal'
PUTLST-symbolname[,l]
PUTLST-'literal',symbolname[,l],'literal'...
```

This write command writes a line to the system printer (QPACLIST) without needing further file definitions. **Working storage positions 5201 - 5320** contain the line to be written as far as no literal or field symbol have been specified. Position 5200 is reserved for the storage of an ASA control character (default value is blank).

Before the first `PUTLST` from working storage position 5201-5320, a page skip is effected. The output area is not cleared automatically.
A width of 121 positions will be assumed for a print line (including control character). The `PARM` option `LISTL=` (list length) enables the width of the print line to be extended to 250 positions.

Multiple operands can be defined, separated by comma, literals or field symbols. The length of their contents must not be longer than 120 bytes.

The PUTPCH Instruction

```
PUTPCH
```

This punch command punches one card on the system punch (QPACPUN).

Working storage positions 5101 - 5180 contain the record to be punched.

The punch area is not cleared automatically.

Titles for Printer Files

The HEADER Definition (Static Title Lines)

```
HDR=
HDR-On=
```

A print output title line is defined by a pair of HDR definitions.
The first HDR defines print positions 1-66, the second HDR defines print positions 67-132.

The sequence of the HDR definitions dictates the sequence of the title lines.

HDR definitions with file identification number (HDR-O1) can only cover print positions 1-65 or 68-131.

Please note that the **TITLE routine** (described on the next page) can be used instead of the static HDR definitions. If both, HDR and TITLE are defined for the same print file, HDR is ignored.

Please note that if the file definition parameter IPC is specified, no title lines are produced.

The TITLE Definition (Dynamic Title Lines)

As well as the possibility of defining static title lines by the means of HDR, dynamic title lines, whose contents can be altered during execution, can also be defined.

TITLE	**TITLEND**
TITLE-O*n*	**TITLEND**

TITLE to TITLEND is a subroutine structure block that is **executed automatically** at 'title time'.

'Title-Time' is:

- immediately before writing the first print line
- when the line counter maximum is reached
- immediately before an SK1 or a WASK1 command

The key word TITLE (with file identification if using explicit processing logic) marks the beginning of the routine structure block, and TITLEND its end.

The TITLE definition implicitly contains a skip command to the new page, and an output address assignment to the printer output area. The cleared printer output area is made available to the user. The implicit assignment of an input area is not uniquely defined at title time.

When using printer file definition for **implicit processing logic** (OPF=), the TITLEND definition contains a write command (W). It is however possible to explicitly specify **additional write commands within the routine**, if several title lines are to be written.
The address assignment after TITLEND is the same as that before TITLE.

The TITLE routine structure block can be placed anywhere within the QPAC definitions, but only after the respective printer file definition.

```
IPF=...
OPF=PR

TITLE
 SET OPOS20          = 'TITLE LINE 1'
 SET OPOS90          = 'DATE:'
 SET OPOS96          = DATE
 W
 SET OPOS20          = 'TITLE LINE 2'
 SET OPOS90          = 'PAGE:'
 SET OPOS96,M'ZZZ9'  = OPCNT
TITLEND

normal processing automatically skips the TITLE block
```

Fig. 38: The dynamic TITLE routine

Please note that if the file definition parameter IPC is specified, no title lines are produced.

Processing Limit Definitions

General format

```
LIPF[n]=s-e    [ ,s-e,s-e, ... ]
LUPF[n]=
LOPF[n]=                                [ ,EOP ]

LIDB[n]=  (for data bases)
LUDB[n]=
LODB[n]=
```

The limitation refers to the number of records to be processed for the file.

s (start record) defines the first record to be processed
e (end record) defines the last record to be processed

```
LIPF=2-50          start processing with 2nd record and end after 50th
                        record (51st record forces EOF condition)
```

Fig. 39: Processing limit definition general format

An unlimited number of *from-to* groups can be defined in one statement.
The *from-to* groups can also be split into two or more statements:

```
LIPF=1-100,200-350
LIPF=400-1000
```

Fig. 40: Processing limit definition (several from-to groups)

The numbers appearing in the *from-to* groups must be **in ascending order**.

`EOP` forces an end of processing if the output limit is reached. If there are multiple `EOP` definitions all the output limits must be reached before processing is terminated.

```
LOPF=50-100,EOP                    write records 50-100;
                                   then end of processing
```

Fig. 41: End of processing when output limit is reached

Special formats

`LIPF[n]=-e`	(*s*=1 is assumed)
`LIPF[n]=s-`	(*e*=EOF is assumed)

`LIPF1=-100`	only records from 1 to 100 are read
`LUPF3=10-`	only records from 10 to EOF are read for update

Fig. 42: Processing limit definition special formats

Operator Communication Instructions

The WTO Instruction (Write to Operator with no Response)

```
WTO-sendg_field[,length]
WTO-'literal'
```

The contents of the sending field in a length of *length* are displayed on the console.
A character constant can be defined as sending field.
Lower case letters are automatically translated to upper case.

```
WTO-WPOS5000,10

WTO-'MESSAGE TO OPERATOR'
```

Fig. 43: Operator communication - console output

The WTOR Instruction (Write to Operator with Response)

```
WTOR-'literal',recvg_field[,length]
WTOR-sendg_field[,length],recvg_field[,length]
```

The contents of the sending field in a length of *length* are displayed on the system console. The receiving field contains the operator's response.

Synchronisation Instructions

```
ENQ-fieldname  [,STEP]
ENQ-'literal'  [,SYSTEM]
               [,SYSTEMS]
```

The usage of the ENQ (enqueue) commands is basically the one of the ENQ macro as described in the IBM literature.

The contents of the assigned field symbol or directly defined literal is divided into two parts. The first 8 bytes are the major part or queue name, the following part is the minor part or resource name.
The maximum length of the literal or field is 128 bytes.
STEP, SYSTEM or SYSTEMS define the "scope" as described in the IBM literature.
Scope SYSTEMS is the default if nothing is specified.

```
DEQ-fieldname  [,STEP]
DEQ-'literal'  [,SYSTEM]
               [,SYSTEMS]
```

The DEQ command releases any ENQed resource.

Chapter 4. Static Program Structure

Automatic Processing Control

NORMAL	(optional)
LAST	(optional)
END	(necessary)

Processing can be structured into major control sections by these three keywords.

The END Instruction

The **END** statement is a control instruction for the QPAC assembler.

It declares the physical end of all definitions. Information following on the same line as the END command is ignored. Following records are considered to be either data or JCL records.

If only the END statement is defined, the whole QPAC program consists of one main processing part:

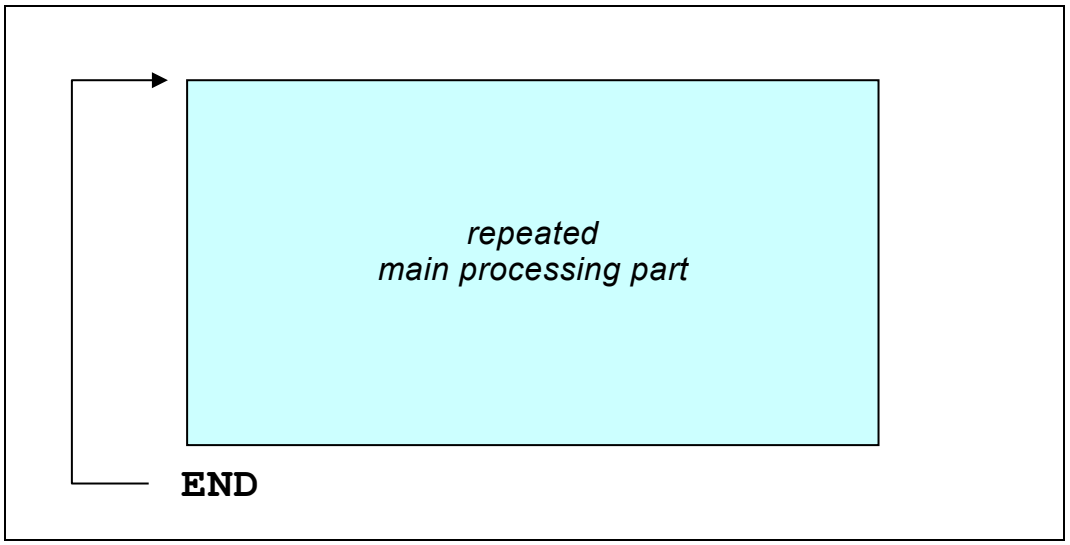


Fig. 44:The END statement is the physical end of all definitions

The END statement can be coded anywhere between positions 1 and 71:

e.g.

```
IPF=VS  OPF=VS     SET OPOS1 = IPOS1,CL80     END
```

The NORMAL Instruction

If in addition to the END statement a **NORMAL** statement is defined, the preceding program part will become a **single processed introduction part**. The repeatedly processed main part follows the NORMAL statement:

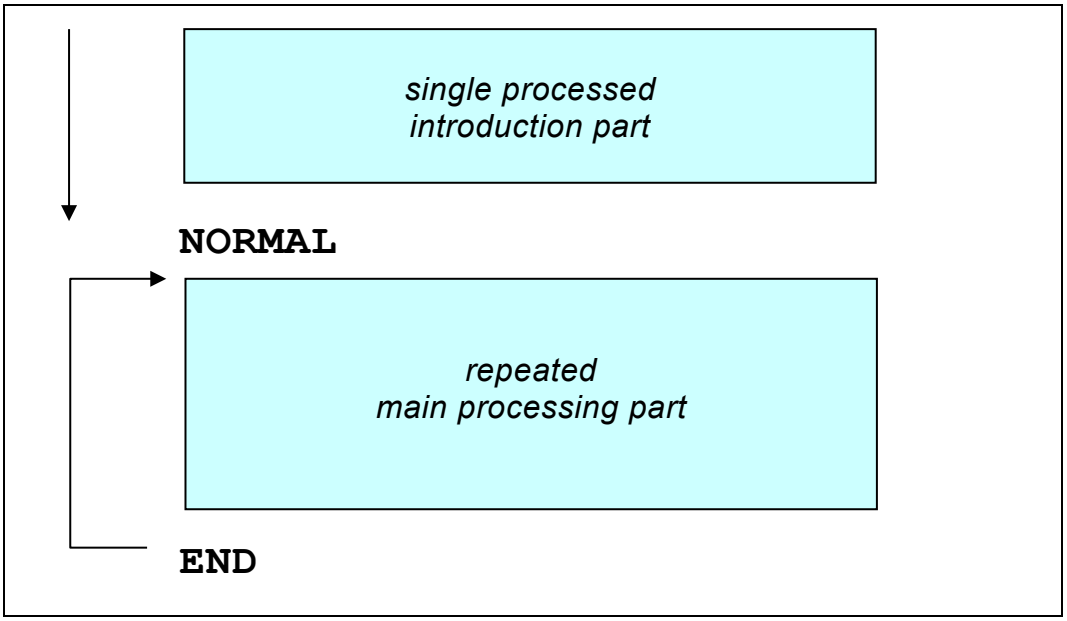


Fig. 45: Preprocessing with the NORMAL instruction

The LAST Instruction

For performing a termination routine when end of normal processing is reached, (usually EOF), the **LAST** statement can be coded. The instructions following LAST are executed **only once, at termination time**:

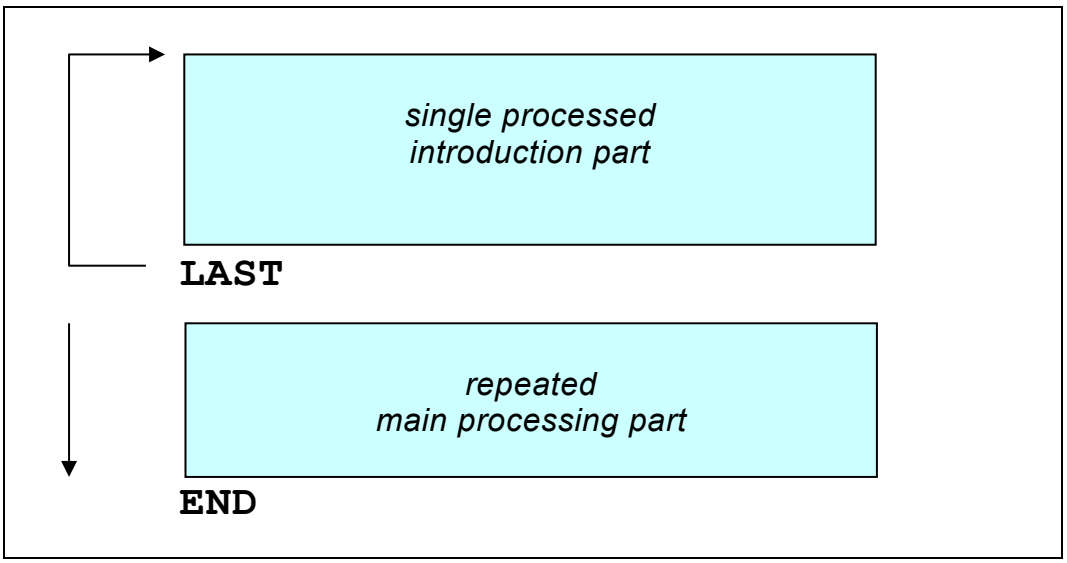


Fig. 46: End processing with the LAST instruction

The FIRST Instruction

It is possible to define several processing sequences with the **FIRST** instruction:

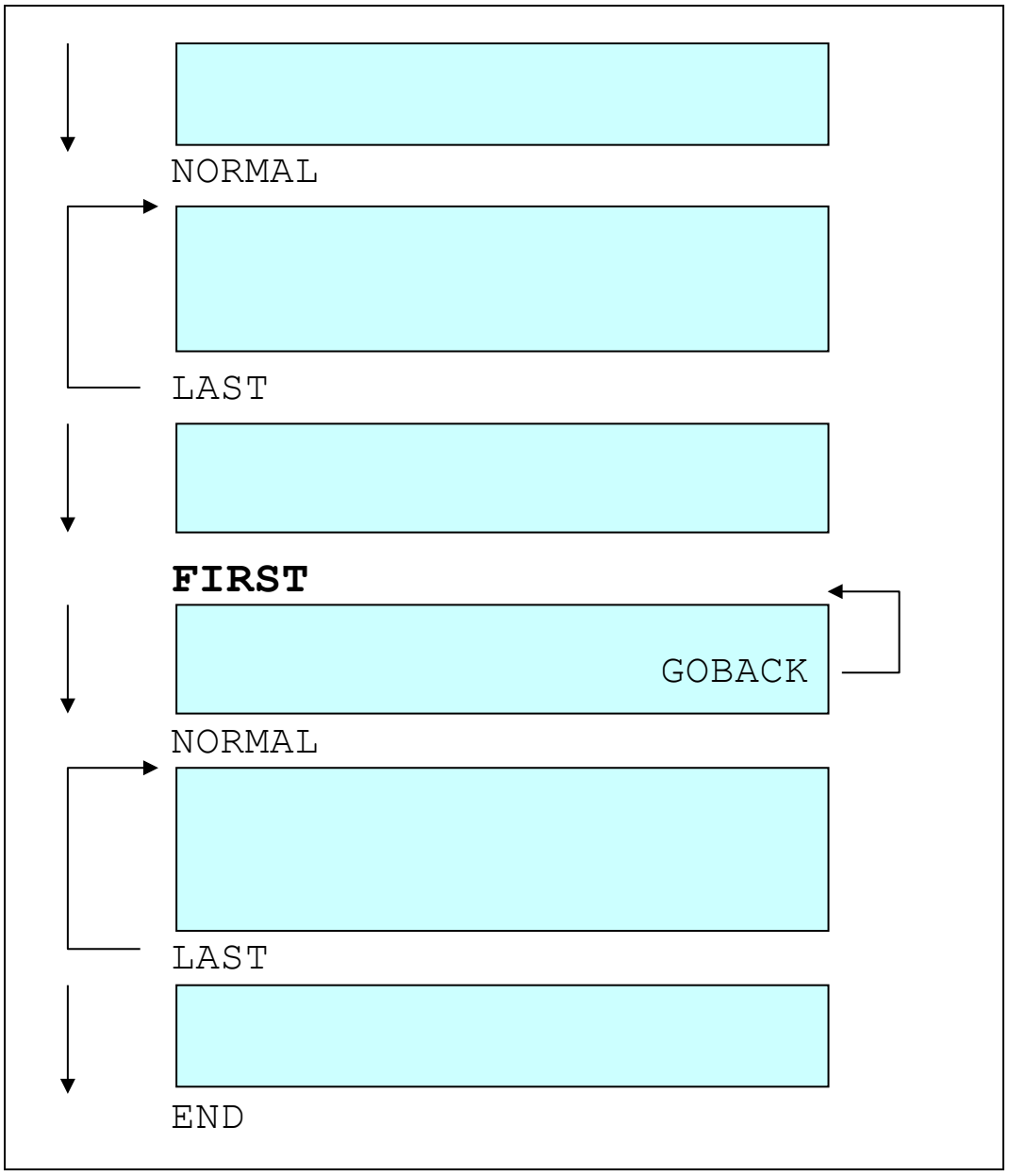


Fig. 47: The FIRST instruction allows several processing sequences

All these keywords must be placed on hierarchical level 0, i.e. all `IF`, `SUB` and `DO` blocks must be terminated by their own *xx*`END` definitions. If this is not the case and any structure blocks remain open, when `END` is reached, an error is reported:

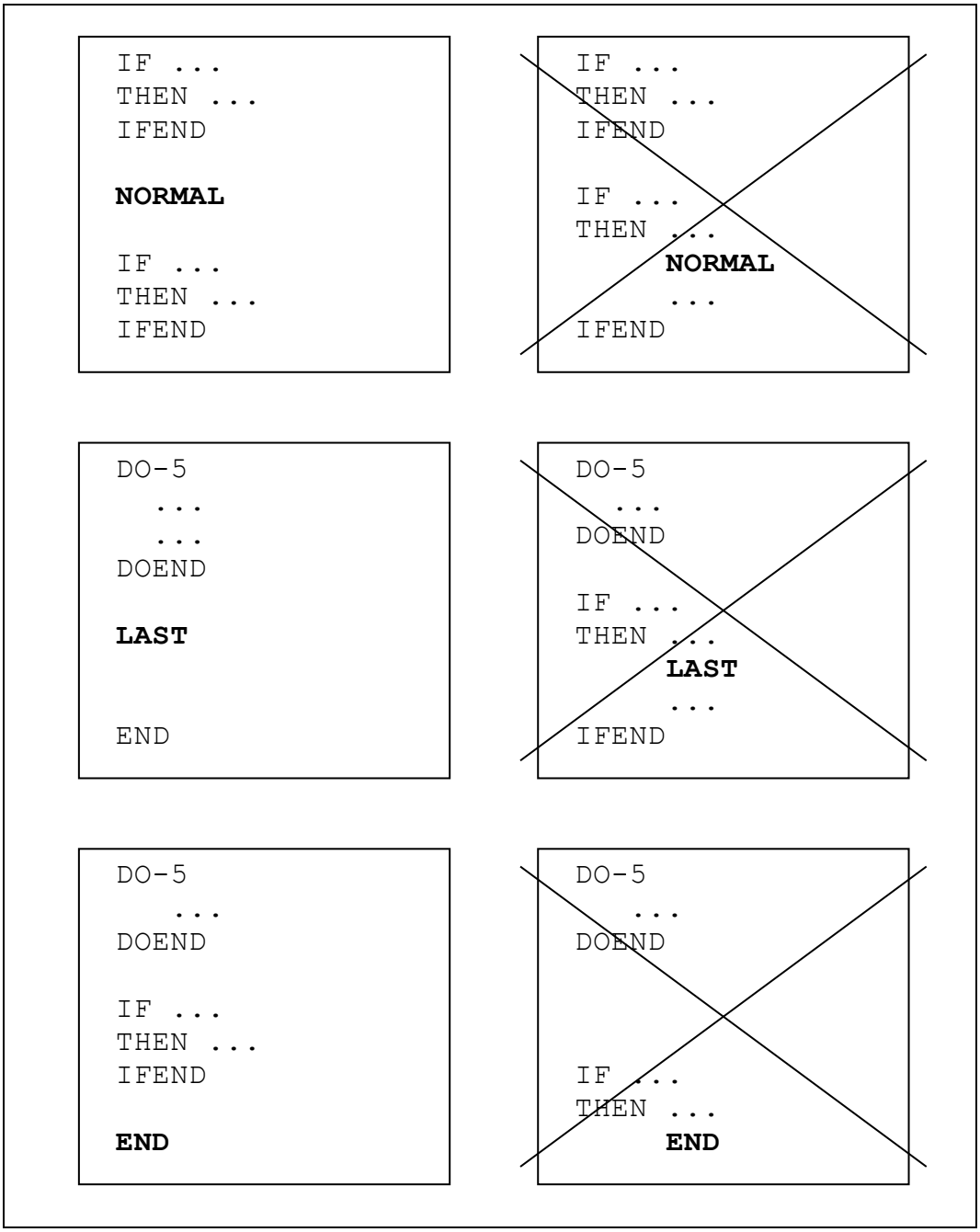


Fig. 48: Hierarchical level 0

Teamwork of NORMAL, LAST, END with Implicit Logic

If IPF or UPF are available as implicit file definitions, the NORMAL statement contains a GET command.

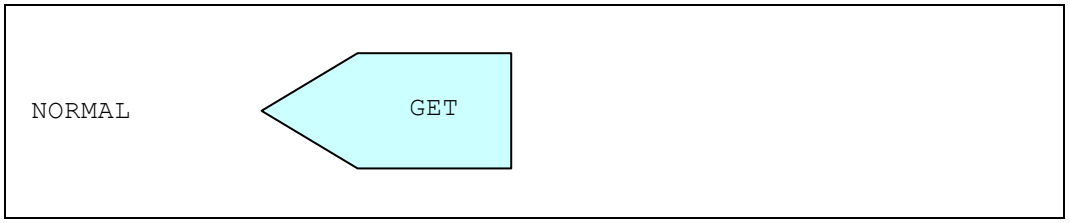


Fig. 49: Generated GET within NORMAL with implicit logic control

If OPF or UPF are available as implicit file definitions, the LAST statement or, if LAST is not specified, the END statement contain a PUT command.

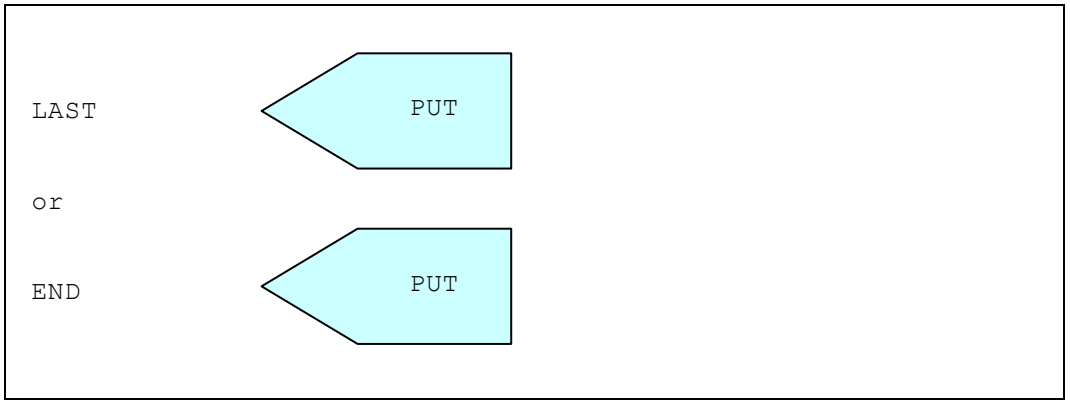


Fig. 50: Generated PUT within LAST or END with implicit logic control

Further detailed explanation is given in chapter "Internal Logic Control".

The GOSTART Instruction

This keyword instruction causes an immediate branch to the absolute beginning. The current state of the internal working storage area is unchanged.

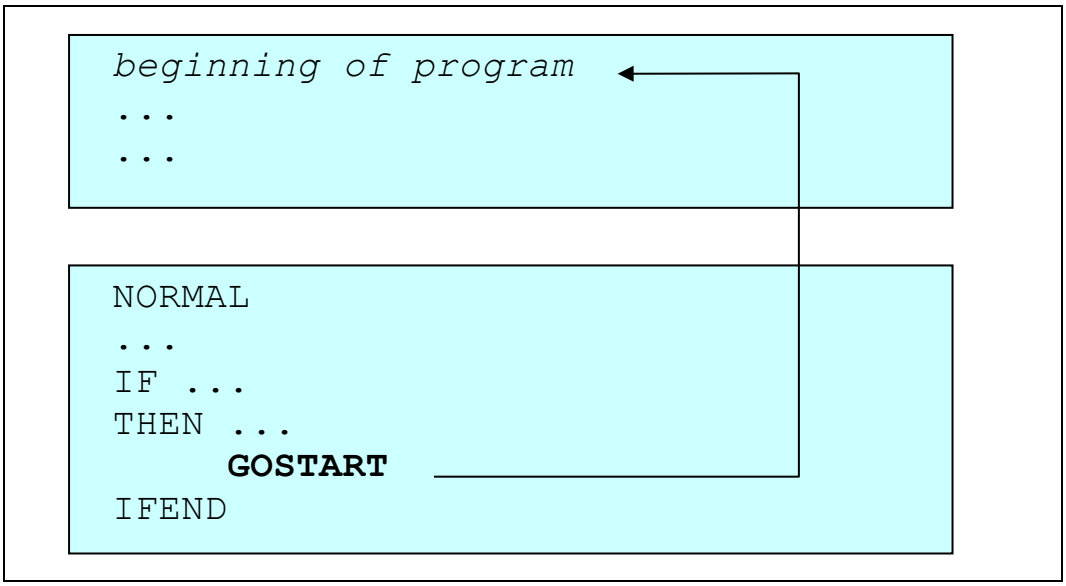


Fig. 51: The GOSTART instruction

The GOBACK Instruction

This keyword instruction causes the immediate return to the logical beginning of the processing instructions (beginning of the main processing part).
If the keyword `NORMAL` is missing, the start of processing is the return point. If the keyword `NORMAL` is present, then this is the `GOBACK` re-entry point.

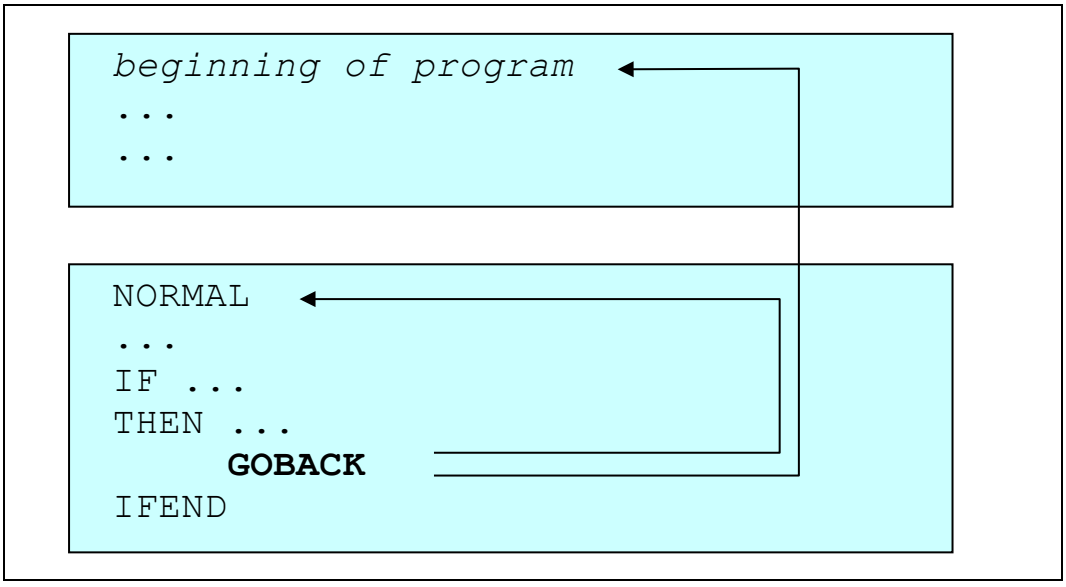


Fig. 52: The GOBACK instruction

The GOLAST Instruction

This instruction effects a **branch behind the next `LAST`** statement or, if no `LAST` has been coded, to the `END` statement.

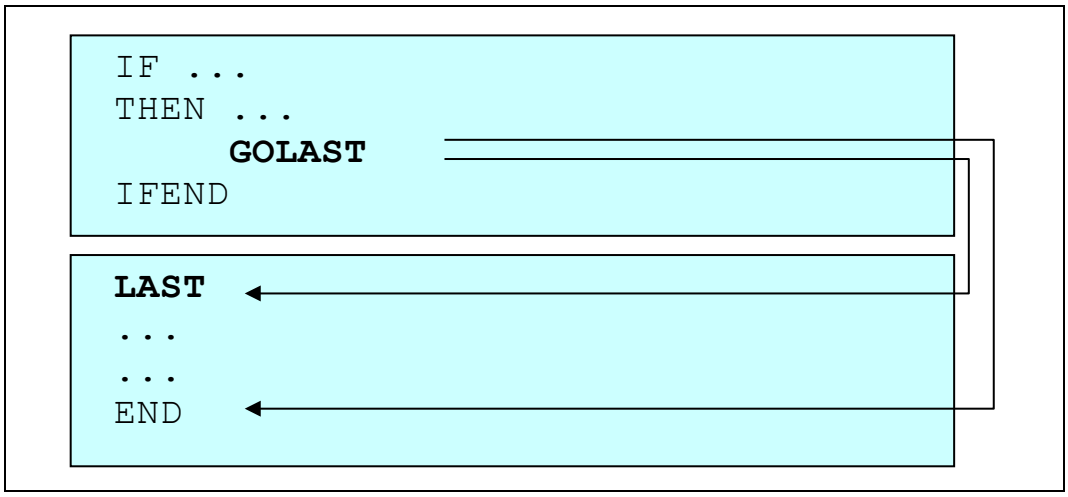


Fig. 53: The GOLAST instruction

The GO TO Instruction

This jump command can be used to jump **user defined labels**. It should be noted that the label itself must lie on hierarchical level 0. The label is an alpha-numeric symbol, of a maximum length of 8 characters, of which the first must be a letter. The label definition is terminated by a colon.

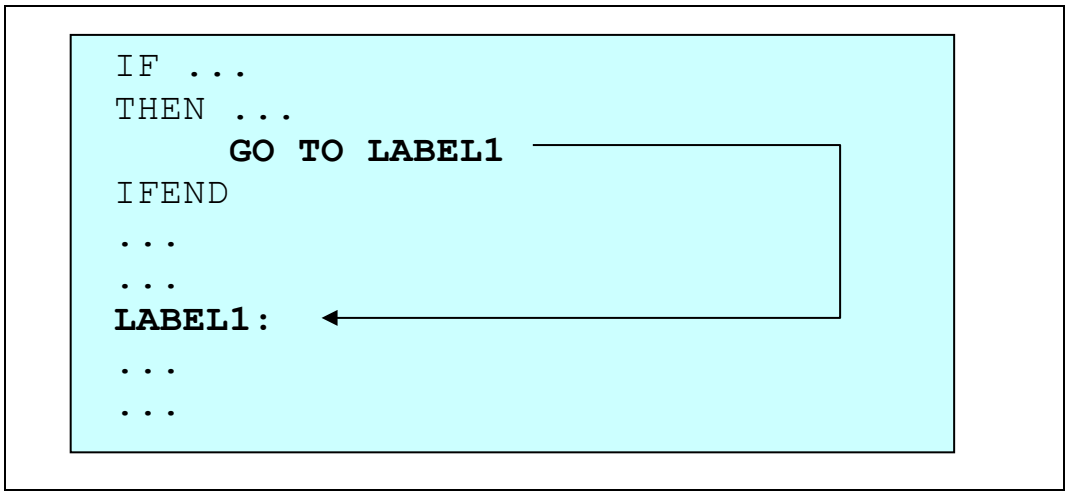


Fig. 54: The GO TO instruction

The GODUMP Instruction

This instruction causes processing to be terminated and a **dump** produced.

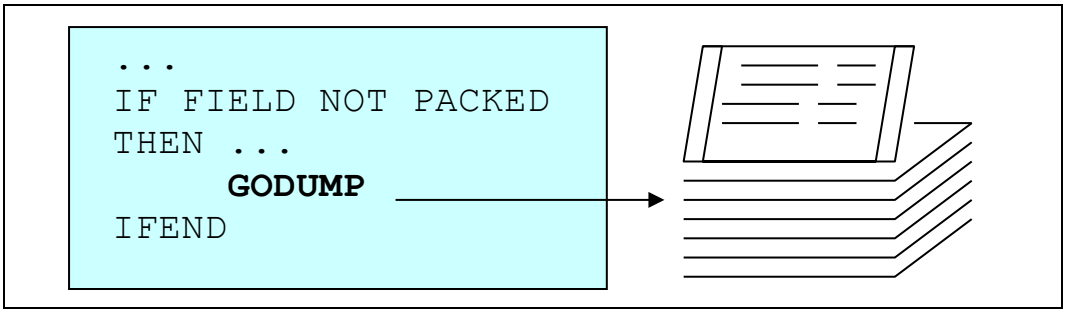


Fig. 55: The GODUMP instruction

The GOABEND [,nn] Instruction

This keyword instruction effects immediate termination of processing (CANCEL). An **abend code 1 - 4095** can be defined, default value is 12.

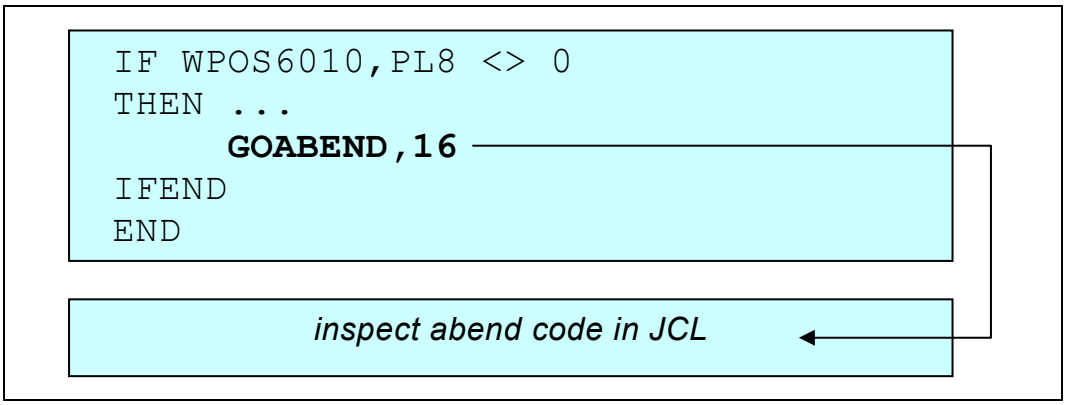


Fig. 56: The GOABEND instruction

The GOEND [,nn] Instruction

This keyword instruction causes the **end of execution**. A return code (condition code) **1 - 4095** can be defined.

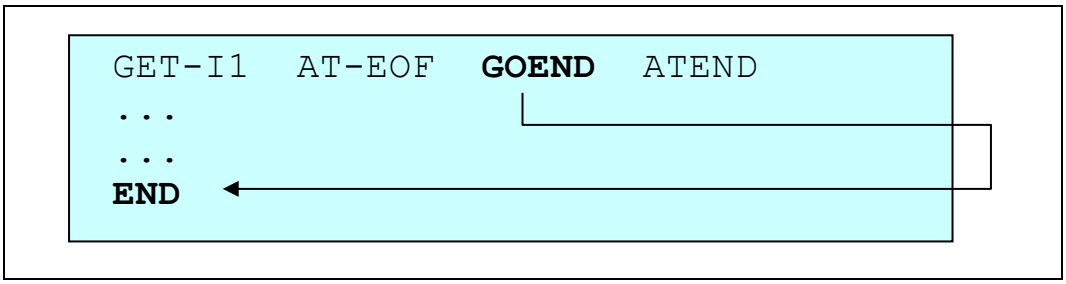


Fig. 57: The GOEND instruction

Chapter 5. Internal Logical Control

Implicit Processing Logic

Using **implicit processing logic**, several processes such as OPEN, CLOSE, GET, PUT (or W for printers) are automatically generated. The user need only define the file definition and the data handling instructions.

QPAC operates with implicit processing logic, if file identifications are **not appended by a number**:

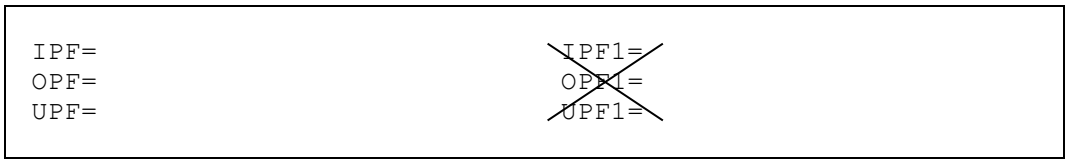


Fig. 58: Implicit processing logic with file definitions without numbers

Therefore, **only one input and/or one output file or an update file** respectively are normally defined.

Implicit processing logic is for processing single file applications in an easy manner without considering I/O operations. The user however can use I/O operations in addition to the logic generated as follows:

- Under implicit control, the **IPF** or **UPF** definition contains the read command **GET**.
- When an OPF or UPF definition is present, the **END** statement contains a **PUT** command.
- If **NORMAL** is defined, it contains the implicit **GET** command, which is thereby removed from the previously stated IPF or UPF definition.
- If **LAST** is defined, it contains the implicit **PUT** command, which is thereby removed from the later defined END statement.
- **Before a NORMAL** command, input records must be read explicitly with a **GET**.
- **Following a LAST** command, all output records must be written explicitly with a **PUT** or a **W**.
- The automatic logic for printer files includes the page skip with printing of any title lines, as well as line printing and line advancing.

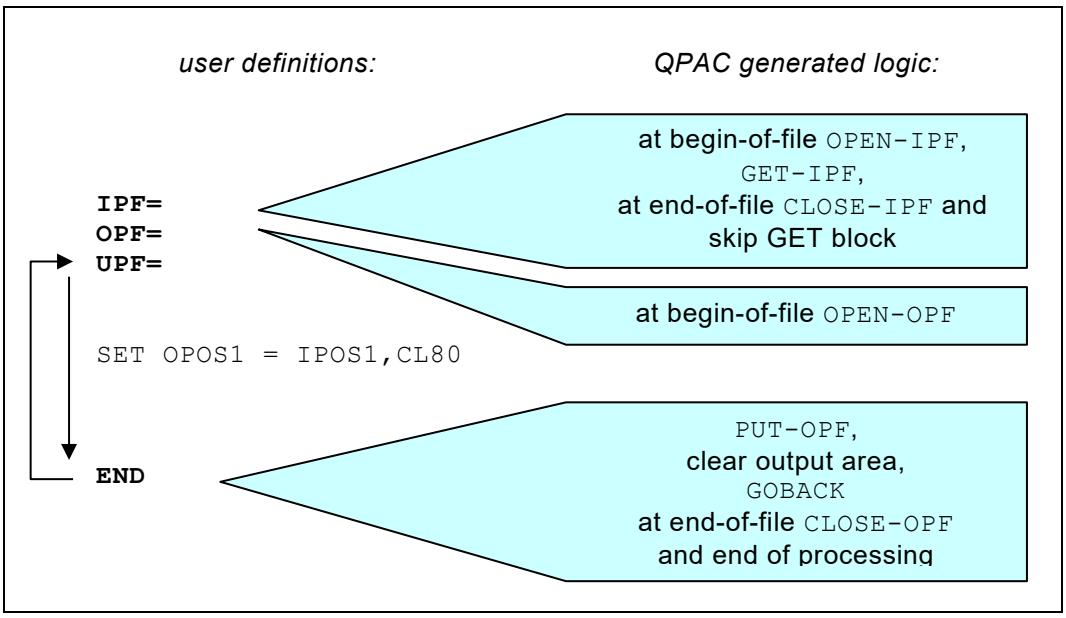


Fig. 59: Generated logic with implicit processing

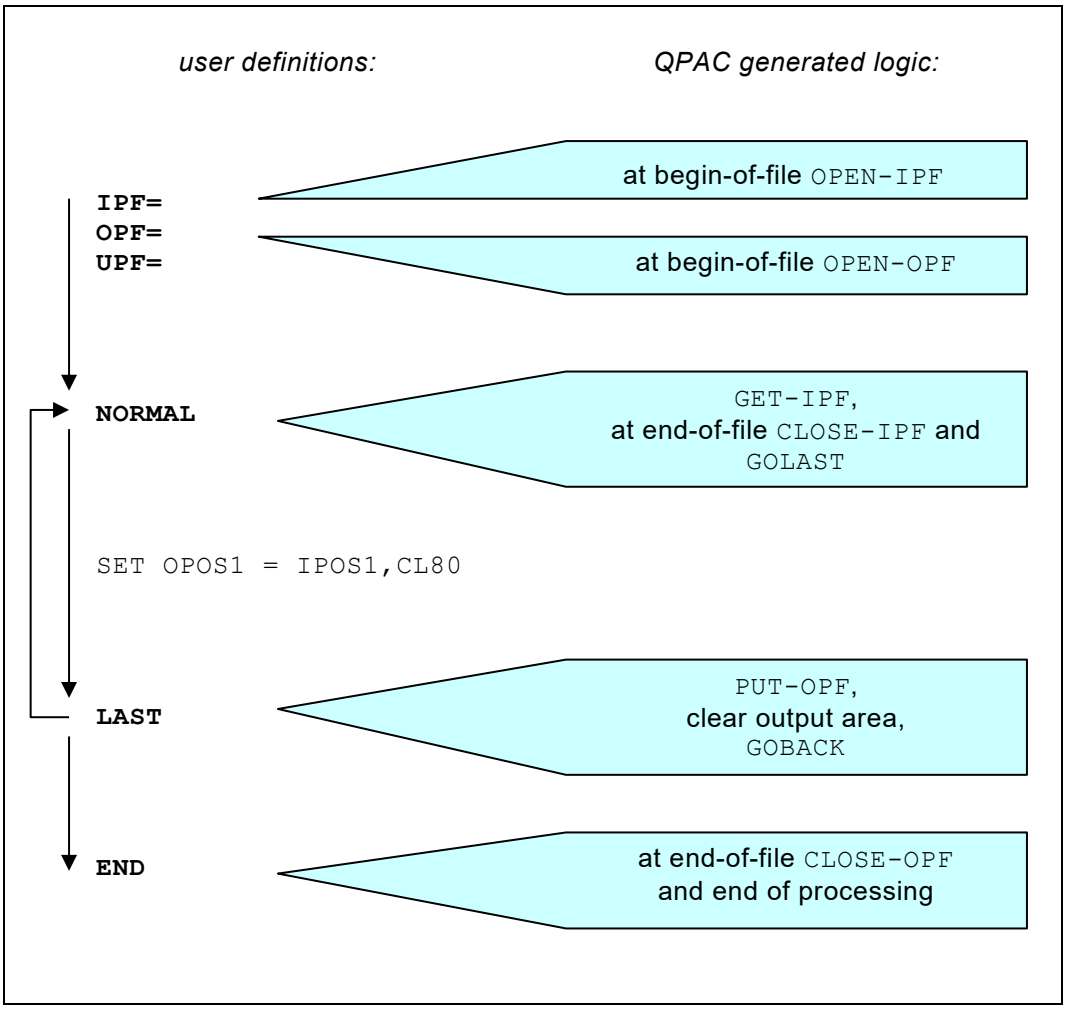


Fig. 60: Generated logic with implicit processing with NORMAL and LAST

Explicit Processing Logic

Under **explicit processing logic** the input and output commands must be explicitly defined.

With the exception of `OPEN` and `CLOSE` and printer page skips, no further I/O automatic coding is generated.

QPAC operates under explicit logical control if file identifications are **appended by a number** (from 1 to 99):

```
IPF=                                IPF1=
OPF=                                OPF1=
UPF=                                UPF1=
```

Fig. 61: Explicit processing logic with file definitions with numbers

`UPF` excludes `IPF` and `OPF` with same numbers.

Explicit control permits the processing of **several input and/or output files** at the same time. The necessary I/O statements contain the file identification as part of the format.

```
OPEN-IPF1           OPEN-I3
GET-IPF3            PUT-O2
CLOSE-I1            CLOSE-IPF3
```

Fig. 62: I/O instructions with explicit processing logic

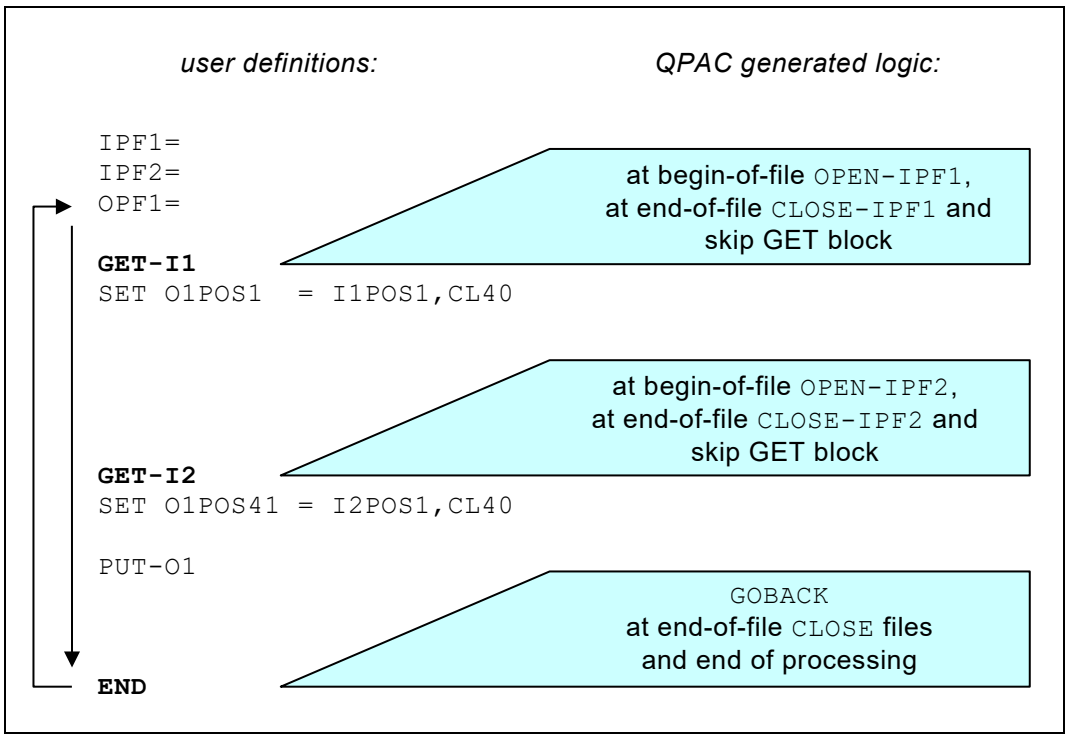


Fig. 63: Generated logic with explicit processing logic

The Get Block Concept

The following rules apply if several input files are used:

- All statements that follow a GET instruction form a logical unit called a **GET block**.

  If the execution of the GET instruction leads to EOF the whole GET block is skipped. The GET block is also skipped if a GET instruction is to be executed on a file that is already in an EOF state.

  Processing comes to an end if all **leading input files** have reached EOF status.

  If a GET command is immediately followed by an AT-EOF, this EOF block that is terminated by an ATEND, is processed when EOF state occurs. In this case the GET block will not be skipped.

- Input files are considered to be **leading** if they have GET instructions on hierarchical level 0, i.e. if they have GET instructions not exclusively in IF or DO blocks or SUBroutines.
  Input files with all GETs not on level 0 are considered to be **condition dependent**, and do not influence logical control.

- GET blocks are terminated by NORMAL, LAST, END, a new GET instruction, ELSE, IFEND, DOEND or SUBEND.

  This means, that if skipping of a GET block occurs, it results in a branch to the respective block termination point:
  In the NORMAL section to the next GET or LAST or END, in the LAST section to the next GET or END, in an IF block to the next GET or IFEND, in a DO block to the next GET or DOEND, etc.

- Within a GET block the **input addresses are implicitly assigned to the input file addressed in the GET instruction**, if the old limited **QPAC basic instruction set** is used.

  Please note that within IF's or DO's, this need not necessarily be the same file as after the respective IFEND/DOEND.
  See Appendix A: Basic Instruction Formats.

  If symbols - implicit or explicit - are used, this rule is meaningless.

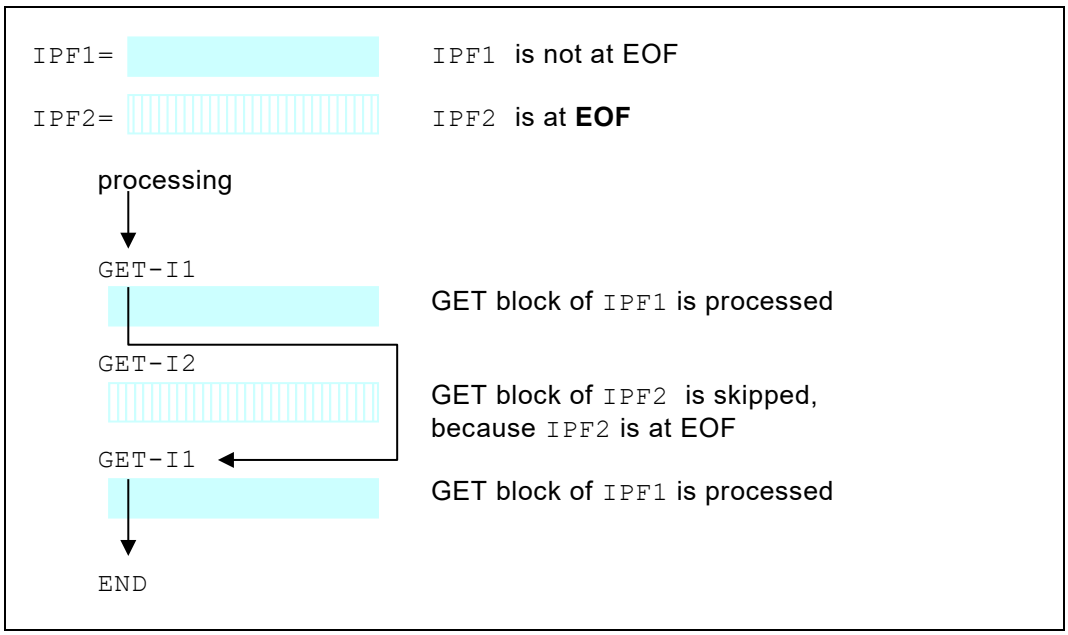


Fig. 64: EOF control without AT-EOF definition

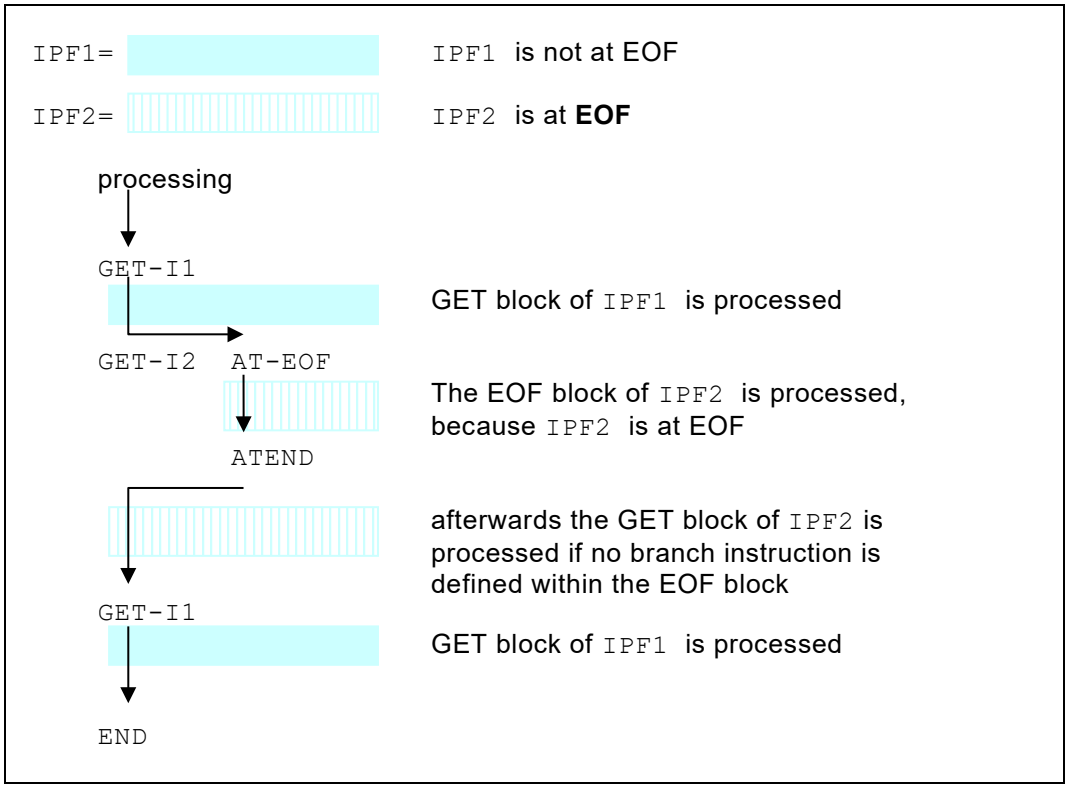


Fig. 65: EOF control with AT-EOF definition

Chapter 6. Field Definitions and Symbol Associations

Overview and Hints

The individual fields in the record structures of the datasets can be assigned to a symbol. Furthermore, in the internal **working storage** and in the **hiper space**, any number of fields with symbols can be defined, to be used as temporary storage. When naming these symbols, it should be noted that there are **reserved symbol names**, which are pre-defined internally by QPAC. The symbols used have their field length and format defined. QPAC uses this information to be able to automatically carry out any necessary field conversion for instructions.

The Internal Working Storage Area (Below the 16 MB Line)

Per default an internal working storage area of 32767 bytes is at the disposal of the QPAC user, to store or accumulate data, which can be used at a later time.
The area is accessed using addresses from `WPOS1 - WPOS32767`.
In this manual, these address values are indicated by *`wadr`*. This area can be enlarged up to a maximum of 16 MB with `PARM=WORK=`. A `WORK` area that is larger than 1 MB is allocated above the 16 MB line.

At QPAC **initialization time** the area is formatted as follows:

1	-	4999	*not preformatted*
5000	-	5999	Low Value X'00'
6000	-	6999	all 10 bytes as follows: X'000000000000000C4040
7000	-	7999	Blank X'40'
8000	-	8999	High Value X'FF'
9000	-	9999	Low Value X'00'
10000	-	32767	*not preformatted*

Fig. 66: The preformatted internal working storage area

The Internal Hiper Space (Above the 16 MB Line)

This area is accessed using the addresses `HPOS1 - HPOSnnnnn`.

The External Area (located outside the QPAC program)

This area is accessed using the addresses `XPOS1 - XPOSnnnnn`.

The description of this external area is found in Chapter 9. Subroutines and External Programs.

Implicit Symbol Association

QPAC recognises the **position symbol** and **reserved symbol names**, which have a fixed definition internally.

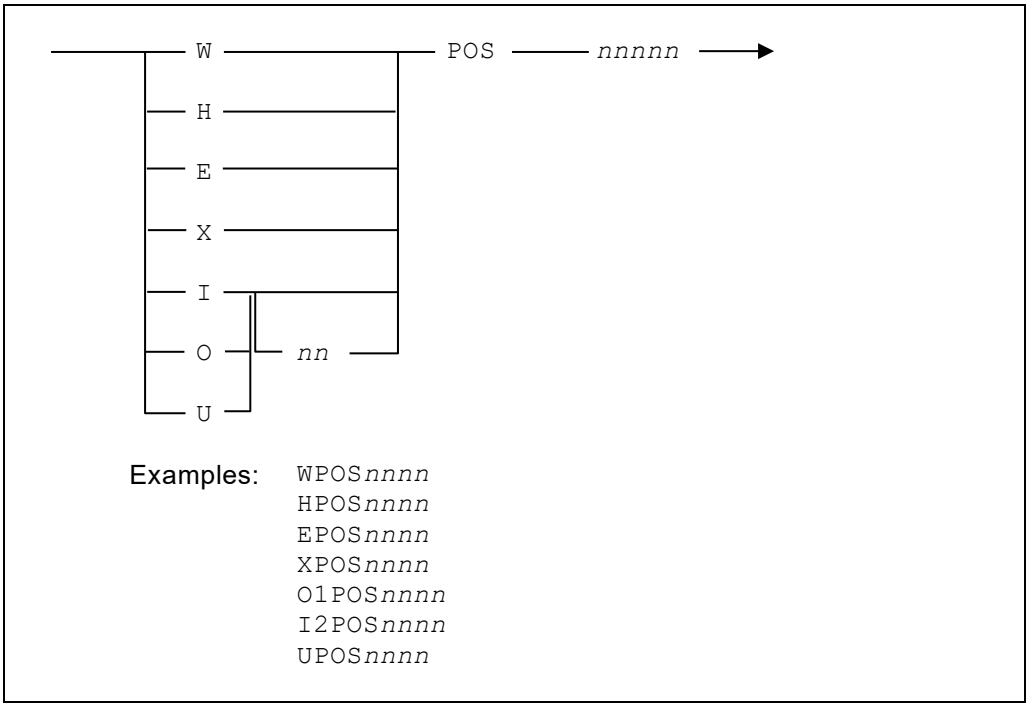


Fig. 67: I/O Instructions with explicit processing logic

Implicit position symbols are nothing other than direct position addressing in symbol form. Position addresses for the work area, the hiper space and the external area are differentiated from those for the I/O area.

Position addresses e.g. for the internal working storage area generally begin with `WPOS` followed by the position details.

Position addresses for specific file areas begin with the **short form** of the file identification preceding the word `POS`, with the position details following.

```
OPOS11      (position 11 of record area OPF=)
UPOS1       (position 1 of record area UPF=)
WPOS6000    (position 6000 of internal work area)
HPOS300     (position 300 of hiper space)
I1POS50     (position 50 of record area IPF1=)
O2POS30     (position 30 of record area OPF2=)
```

Fig. 68: Implicit position symbols

Each symbolic position address must normally have its field format and length stated, separated by a comma, within the instruction using it.

```
WPOS6000,PL8
SET O1POS102,CL2 = X'0000'
U2POS114,Z5
WPOS5120,ZL4 = O1PCNT
```

Fig. 69: Explicit definition of field formats

Explicit Symbol Association

Field definitions that are assigned to a specific area (an I/O area, the internal working storage area, the hiper space or the external area) are differentiated from single fields and literals. Single fields are not assigned to any specific area. They are allocated somewhere in the dynamic main storage and are used for simple work fields.
Definitions for a specific area are assigned a position address from 1 to *nnnnn* (end of area). Single Fields and Literals are assigned a position address 0.

Furthermore (except for I/O areas) an identifier is set after the position address which specifies the type of area or whether it is a single field or a literal.

*nnnn***W**=*symbol*	addresses the internal working storage area
*nnnn***H**=*symbol*	addresses the hyperspace
*nnnn***E**=*symbol*	addresses the EXCI communication area
nnnn=*symbol*	addresses the I/O area of the preceding file definition
0S=*symbol*,*format-length*	addresses a single field
0L=*symbol*,*value*	addresses a literal

Fig. 70: Explicit symbol types

Basic Format of Explicit Symbol Association for Single Fields and Literals

0L=*symbol*,*format-length*,*value*	literals
0S=*symbol*, ─┬─ *format-length*[,*value*]	single fields
├─ ' '	character literals
├─ C' '	
├─ CL*n*' '	
├─ V*n*' '	variable character
├─ VL*n*' '	
├─ X' '	hexadecimal literal
├─ XL*n*' '	
├─ Z'*nnnnn*'	zoned decimal literal
├─ ZL*n*'*nnn*'	
├─ P'*nnnnn*'	packed decimal literal
├─ PL*n*'*nnn*'	
├─ BL*n*'*nnn*'	binary literal
└─ B'........'	bit literal
B'........'	is a special format as a bit literal if no length attribute is specified. It has a fixed length of 8 bits (1 byte) and can only contain zeroes and ones.

Fig. 71: Basic format explicit symbol association for single fields and literals

Basic Format of Explicit Symbol Association for Areas

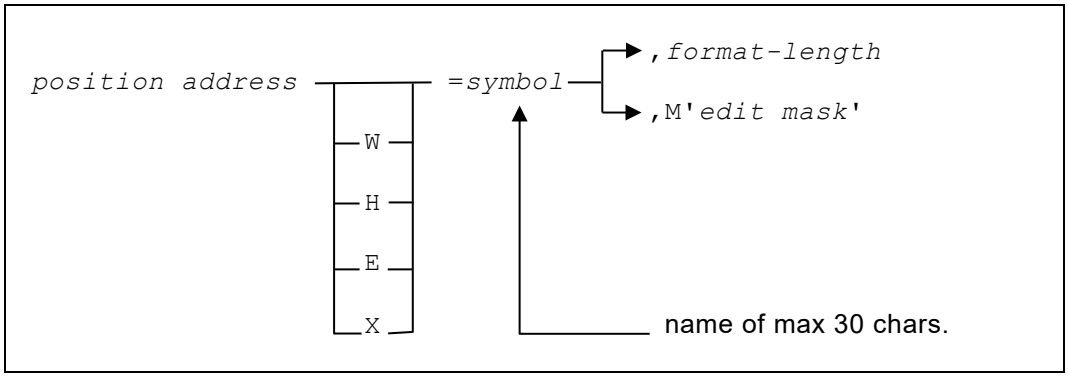


Fig. 72: Basic format explicit symbol association for areas

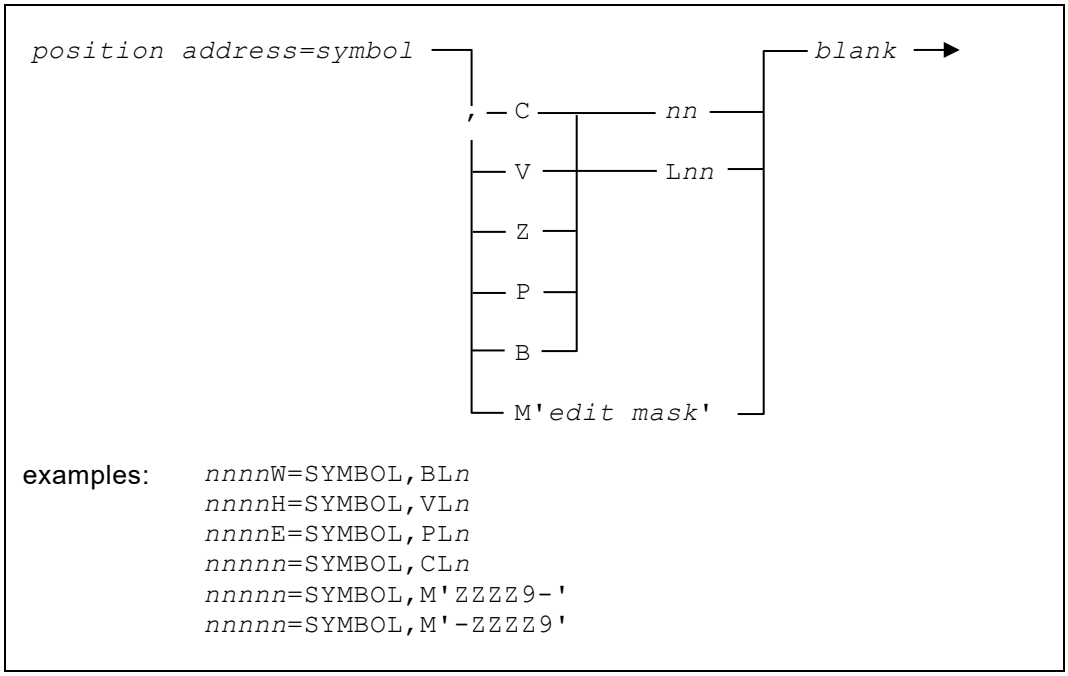


Fig. 73: Diagram explicit symbol definition

The position address of the symbol (may be accompanied by `W` for the internal working storage, `H` for the hiper space or `E` for the external EXCI area) is defined to the left of the equals sign, the symbol itself on the right. Blanks are **not** allowed either to the left or right of the equals sign.
A length and field format can be defined after the symbol. The length is defined in **bytes**.

Five field formats are supported:

, CL*n*	or	, C*n*	character	(1 – 9999999)
, VL*n*	or	, V*n*	variable character	(2 – 32767)
, ZL*n*	or	, Z*n*	zoned decimal	(1 – 20)
, PL*n*	or	, P*n*	packed	(1 – 16)
, BL*n*	or	, B*n*	binary	(1 – 8)

Fig. 74: Field formats for conversion instructions

Important with variable character fields (`Vn`):
Every variable character field is automatically preceded by a 2 bytes binary field.
This length field has the same symbol name but with an appended # sign.
V means a character attribute and cannot be defined as a literal.

```
    1=RECORDTYPE,CL1
 5010=ARTICLE,CL30          (W is assumed if value is outside the I/O
                             area size)
5010W=ARTICLE,CL30
 7100=QUANTITY,P5           (W is assumed)

   1H=HIPERFIELD,CL5        (hiper space)
   1E=EXCIFIELD,BL2         (CICS EXCI communication)
   1X=EXTERNALFIELD,CL100   (QPAC as a sub routine)
```

Fig. 75: Sample explicit symbols with field formats

Instead of a field format, an edit mask can be defined for a numeric value, the mask defining the length of the value. The number of numerals is defined by **9**, and where leading zero suppression is required, a **Z** is used instead of a 9.
The mask can be terminated with a – or a + character.
A **+** character remains unchanged when the value is positive and is replaced by a – character when the value is negative.
A **–** character is replaced by blank when the value is positive and remains unchanged when the value is negative.
A **–** character can also be defined on the left side of the mask. It then becomes a sliding negative sign.
The whole mask is contained within apostrophes and a leading attribute `M`.

```
10=TOTAL,M'ZZ.ZZ9,99-'
10=TOTAL,M'-ZZ.ZZ9,99'
80=ORDER_TIME,M'99:99:99'
```

Fig. 76: Numeric explicit symbols with edit masks

Symbol names can be alphanumeric. They must begin with an alpha character and can contain the $ # @ _ special characters.

Remember the existence of **reserved symbol names**.

Simplified Format of Explicit Symbol Association

A simplified form of field structures can be defined.
Fields without a position definition can be directly appended to a preceding position definition with a leading equal sign.
The pseudo symbol `FILLER` can be used as a place holder.

```
1=RECORDTYPE,CL1
 =ARTICLE,CL30
 =QUANTITY,PL3
 =PRICE,PL5
 =FILLER,CL3
 =LOCATION,CL1
```

Fig. 77: Simplified format of explicit symbol association

Redefines in Structures

Within a structure an individual field may be redefined by prefixing the equals sign (without a position definition) with an `R` (`R=`).
The Redefine (`R=`) always starts at the beginning of the preceding field.

Work area structure:

```
100W=PACKED_FIELD,PL5
    =BINARY_FIELD,BL4
    =DATE_FIELD,CL8
   R=YEAR,CL4
    =MONTH,CL2
    =DAY,CL2
```

Fig. 78: Redefining work area structures

Single field structure:

```
0S=FIELD,CL14
 R=FIELD1,CL1
  =FIELD2,CL1
  =FIELD3,CL8
 R=FIELD4,CL1
  =FILLER,CL6
  =FIELD5,CL1

0S=FIELDZ,CL5
```

Fig. 79: Redefining single field structures

Literal structure:

```
0L=LITERAL,CL10'1234567890'
 R=LITERAL1,CL1
  =FILLER,CL3
  =LITERALS,CL6
 R=LITERAL5,CL1
  =FILLER,CL4
  =LITERAL0,CL1
```

Fig. 80: Redefining literal structures

Based structure:

```
0B=BASE,PTR
1B=BASE0,CL10
 R=BASE1,ZL5
  =BASE2,ZL5
 R=BASE3,ZL1
  =FILLER,CL3
  =BASE5,ZL1
```

Fig. 81: Redefining based structures

I/O structure:

```
IPF=VSAM
 1=IOAREA,CL10
 R=IOARE1,ZL5
  =IOARE2,ZL5
11=IOARER,CL65
```

Fig. 82: Redefining I/O structures

Explicit Symbols for File Definitions

The association of the position address to the relevant record area of the file definition must comply with the following rules:

a) Following a file definition, the symbols will be associated with this input/output area:

```
IPF=SQ
  1=INPUTREC,CL80
...
OPF=PR
  1=PRINTAREA,CL132
...
```

Fig. 83: File area association with explicit symbols

b) Following an input operation, the symbol positions are associated with the area of the file referred to by the operation.

c) The association of symbols to the internal working storage area can occur anywhere, as the association criteria are independent of file definitions. They are therefore not subject to a structure block hierarchy. Position addresses starting with 5000 are only assigned to the internal work area if the I/O area of the preceding file definition is smaller.

The allocation rule is also influenced by the hierarchical structure, i.e. after a structure block end, the allocation in effect before beginning that structure block is resumed.

Fig. 84: Hierarchical structure of symbol association

Explicit COBOL and PL/I Record Structure Assignment

Catalogued COBOL and PL/I record structures can be loaded from a source library and be associated with a file definition or work area position. The field names are converted to QPAC symbol names (- signs are converted to _ signs).
Initial values and edit masks are ignored.
Based definitions within PL/I copy books are supported with some restrictions.

```
position_address=COBREC=bookname
position_address=PLIREC=bookname
```

Fig. 85: Diagram explicit COBOL and PL/I record structure association

position_address can be a record position or a work area position (*nn*W=).
bookname is the name of a cataloged COBOL or PL/I record structure.
COBREC= and PLIREC= are keywords.

```
IPF=VSAM
  1=COBREC=F4000RD
...

7001W=COBREC=F4200RD
```

Fig. 86: Import of an existing COBOL record structure

Under z/OS these copy books are read in by the PDS DD name //QPACCOPY.

Important with PL/I books:
With BASE(ADDR(SYMBOL)) an additional # sign is attached to the symbol, defined as a PTR field.

BASED Structures

In QPAC based structures can be defined to easily perform table processing without indexing the individual table elements with index registers. In the assembler language they correspond to so called DSECTs (dummy sections).

A based structure must be preceded by a pointer field whose content must contain the actual storage address. The fields of the following structure are addressed during execution in relation to the pointer. The pointer content may be modified by adding or subtracting values.

The content of the pointer field is initialized by a direct value or by another field's address. For that case a `SET` special value instruction `=ADR` is provided which causes the address of the sending field to be loaded into the receiving pointer.

```
0B=POINTER,PTR
1B=ADDRESSFIELD1,BL4
  =FIELD2,PL3
8B=FIELD3,CL5
...
```

Fig. 87: Basic format of based structures

Between position address and equals sign (=) the identifier "`B`" is set.
It identifies a based structure.

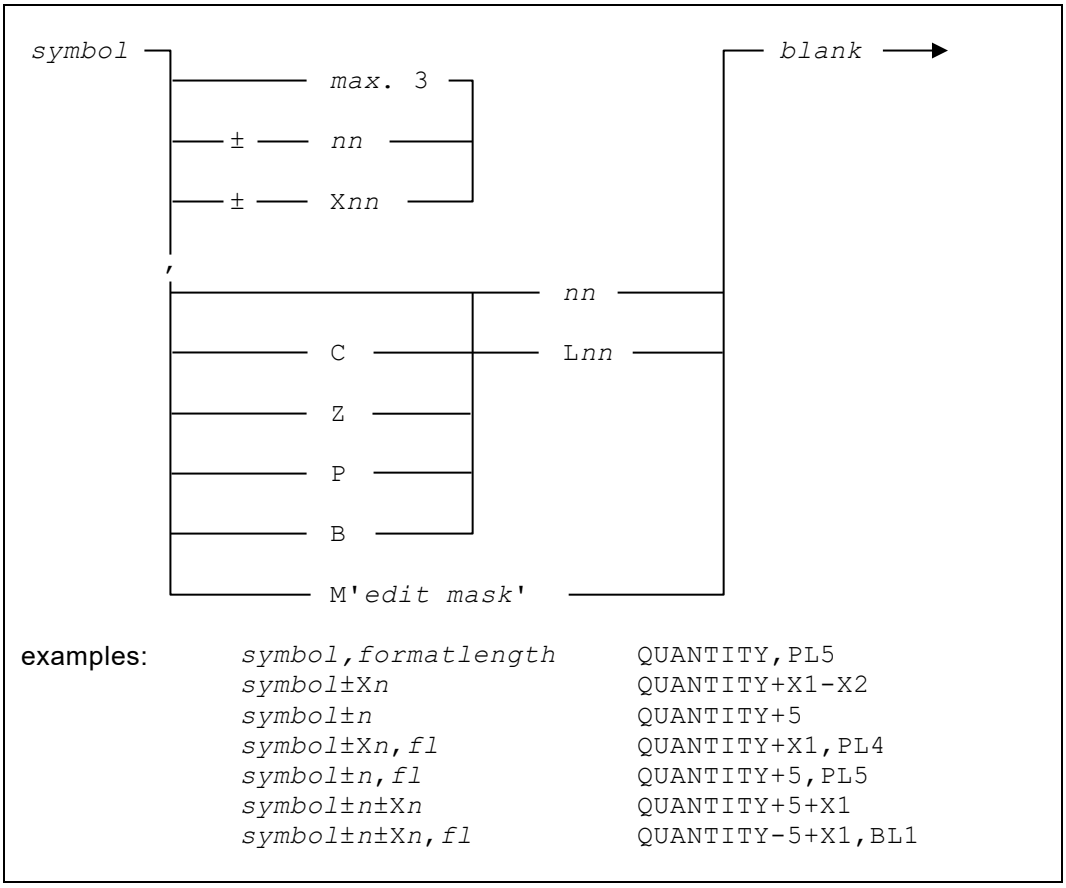
The head of a based structure is defined by the accompanying pointer field.
Internally a pointer field is represented by a 4 bytes binary field.

```
SET POINTER =ADR WPOS10001

or

SET POINTER = nnnnn
```

Fig. 88: Loading the pointer field

```
SET POINTER = + nnnnn
SET POINTER = - nnnnn
```

Fig. 89: Moving a structure



examples:		
	symbol,formatlength	QUANTITY,PL5
	symbol±X*n*	QUANTITY+X1-X2
	symbol±*n*	QUANTITY+5
	symbol±X*n*,*fl*	QUANTITY+X1,PL4
	symbol±*n*,*fl*	QUANTITY+5,PL5
	symbol±*n*±X*n*	QUANTITY+5+X1
	symbol±*n*±X*n*,*fl*	QUANTITY-5+X1,BL1

Fig. 90: Diagram symbolic indexed addressing

An address modification can be made by defining an absolute value, or an index register for the dynamic form.
Indexed addressing occurs when an **index register** is **appended** to the field symbol with a plus or minus operator.
A maximum of 3 index registers may be appended.

```
SYMBOL+X1        SYMBOL-X9        SYMBOL+X6
SYMBOL-X1+X2     SYMBOL-X9-X8     SYMBOL-X6+X7+X8
```

Fig. 91: Indexed addressing by appending index registers

An implicit format length can be temporarily overwritten by an explicit definition.

```
7001=DATE_TABLE,CL72                    *. 12 MONTHS
7001=INCOMING_DATE,ZL6                  *. YYMMDD
X1=0                                    *. INIT OFFSET X1
DO-12
   SET INCOMING_DATE+X1,ZL2 = 92        *. INIT YEAR
   X1+6                                 *. NEXT YY
DOEND
```

Fig. 92: Explicit length specification overrides implicit length definition

Reserved Field Symbols

Reserved symbol names are internal fields or registers available to the user, without the user having to explicitly define them.

The following field symbols are predefined with respect to format and length and are reserved names within QPAC-Batch.

.. Some field symbols contain the corresponding file identifier in their symbol name represented by two dots (e.g. ..KEY). Replace them by any valid short form file identifier.

Reserved Field Symbols by Group

symbol name	description		format length	B	O
X*n*	Index registers (*n* = 1-99)		BL4	B	O
ACCU*n*	accumulators (*n* = 0 - 99) they are fixed in the work area positions 6000 - 6990		PL8	B	O
FILLER	place holder for explicit symbol association			B	O
GPOS*nnnn*	global work area position (*nnnn* = 1 - 32767)				O
WPOS*nnnn*	internal work area position (1 - *nnnn*)			B	O
HPOS*nnnn*	internal hiper space position (1 - *nnnn*)			B	
EPOS*nnnn*	EXCI communication area position (1 – *nnnn*)			B	
XPOS*nnnn*	external area position (1 – *nnnn*)			B	
I[*n*]POS*nnnn*	record area position *nnnn* of IPF[*n*]			B	
O[*n*]POS*nnnn*	record area position *nnnn* of OPF[*n*]			B	
U[*n*]POS*nnnn*	record area position *nnnn* of UPF[*n*]			B	
D*n*POS*nnnn*	record area position *nnnn* of DS*n*, DB*n*				O
T*n*POS*nnnn*	record area position *nnnn* of TD*n*, TS*n*				O
S*n*POS*nnn*	record area position *nnn* of SPO*n*				O
DIVREM	division remainder field		PL8	B	O
DB2COMMIT	DB2 auto commit counter		PL8	B	
USDATE	system date US format	'MM/DD/CCYY'	CL10	B	O
EDATE	system date EURO format	'DD.MM.CCYY'	CL10	B	O
ISODATE	system date ISO standard	'CCYY-MM-DD'	CL10	B	O
DATE	system date	'DD.MM.YY'	CL8	B	O
DAY	day of system date	DD	ZL2	B	O
MONTH	month of system date	MM	ZL2	B	O
YEAR	year of system date	YY	ZL2	B	O
TIME	QPAC start time	'HH:MM:SS'	CL8	B	O
HOUR	hour of start time	HH	ZL2	B	O
MINUTE	minute of start time	MM	ZL2	B	O
SECOND	second of start time	SS	ZL2	B	O
CDATE	actual current date	DDMMYY	PL4	B	O
CTIME	actual current time	HHMMSS	PL4	B	O
CDTIME	current date and time (Attention: only valid for basic instruction format)	DDMMYYHHMMSS	2xPL4	B	O

symbol name, description, format length					
CCDATE	actual century date	CCYYMMDD	PL5	B	O
CCYEAR	actual century year	CCYY	ZL4	B	O
RC	return code	special register	BL4	B	O
FC	function code	special register	BL4	B	O
IV	interval value	special register	BL4	B	O
EPARM	external parameter area	max.	CL100	B	
EPARML	length of external parameter value		BL2	B	
FUNCMSG	function return message		CL72	B	O
USERID	QPAC-Online user identification		CL16		O
PRGNAME	QPAC-Online program name		CL16		O
MAPNAME	QPAC-Online active map name		CL16		O
APPLID	VTAM CICS application identification		CL8		O
NETNAME	VTAM terminal netname		CL8		O
CUSERID	CICS user identification		CL8		O
CICSID	CICS system identification		CL4		O
TERMID	CICS terminal identification		CL4		O
OPERID	CICS terminal operator identification		CL3		O
CURSOR	cursor position after GET MAP		BL4		O
MAXROW	maximum rows on screen		BL2		O
MAXCOL	maximum columns on screen		BL2		O
JOBNAME	JCL job name		CL8	B	
STEPNAME	z/OS JCL step name		CL8	B	
JOBNUM	z/OS JCL job number		ZL5	B	
JOBACTELNO	z/OS Job accounting element no		BL4	B	
JOBACTINFO	z/OS job accounting info		CL144	B	
JOBCLASS	z/OS job class from JOB statement CLASS=		CL1	B	
JOBCLASSLG	z/OS long job class from JOB statement CLASS= or //*MAIN CLASS=		CL1	B	
JOBPROGRNM	z/OS programmer name from JOB statement		CL20	B	
JOBSTIME	z/OS job start time hhmmss		PL4	B	
SYSNAME	z/OS system name / SYSID		CL8	B	
RACFUSER	z/OS RACF user id		CL8	B	
CALDRDATE	CALENDAR date format CCYYMMDD		ZL6	B	O
CALDRWKNR	CALENDAR number of week (01-53)		ZL2	B	O
CALDRWKDY	CALENDAR day of week (1-7)		ZL1	B	O
CALDRWARN	CALENDAR warning for day adjustment at end of month		CL1	B	O
CALDRTXMT	CALENDAR name of month (January – December)		CL10	B	O
CALDRTXWD	CALENDAR name of day (Monday – Sunday)		CL10	B	O
C.FCNT	EXECSQL fetched counter		PL8	B	O
..GCNT	GET sequential read counter		PL8	B	

symbol name, description, format length				
..RCNT	READ random read counter	PL8	B	
	MAP info retrieval receive counter			O
..PCNT	PUT record counter	PL8	B	
	page counter (print files)	PL4	B	O
..LCNT	line counter per page (print files)	PL2	B	O
..MAXLCNT	maximum number of lines per page (print files)	PL2	B	O
..UCNT	updated records counter UPF file	PL8	B	
..ACNT	added records counter UPF file	PL8	B	
..DCNT	deleted records counter UPF file	PL8	B	
..SCNT	MAP info retrieval send counter	PL8		O
..FORM	printer FORM name	CL8	B	O
..FN	info retrieval file name	CL8	B	O
..FT	info retrieval VSAM file type KS,ES,RR	CL2	B	O
..RL	info retrieval defined maximum record length	BL2	B	O
..BL	info retrieval defined maximum block length	BL2	B	O
..KL	info retrieval defined key length	BL2	B	O
..KP	info retrieval defined key position	BL2	B	O
..DSN	info retrieval data set name	CL44	B	
..DDN	info retrieval DD name	CL8	B	
	DD name for dynamic allocation			O
..DBN	FCA DL/I data base name	CL8	B	
	DB name for dynamic allocation			O
..PSB	FCA DL/I PSB name	CL8	B	
	PSB name for dynamic allocation			O
..SEGNM	FCA DL/I segment name	CL8	B	O
..LEV	FCA DL/I segment level	ZL2	B	O
..SSA*n*	FCA DL/I SSA fields (*n* = 1 - 8)	CL256	B	O
..SSAN	FCA DL/I number of SSA fields in use	BL1	B	O
..SEGLENG	FCA DL/I segment length (without DBCTL)	BL4	B	
..KFBAREA	FCA DL/I key feedback area	CL256	B	
..KFBALENG	FCA DL/I key feedback area length	BL4	B	
..PCB	PCB number for dynamic allocation	BL4		O
..RTN	root segment name for dynamic allocation	CL8		O
..KFN	key field name for dynamic allocation	CL8		O
..KEY	FCA key field	CL236	B	O
..RBA	FCA relative byte address	BL4	B	O
..RRN	FCA relative record number	BL4	B	O
..LENG	FCA record length	BL4	B	O
..RC	FCA return code (VSAM, DL/I, queues, ...)	CL2	B	O
..RC1	FCA return code position 1	CL1	B	O
..RC2	FCA return code position 2	CL1	B	O
..SQLCODE	FCA SQL return code	BL4	B	O
..WHERE	*for internal use only (DB2 support)*			
..ROWLENG	FCA SQL row length	BL4	B	O
..TBN	FCA SQL table name	CL18	B	O
..QNM	FCA queue name	CL8		O
..ITEM	FCA queue item	BL2		O

symbol name, description, format length

..MEMDIRVV	FCA version from statistic record	PDS	BL1	B
..MEMDIRMM	FCA modification from statistic record	PDS	BL1	B
..MEMDIRCRDT	FCA creation date from statistic record	PDS	PL5	B
..MEMDIRCHDT	FCA changed date from statistic record	PDS	PL5	B
..MEMDIRTIME	FCA last changed time from stat. record	PDS	PL4	B
..MEMDIRSIZE	FCA number of records in member	PDS	BL2	B
..MEMDIRINIT	FCA number of initial records	PDS	BL2	B
..MEMDIRUSER	FCA User Id	PDS	CL7	B
..MEMNM	FCA member name	PDS	CL8	B
..STOWID	FCA STOW identification	PDS	CL1	B
..STOWVV	FCA STOW statistic version	PDS	BL1	B
..STOWMM	FCA STOW statistic modification	PDS	BL1	B
..STOWUSER	FCA STOW statistic user id	PDS	CL7	B

..DSN	data set name for dyn. allocation ALLOC		CL44	B
..DDN	DD name for ALLOC		CL8	B
..SDISP	status disp for ALLOC:	DISP=SHR	CL1	B
..NDISP	normal disp for ALLOC:	DISP=(,CATLG...	CL1	B
..CDISP	cancel disp for ALLOC:	DISP=(...,...,DELETE	CL1	B
..TYPSP	type of space for ALLOC:	SPACE=(CYL,...	CL1	B
..PRISP	primary space for ALLOC:	SPACE=(...,(*nn*,...	BL2	B
..SECSP	sec. space for ALLOC:	SPACE=(...,(...,*nn*	BL2	B
..DIRBL	directory blks for ALLOC:	SPACE=(...,(...,...,*nn*	BL2	B
..RLSE	release space for ALLOC:	Yes oder No	CL1	B
..UNIT	unit name for ALLOC:	SYSDA	CL8	B
..DIR	option DIR for PDS for ALLOC 'D'		CL1	B
..BWD	option BWD for VSAM for ALLOC 'B'		CL1	B
..DSORG	data set organization for ALLOC:		CL2	B
..RECFM	record format for ALLOC:		CL3	B
..VOLID	volume serial ident for ALLOC:		CL6	B
..LABEL	tape label NL or SL for ALLOC:		CL2	B
..DCLAS	data class name for ALLOC:	SMS only	CL8	B
..MCLAS	management class for ALLOC:	SMS only	CL8	B
..SCLAS	storage class for ALLOC:	SMS only	CL8	B
..MN	member name for generic selection ALLOC		CL8	B

..SCATNM	selected catalog name	CL44	B
..SDSNM	selected generic dataset name	CL44	B

W.AT	WEB AT position symbol	CL16	O
W.BOOKMARK	WEB bookmark symbol	CL16	O
W.CADDRLENGTH	WEB client address length	BL4	O
W.CLIENTADDR	WEB client address	CL*n*	O
W.CLIENTCODEPAGE	WEB client code page value	CL40	O
W.CLIENTNAME	WEB client name	CL*n*	O

symbol name	description	format length		
W.CNAMELENGTH	WEB client name length	BL4		O
W.DATAONLY	WEB data only flag	CL1		O
W.DELIMITER	WEB	CL1		O
W.DOCSIZE	WEB document size	BL4		O
W.DOCTOKEN	WEB document token	CL16		O
W.FNAMELENGTH	WEB form field value length	BL4		O
W.FORMFIELD	WEB form field area	CL*n*		O
W.FROMDOC	WEB From document value	CL16		O
W.HNAMELENGTH	WEB HTTP header value length	BL4		O
W.HOSTECODEPAGE	WEB host code page value	CL8		O
W.HTTPHEADER	WEB HTTP header area	CL*n*		O
W.LENGTH	WEB current value length	BL4		O
W.PORTNUMBER	WEB port number	CL5		O
W.PORTNUMNU	WEB port number binary	BL4		O
W.RESP2	WEB CICS reason code	BL2		O
W.SADDRLENGTH	WEB server address length	BL4		O
W.SERVERADDR	WEB server address	CL*n*		O
W.SERVERNAME	WEB server name	CL*n*		O
W.SNAMELENGTH	WEB server name length	BL4		O
W.SYMBOL	WEB symbol in symbol table	CL32		O
W.TCPIPSERVICE	WEB TCP/IP service info	CL48		O
W.TEMPLATE	WEB template name	CL48		O
W.TO	WEB TO position symbol	CL16		O
W.UNESCAPED	WEB	CL1		O
Q.BUFFLENG	MQSeries maximum buffer length	BL4	B	O
Q.CHARATTAREA	MQSeries character attribute area	CL256	B	O
Q.CHARATTLENG	MQSeries character attribute length	BL4	B	O
Q.CLOSEOPT	MQSeries close options	BL4	B	O
Q.CMDTEXT	MQSeries command text	CL10	B	O
Q.COMPCODE	MQSeries completion code	BL4	B	O
Q.CORRELID	MQSeries correlation id	CL24	B	O
Q.DATALENG	MQSeries current message length	BL4		
Q.HCONN	MQSeries connection handler	BL4		
Q.HOBJ	MQSeries object handler	BL4		
Q.INTATTARRAY	MQSeries int attribute array	CL64		
Q.INTATTCNT	MQSeries int attribute counter	BL4		
Q.INTATT*n*	MQSeries int attribute 1-16	BL4		
Q.MGRNAME	MQSeries manager name	CL48		
Q.MSGID	MQSeries message id	CL24		
Q.OPENOPT	MQSeries open options	BL4		
Q.QNAME	MQSeries queue name	CL48		
Q.REASON	MQSeries reason code	BL4		
Q.REASONTEXT	MQSeries reason text	CL30		
Q.SELCNT	MQSeries selector counter	BL4		
Q.SELECTORS	MQSeries selector array	CL64		

symbol name, description, format length		
Q.SELECTOR*n*	MQSeries selector 1-16	BL4

Reserved Field Symbols in Alphabetical Order

symbol name	description, format length				
ACCU*n*	accumulators (*n* = 0 - 99) they are fixed in the work area positions 6000 - 6990		PL8	B	O
..ACNT	added records counter UPF file		PL8	B	
APPLID	VTAM CICS application identification		CL8		O
..BL	info retrieval defined maximum block length		BL2	B	O
..BWD	option BWD for VSAM for ALLOC 'B'		CL1	B	
CALDRDATE	CALENDAR date format CCYYMMDD		ZL6	B	O
CALDRTXMT	CALENDAR name of month (January – December)		CL10	B	O
CALDRTXWD	CALENDAR name of day (Monday – Sunday)		CL10	B	O
CALDRWARN	CALENDAR warning for day adjustment at end of month		CL1	B	O
CALDRWKDY	CALENDAR day of week (1-7)		ZL1	B	O
CALDRWKNR	CALENDAR number of week (01-53)		ZL2	B	O
CCDATE	actual century date	CCYYMMDD	PL5	B	O
CCYEAR	actual century year	CCYY	ZL4	B	O
CDATE	actual current date	DDMMYY	PL4	B	O
..CDISP	cancel disp for ALLOC:	DISP=(...,...,DELETE	CL1	B	
CDTIME	current date and time	DDMMYYHHMMSS	2xPL4	B	O
	(Attention: only valid for basic instruction format)				
C.FCNT	EXECSQL fetched counter		PL8	B	O
CICSID	CICS system identification		CL4		O
CTIME	actual current time	HHMMSS	PL4	B	O
CURSOR	cursor position after GET MAP		BL4		O
CUSERID	CICS user identification		CL8		O
DATE	system date	'DD.MM.YY'	CL8	B	O
DAY	day of system date	DD	ZL2	B	O
DB2COMMIT	DB2 auto commit counter		PL8	B	
..DBN	FCA DL/I data base name DB name for dynamic allocation		CL8	B	O
..DCLAS	data class name for ALLOC:	SMS only	CL8	B	
..DCNT	deleted records counter UPF file		PL8	B	
..DDN	info retrieval DD name DD name for dynamic allocation		CL8	B	O
..DDN	DD name for ALLOC		CL8	B	
..DSN	info retrieval data set name		CL44	B	
..DSN	data set name for dyn. allocation ALLOC		CL44	B	
..DSORG	data set organization for ALLOC:		CL2	B	
..DIR	option DIR for PDS for ALLOC 'D'		CL1	B	
..DIRBL	directory blks for ALLOC:	SPACE=(...,(...,...,*nn*	BL2	B	
DIVREM	division remainder field		PL8	B	O
D*n*POS*nnnn*	record area position *nnnn* of DS*n*, DB*n*				O
EDATE	system date EURO format	'DD.MM.CCYY'	CL10	B	O
EPARM	external parameter area	max.	CL100	B	
EPARML	length of external parameter value		BL2	B	

symbol name, description, format length						
EPOS*nnnn*	EXCI communication area position (1 – *nnnn*)				B	
FC	function code	special register		BL4	B	O
FILLER	place holder for explicit symbol association				B	O
..FN	info retrieval file name			CL8	B	O
..FORM	printer FORM name			CL8	B	O
..FT	info retrieval VSAM file type KS,ES,RR			CL2	B	O
FUNCMSG	function return message			CL72	B	O
..GCNT	GET sequential read counter			PL8	B	
GPOS*nnnn*	global work area position (*nnnn* = 1 - 4096)					O
HOUR	hour of start time	HH		ZL2	B	O
HPOS*nnnn*	internal hiper space position (1 - *nnnn*)				B	
ISODATE	system date ISO standard	'CCYY-MM-DD'		CL10	B	O
..ITEM	FCA queue item			BL2		O
IV	interval value	special register		BL4	B	O
I[*n*]POS*nnnn*	record area position *nnnn* of IPF[*n*]				B	
JOBACTELNO	z/OS Job accounting element no			BL4	B	
JOBACTINFO	z/OS job accounting info			CL144	B	
JOBCLASS	z/OS job class from JOB statement CLASS=			CL1	B	
JOBCLASSLG	z/OS long job class from JOB statement CLASS= or //*MAIN CLASS=			CL1	B	
JOBNAME	JCL job name			CL8	B	
JOBNUM	z/OS JCL job number			ZL5	B	
JOBPROGRNM	z/OS programmer name from JOB statement			CL20	B	
JOBSTIME	z/OS job start time hhmmss			PL4	B	
..KEY	FCA key field			CL236	B	O
..KFBAREA	FCA DL/I key feedback area			CL256	B	
..KFBALENG	FCA DL/I key feedback area length			BL4	B	
..KFN	key field name for dynamic allocation			CL8		O
..KL	info retrieval defined key length			BL2	B	O
..KP	info retrieval defined key position			BL2	B	O
..LABEL	tape label NL or SL for ALLOC:			CL2	B	
..LCNT	line counter per page (print files)			PL2	B	O
..LENG	FCA record length (for LIBR BL2)			BL4	B	O
..LENG	FCA record length	LIBR		BL2	B	
..LEV	FCA DL/I segment level			ZL2	B	O
MAPNAME	QPAC-Online active map name			CL16		O
MAXCOL	maximum columns on screen			BL2		O
..MAXLCNT	maximum number of lines per page (print files)			PL2	B	O
MAXROW	maximum rows on screen			BL2		O
..MCLAS	management class for ALLOC:	SMS only		CL8	B	
..MEMDIRCHDT	FCA changed date from statistic record		PDS	PL5	B	
..MEMDIRCRDT	FCA creation date from statistic record		PDS	PL5	B	
..MEMDIRINIT	FCA number of initial records		PDS	BL2	B	
..MEMDIRMM	FCA modification from statistic record		PDS	BL1	B	
..MEMDIRSIZE	FCA number of records in member		PDS	BL2	B	
..MEMDIRTIME	FCA last changed time from stat. record		PDS	PL4	B	

symbol name, description, format length						
..MEMDIRUSER	FCA User Id		PDS	CL7	B	
..MEMDIRVV	FCA version from statistic record		PDS	BL1	B	
..MEMNM	FCA member name	PDS		CL8	B	
MINUTE	minute of start time	MM		ZL2	B	O
..MN	member name for generic selection	ALLOC		CL8	B	
MONTH	month of system date	MM		ZL2	B	O
..NDISP	normal disp for ALLOC:	DISP=(,CATLG...		CL1	B	
NETNAME	VTAM terminal netname			CL8		O
OPERID	CICS terminal operator identification			CL3		O
O[*n*]POS*nnnn*	record area position *nnnn* of OPF[*n*]				B	
..PCB	PCB number for dynamic allocation			BL4		O
..PCNT	PUT record counter			PL8	B	
	page counter (print files)			PL4	B	O
PRGNAME	QPAC-Online program name			CL16		O
..PRISP	primary space for ALLOC:	SPACE=(...,(*nn*,...		BL2	B	
..PSB	FCA DL/I PSB name			CL8	B	
	PSB name for dynamic allocation					O
..QNM	FCA queue name			CL8		O
Q.BUFFLENG	MQSeries maximum buffer length			BL4	B	O
Q.CHARATTAREA	MQSeries character attribute area			CL256	B	O
Q.CHARATTLENG	MQSeries character attribute length			BL4	B	O
Q.CLOSEOPT	MQSeries close options			BL4	B	O
Q.CMDTEXT	MQSeries command text			CL10	B	O
Q.COMPCODE	MQSeries completion code			BL4	B	O
Q.CORRELID	MQSeries correlation id			CL24	B	O
Q.DATALENG	MQSeries current message length			BL4	B	O
Q.HCONN	MQSeries connection handler			BL4	B	O
Q.HOBJ	MQSeries object handler			BL4	B	O
Q.INTATTARRAY	MQSeries int attribute array			CL64	B	O
Q.INTATTCNT	MQSeries int attribute counter			BL4	B	O
Q.INTATT*n*	MQSeries int attribute 1-16			BL4	B	O
Q.MGRNAME	MQSeries manager name			CL48	B	O
Q.MSGID	MQSeries message id			CL24	B	O
Q.OPENOPT	MQSeries open options			BL4	B	O
Q.QNAME	MQSeries queue name			CL48	B	O
Q.REASON	MQSeries reason code			BL4	B	O
Q.REASONTEXT	MQSeries reason text			CL30	B	O
Q.SELCNT	MQSeries selector counter			BL4	B	O
Q.SELECTORS	MQSeries selector array			CL64	B	O
Q.SELECTOR*n*	MQSeries selector 1-16			BL4	B	O
RACFUSER	z/OS RACF user id			CL8	B	
..RBA	FCA relative byte address			BL4	B	O
RC	return code	special register		BL4	B	O
..RC	FCA return code (VSAM, DL/I, queues, ...)			CL2	B	O
..RC1	FCA return code position 1			CL1	B	O
..RC2	FCA return code position 2			CL1	B	O

symbol name	description, format length					
..RCNT	READ random read counter MAP info retrieval receive counter			PL8	B	O
..RECFM	record format for ALLOC:			CL3	B	
..RL	info retrieval defined maximum record length			BL2	B	O
..RLSE	release space for ALLOC:	Yes or No		CL1	B	
..ROWLENG	FCA SQL row length			BL4	B	O
..RRN	FCA relative record number			BL4	B	O
..RTN	root segment name for dynamic allocation			CL8		O
..SCATNM	selected catalog name			CL44	B	
..SCLAS	storage class for ALLOC:	SMS only		CL8	B	
..SCNT	MAP info retrieval send counter			PL8		O
..SDISP	status disp for ALLOC:	DISP=SHR		CL1	B	
..SDSNM	selected generic dataset name			CL44	B	
SECOND	second of start time	SS		ZL2	B	O
..SECSP	sec. space for ALLOC:	SPACE=(...,(...,*nn*		BL2	B	
..SEGLENG	FCA DL/I segment length (without MPB/DBCTL)			BL4	B	
..SEGNM	FCA DL/I segment name			CL8	B	O
..SQLCODE	FCA SQL return code			BL4	B	O
..SSAN	FCA DL/I number of SSA fields in use			BL1	B	O
..SSA*n*	FCA DL/I SSA fields (*n* = 1 - 8)			CL256	B	O
STEPNAME	z/OS JCL step name			CL8	B	
..STOWID	FCA STOW identification	PDS		CL1	B	
..STOWMM	FCA STOW statistic modification		PDS	BL1	B	
..STOWUSER	FCA STOW statistic user id		PDS	CL7	B	
..STOWVV	FCA STOW statistic version		PDS	BL1	B	
SYSNAME	z/OS system name / SYSID			CL8	B	
S*n*POS*nnn*	record area position *nnn* of SPO*n*					O
..TBN	FCA SQL table name			CL18	B	O
TERMID	CICS terminal identification			CL4		O
TIME	QPAC start time	'HH:MM:SS'		CL8	B	O
..TYPSP	type of space for ALLOC:	SPACE=(CYL,...		CL1	B	
T*n*POS*nnnn*	record area position *nnnn* of TD*n*, TS*n*					O
..UCNT	updated records counter UPF file			PL8	B	
..UNIT	unit name for ALLOC:	SYSDA		CL8	B	
USDATE	system date US format	'MM/DD/CCYY'		CL10	B	O
USERID	QPAC-Online user identification			CL16		O
U[*n*]POS*nnnn*	record area position *nnnn* of UPF[*n*]				B	
..VOLID	volume serial ident for ALLOC:			CL6	B	
..WHERE	*for internal use only (DB2 support)*					
WPOS*nnnn*	internal work area position (1 - *nnnn*)				B	O
W.AT	WEB AT position symbol			CL16		O
W.BOOKMARK	WEB bookmark symbol			CL16		O
W.CADDRLENGTH	WEB client address length			BL4		O
W.CLIENTADDR	WEB client address			CL*n*		O
W.CLIENTCODEPAGE	WEB client code page value			CL40		O
W.CLIENTNAME	WEB client name			CL*n*		O

symbol name, description, format length				
W.CNAMELENGTH	WEB client name length		BL4	O
W.DATAONLY	WEB data only flag		CL1	O
W.DELIMITER	WEB		CL1	
W.DOCSIZE	WEB document size		BL4	
W.DOCTOKEN	WEB document token		CL16	
W.FNAMELENGTH	WEB form field value length		BL4	
W.FORMFIELD	WEB form field area		CL*n*	
W.FROMDOC	WEB From document value		CL16	
W.HNAMELENGTH	WEB HTTP header value length		BL4	
W.HOSTECODEPAGE	WEB host code page value		CL8	
W.HTTPHEADER	WEB HTTP header area		CL*n*	
W.LENGTH	WEB current value length		BL4	
W.PORTNUMBER	WEB port number		CL5	
W.PORTNUMNU	WEB port number binary		BL4	
W.RESP2	WEB CICS reason code		BL2	
W.SADDRLENGTH	WEB server address length		BL4	
W.SERVERADDR	WEB server address		CL*n*	
W.SERVERNAME	WEB server name		CL*n*	
W.SNAMELENGTH	WEB server name length		BL4	
W.SYMBOL	WEB symbol in symbol table		CL32	
W.TCPIPSERVICE	WEB TCP/IP service info		CL48	
W.TEMPLATE	WEB template name		CL48	
W.TO	WEB TO position symbol		CL16	
W.UNESCAPED	WEB		CL1	
XPOS*nnnn*	external area position (1 – *nnnn*)			
X*n*	Index registers (n = 1-99)		BL4	
YEAR	year of system date	YY	ZL2	

Additional Information to Reserved Field Symbols

JOBACTELNO — Contains the number of elements that are defined in the JOB statement and made available in the field JOBACTINFO.

JOBACTINFO — Contains job accounting information from the JOB statement. The field is divided into 9 sub fields of 16 bytes each where the information from the JOB statement is stored left justified.

SYSNAME — Contains the z/OS system name that is also known under the term SYSID.

Symbol Cross-Reference

The cross-reference list clearly shows how a symbol is handled by the QPAC compiler, as long as the list is not suppressed by the `PARM` option (`NOXREF`). The cross-reference gives details of the format, length and defined addressing format.

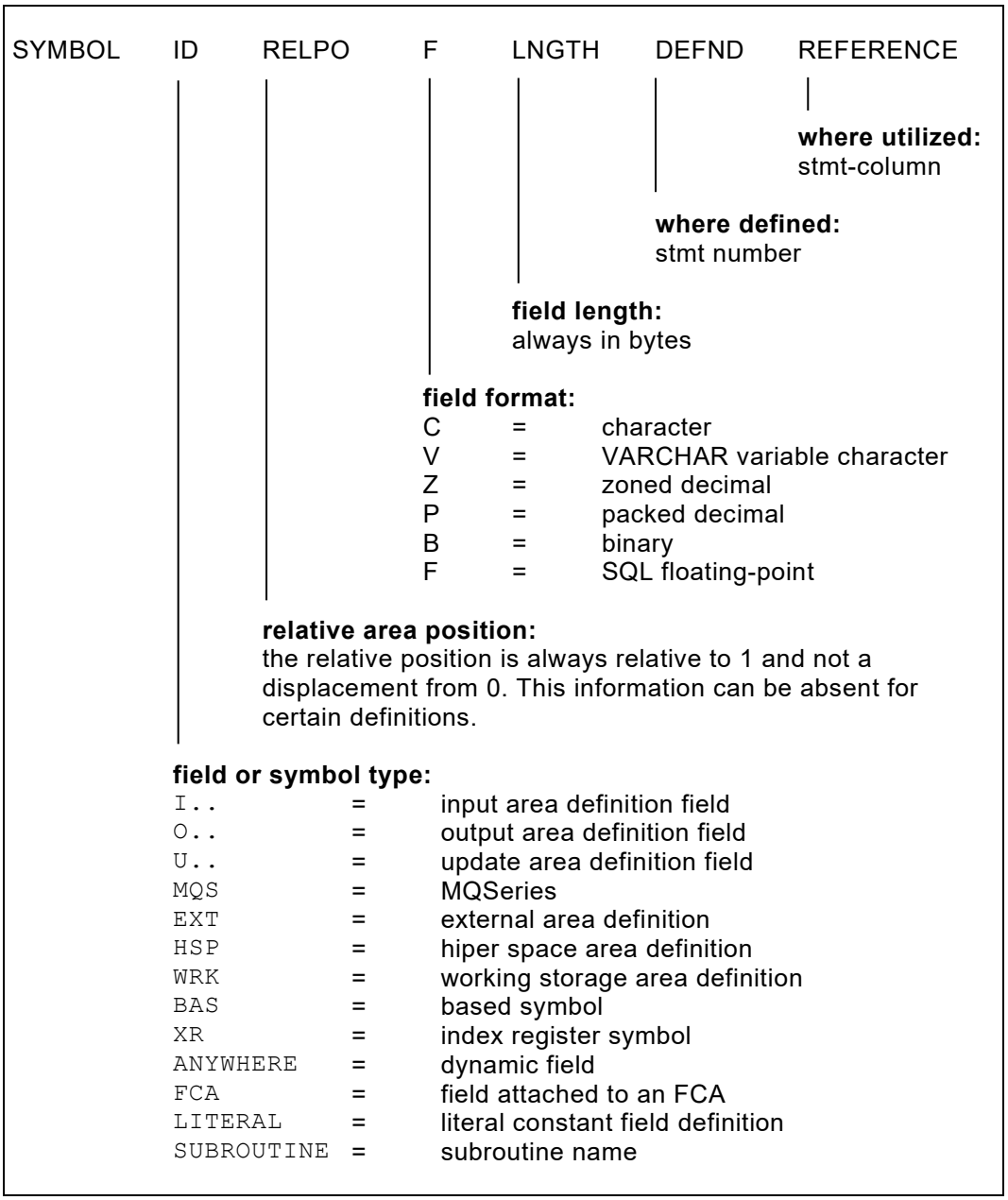


Fig. 93: Diagram symbol cross reference list

```
SYMBOL     ID          RELPO    F   LNGTH     DEFND      REFERENCE

ACCU0      WRK         6000     P   8         0001       0001-01
DATE       ANYWHERE            C   8                    0020-10
FELDX      HSP         1        C   1         0005       0009-01
```

Fig. 94: Extract of a symbol cross reference list

Chapter 7. Processing Commands

Overview and Hints

QPAC-Batch contains a complete "high-level-format" instruction set, which enables automatic data conversion to take place based on field formats. Logical and arithmetic field formats must be distinguished between, and are normally not allowed to mix:

`C` = character	logical format
`P` = packed	arithmetic format
`Z` = zoned	arithmetic format
`B` = binary	arithmetic format
`edit mask`	receiving field for an arithmetic sending field

Fig. 95: Field formats overview

The field format specified with the field definition (character or zoned) cannot be dynamically changed.

The High-Level-Format Instruction SET

Basic Format

```
SET recvg_field  =  sendg_field-1  [  op  sendg_field-2
                     literal         op  sendg_field-3
                                     op  literal  ...  ]
```

op can be:	+	for addition
	-	for subtraction
	*	for multiplication
	/	for division
	%	for modulo
	|	for concatenation

literal can be:	*nnn*	for arithmetic expressions
	-nnn	negative arithmetic expressions
	'..'	character constant, logical expression
	X'..'	hexadecimal constant, logical expression
	B'........'	bit constant, 8 bits 0 or 1
	SPACE	figurative constant
	BLANK	figurative constant
	LOWVAL	figurative constant
	HIGHVAL	figurative constant

A character literal has a maximum length of up to 2048 bytes, a hexadecimal literal has a maximum length of up to 1024 bytes. If the definition would exceed one statement line, this and every following line may be terminated by "single quote - slash - blank" and the following statement continues with a single quote.

```
CHARCONST = 'ABCDEFGH'/          *. COMMENT
            'IJKLMNOP'/          *. COMMENT
            'QRSTUVW'/           *. COMMENT
            'XYZ'

HEXCONST = X'0102030405'/        *. COMMENT
           '060708090A'/         *. COMMENT
           '0B0C0D0E0F'          *. COMMENT
```

Fig. 96: Literals over several lines

As an extended definition format an **explicit length** can precede an individual literal string:

e.g. `CL80'ABCDEF'`
`XL40'01020304'`

If the following string is shorter than the explicit length definition the character string is right padded with blanks or a hexadecimal string is right padded with low values. If continuation lines are defined they may be mixed with or without explicit lengths. In such a case any single explicit length definition is valid for the current line only:

```
e.g.   0L=CL80'ABCDE'/
              '12345'/
              '67890'/
         CL70'VWXYZ'
```

The example above results in a total length of 160 bytes.
Between the number "0" and the letter "V" are 65 blanks, following the letter "Z" are 65 blanks.

Special formats

```
SET recvg_field  =CX  sendg_field   (char -> hex)
                 =XC               (hex -> char)
                 =TR               (translate)
                 =MN               (move numeric)
                 =MZ               (move zone)
                 =MO               (move with offset)
                 =AND              (boolean AND)
                 =OR               (boolean OR)
                 =XOR              (boolean exclusive OR)
                 =ADR              (load field address)
                 =C'.'             (padding character)
                 =X'..'            (padding hex value)
                 =CTS              (convert timestamp)

                 =edit mask        (predefined edit masks)
```

```
SET recvg_field,CLXn = sendg_field       (variable field lengths)
SET recvg_field      = sendg_field,CLXn
SET recvg_field,CLXn = sendg_field,CLXn
```

Logical and **arithmetical** operation codes are distinguished between and **may not be mixed** within a set of operands.

Bracketed expressions are supported with the basic format with an arithmetic set of operands.

The SET high-level-format instruction begins with the keyword SET and is designated a yield format instruction. When dealing with such a field, its format attribute will be taken into consideration, i.e. different formats will be converted automatically into the correct format.

The individual logical members of a SET instruction are separated from each other by a blank, i.e. a blank is the delimiter for a logical entity. An individual instruction can contain one receiving field and many sending fields, or operands.

The receiving field is separated from the sending field by the equals sign.

The order of the operands is defined by the presence or absence of an operation code. If it is absent, the end of the instruction is assumed. There must be at least one operand for each instruction.

receiving field = sending field

If, with arithmetic expressions, following the yield definition, there is an operation sign instead of the first operand, the receiving field is taken as the first operand.

receiving field = *op* operand-2

(equals to: receiving field = receiving field *op* operand-2,
op can be a +, −, * / or %)

A symbol can be defined as a receiving field, following the symbol rules as described in the preceding chapter.

A symbol or literal can be defined as the sending field or operand. The literal is either a character string, if it is contained within apostrophes, a hexadecimal value, if the leading apostrophe is preceded by an X, or an arithmetic value (numeric literal), if it is a simple number.

VARCHAR variable character fields have their length in a preceding length field. This length field is disregarded if the field is address modified or defined with an explicit format length.
If a variable character field is defined as a receiving field the total length of the sending part is stored in the length field if it is not longer than the maximum length of the variable character field. If this is the case the maximum length is stored and any remaining rest of the sending part is ignored. If the sending part is smaller than the maximum length of the receiving field the rest is padded with blanks or any explicitly defined padding value.

Variable character fields may be concatenated combined with literals or with index registers variable defined fields.

The SET Transfer Instruction

```
SET recvg_field  =  sendg_field

SET recvg_field  =  sendg_field-1  |  sendg_field-2  ...
```

The SET transfer instruction is a logical instruction.

Logical operation code: | vertical line = concatenate

The operation code must be separated from the operands by at least one blank sign.

The logical operation code category can handle the C field format combined with character or hex literals.

```
SET STRING = FIELD_A
SET STRING = 'ABCDEFG'

SET STRING = FIELD_A | 'ABCDEFGH' | X'00'
```

Fig. 97: Transfer and concatenate with the SET transfer instruction

The receiving field of a logical 'yield instruction' can have a different length to the sending field or the sum of the concatenated operands.

If the receiving field is too short, the rest of the sending side is ignored.
If the receiving field is longer than the sum of the sending side, the remaining part is filled with blanks as far as no padding value is specified, otherwise with the defined character or hex value.

There is one exception to these basic rules:

a) If the receiving field is **address modified and no explicit length is defined**, the implicit length is increased, and replaced by the length of the **sending field** or the total length of all the sending fields, as long as this is smaller than the implicit length of the receiving field.

b) If the receiving field is address modified, and an explicit field length is also defined, the latter is taken as the length of the receiving field:

```
SET FIELD+1 = '*' ──────────────►       *

SET FIELD+1,CL5 = '*' ──────────►  *bbbb
```

Fig. 98: SET transfer instruction with address modified receiving field

c) If the sending field is a figurative constant e.g. SPACE and the only operand (not concatenated) then the whole receiving field is set to this value:

```
SET FIELD = LOWVAL
```

Fig. 99: SET transfer instruction with figurative expression

The SET Arithmetic Instruction

```
SET recvg_field  =  sendg_field-1  op  sendg_field-2
```

Logical and SET arithmetic operation codes are to be distinguished between, and they may not occur together within one set of operands.

arithmetic operation codes:

+	**addition**
-	**subtraction**
*****	**multiplication**
/	**division** (any remainder value is in the field DIVREM)
%	**modulo**

The operation code must be separated from the operands by at least one blank.

Arithmetic operation codes can cope with the field formats P, Z, B in any order, together with numeric literals.

```
SET ACCU0 = X1 * 5 - ACCU9 / 5
```

Fig. 100: Combination of arithmetic field formats

An arithmetic expression consists of a set of arithmetic operations, and is solved according to mathematical rules, i.e. multiplication, division and modulo are carried out before addition and subtraction.

```
SET ACCU0 = X1 * 5  -  ACCU9 / 5        (1.)
SET ACCU0 = product -  quotient         (2.)
```

Fig. 101: Solution according to mathematical rules

A negative numeric literal can be defined if the number is immediately preceded by a minus sign, without an intervening blank.

```
SET ACCU0 = ACCU0 * -1
```

Fig. 102: Negative numeric literals

Bracketed expressions are supported with arithmetic operations.

```
SET ACCU0 = X1 * ( 5 - ACCU9 / 5 )
```

Fig. 103: Bracketed expressions

Note: After every division a potential remainder value is available in the reserved field symbol DIVREM (division remainder) and its content will be overwritten by any following division operation.

With the modulo operation (% sign) a certain modulo value can be calculated in an arithmetic expression and stored in the receiving field.

```
SET ACCU0 = 10
SET ACCU1 = ACCU0 % 7
```

Fig. 104: Modulo

After execution, the modulo value of the operation ACCU0 mod 7 is in the receiving field ACCU1, in the example the remainder value of the integer division 10 / 7 = 1, remainder 3.

The SET Transfer Instruction (Special Format)

```
SET recvg_field  =CX     sendg_field
SET recvg_field  =XC     sendg_field
SET recvg_field  =TR     sendg_field
SET recvg_field  =MN     sendg_field
SET recvg_field  =MZ     sendg_field
SET recvg_field  =MO     sendg_field
SET recvg_field  =AND    sendg_field
SET recvg_field  =OR     sendg_field
SET recvg_field  =XOR    sendg_field
SET recvg_field  =C'.'   sendg_field
SET recvg_field  =X'..'  sendg_field
SET recvg_field  =CTS    sendg_field
```

The special instruction **=CX** (character to hexadecimal)
converts the sending field into hexadecimal format and places the result into the receiving field. The receiving field must be twice as big as the sending field.

The special instruction convert hex to char **=XC** (hexadecimal to character)
converts the sending field, which must contain hexadecimal characters, into non-hexadecimal format, and places the result in the receiving field. The receiving field is half the size of the sending field.

```
SET HEXFIELD,CL8  =CX CHARFIELD,CL4
SET CHARFIELD,CL4 =XC HEXFIELD,CL8
```

Fig. 105: Character - Hexadecimal conversion

The special instruction **=TR** (translate)
is a translation function. The receiving field is translated; the sending field serves as a 256 bytes translation table whose contents is defined by the user.

```
SET FIELD =TR X'00010203......'
SET FIELD =TR TABLE
```

Fig. 106: Translate

The special instruction **=MN** and **=MZ** allow to move only the numeric (=MN) or the zoned (=MZ) part of the sending field to the receiving field. The corresponding opposite half-byte will not be modified.

```
before:                              NULLFIELD  = X'F0F0F0F0'
                                     VALUEFIELD = X'C1A1F1F2'

SET NULLFIELD =MN VALUEFIELD

after:                               NULLFIELD  = X'F1F1F1F2'
                                     VALUEFIELD = unchanged

SET VALUEFIELD =MZ NULLFIELD

after:                               VALUEFIELD = X'F1F1F1F2'
                                     NULLFIELD  = unchanged
```

Fig. 107: Move numeric and move zone

The special instruction **=MO** transfers the sending field which is shifted to the left by half a byte to the receiving field. If the receiving field is longer than the sending field, it will be left padded with binary zeroes.

```
before:                              DATE  X'19990112'
                                     FIELD X'000000000C'

SET FIELD =MO DATE

after:                               FIELD X'019990112C'
```

Fig. 108: Move with offset

The special instruction **=AND**
applies the boolean "AND" function to the receiving and the sending field.

```
SET FIELD1 =AND FIELD2          1 + 1 = 1
                                0 + 1 = 0   both 1 -> 1
                                1 + 0 = 0   other cases -> 0
                                0 + 0 = 0
```

Fig. 109: Boolean AND function

The special instruction **=OR**
applies the Boolean "OR" function to the receiving and the sending field.

```
SET FIELD1 =OR FIELD2           1 + 1 = 1
                                0 + 1 = 1   both 0 -> 0
                                1 + 0 = 1   other cases -> 1
                                0 + 0 = 0
```

Fig. 110: Boolean OR function

The special instruction **=XOR**
applies the Boolean "Exclusive OR" function to the receiving and the sending field.

```
SET FIELD1 =XOR FIELD2          [ 1 + 1 = 0
                                  0 + 1 = 1   both equal -> 0
                                  1 + 0 = 1   other cases -> 1
                                  0 + 0 = 0 ]
```

Fig. 111: Boolean Exclusive OR function

The special instruction **=CTS** supports in a very simple manner the conversion of the TOD clock timestamp format into a zoned format and vice versa.
The timestamp format can be 8 bytes binary (TOD clock) or 20 bytes zoned.
The direction of the conversion is specified by the fields.
If the receiving field has 8 bytes binary format then the sending field MUST have a 20 bytes zoned or character format. The other way round the receiving field must have 20 bytes zoned or character format if the sending field has an 8 bytes binary format.
A description of the TOD clock format can be found in the IBM manual "principles of operation".

```
0S=BINARYTS,BL8                 Timestamp in binary format (TOD clock)
0S=ZONEDTS,ZL20                 Timestamp in display format
                                May also be defined as CL20.
                                ZonedTS format:
                                YYYYMMDDHHMMSShtmu00

SET BINARYTS =CTS ZONEDTS       Conversion from zoned format to the
                                binary format

SET ZONEDTS =CTS BINARYTS       Conversion from the binary format to the
                                zoned format
```

Fig. 112: Timestamp Conversion

The SET Edit Instruction

```
SET RECVG_FIELD,M'Edit-Mask' = SENDG_FIELD

e.g. RECVG_FIELD,M'ZZ9.999,99-'
                ,M'-ZZ9.999,99'
```

Fig. 113: self-defined edit masks

An edit mask may be defined in a "self made" format for the receiving field. Leading Z result in zero-suppression. Z and 9 represent digit positions.

```
SET recvg_field =EDA   sendg_field
                =EDAZ
                =EDAS

                  Mask type A = ▲▲▲▲▲▲–
```

Fig. 114: Mask type A

The numeric *sendg_field* is edited into the *recvg_field* according to the mask type A.
This extended edit operation enables editing without punctuation.
A negative numeric field results in a '–' sign stored in the right most position.

zero suppression takes place if either **EDAZ** or **EDAS** is specified:
- EDAS results in zero suppression up to, **but not including**, the last digit
- EDAZ results in zero suppression up to, **and including**, the last digit

e.g..

```
SET OPOS65,CL4 =EDAZ IPOS10,PL5

                Pos.10        =  X'019376219D'
                fully edited  =     19376219-
                Pos.65        =          219-
```

Fig. 115: Sample resolution mask type A

```
SET recvg_field =EDB sendg_field

                    Mask type B = ▲▲.▲▲.▲▲.▲▲
```

Fig. 116: Mask type B

The numeric *sendg_field* is edited into the *recvg_field* according to the mask type B.
This extended edit operation enables editing in groups of 2 decimal digits, separated by a full stop.
A negative field value is not marked as such.
Zero suppression is not possible.

e.g.

```
SET OPOS65,CL8 =EDB IPOS10,PL5

                    Pos.10        =  X'019376219D'
                    fully edited  =   19.37.62.19
                    Pos.65        =      37.62.19
```

Fig. 117: Sample resolution mask type B

The editing rules for the following mask type are the same as described under EDB:

```
SET recvg_field =EDC sendg_field

                    Mask type C = ▲▲:▲▲:▲▲:▲▲:▲▲
```

Fig. 118: Mask type C

The editing rules for the following mask types are the same as described under EDA:

```
SET recvg_field =EDD sendg_field
                =EDDZ
                =EDDS

                    Mask type D = ▲▲▲.▲▲▲.▲▲▲,▲▲▲
```

Fig. 119: Mask type D

```
SET recvg_field =EDE sendg_field
                =EDEZ
                =EDES

                    Mask type E = ▲▲▲,▲▲▲,▲▲▲.▲▲–
```

Fig. 120: Mask type E

```
SET recvg_field =EDF sendg_field
                =EDFZ
                =EDFS

                  Mask type F = ▲▲▲'▲▲▲'▲▲▲.▲▲–
```

Fig. 121: Maskentyp F

```
SET recvg_field =EDG sendg_field
                =EDGZ
                =EDGS

                  Mask type G = ▲▲▲  ▲▲▲ ▲▲▲–
```

Fig. 122: Mask type G

```
SET recvg_field =EDH sendg_field
                =EDHZ
                =EDHS

                  Mask type H = ▲▲▲,▲▲▲,▲▲▲–
```

Fig. 123: Mask type H

```
SET recvg_field =EDI sendg_field
                =EDIZ
                =EDIS

                  Mask type I = ▲▲▲.▲▲▲.▲▲▲–
```

Fig. 124: Mask type I

```
SET recvg_field =EDK sendg_field
                =EDKZ
                =EDKS

                  Mask type K = ▲▲▲▲▲▲▲▲▲▲,▲▲–
```

Fig. 125: Mask type K

The SET Transfer Instruction for Variable Field Lenghts

This special format transfer instruction allows the definition of variable field lengths.

```
SET rcvg_field,CLXn = sndg_field
SET rcvg_field      = sndg_field,CLXn [ | sndg_field,CLXn ..]
SET rcvg_field,CLXn = sndg_field,CLXn [ | sndg_field,CLXn ..]
```

An index register may explicitly be defined, and its value is taken as the field length.

Index Register Instructions and Indexed Addressing

There are 99 index registers available to the user, which can be used for indexed addressing. The index registers are:
`X1 X2 X3 X4 X5 X8 X7 X8 X9 X10.........X99`
Indexed addressing occurs when an index register is attached to the field symbol by a plus or minus operator.
A maximum of 3 index registers can be defined

The index register contents at execution time will be added to or subtracted from the base address, according to the operator.

It should be noted that index registers wrongly set on the side of the receiving field will be checked, but not on the side of the sending field. It is thereby possible to address an area outside of the record or work area limits.

```
SET TABLE+X1    = INPUTFIELD
SET TABLE+X1-X2 = + 1

IF RECORD+X1 = '000' THEN ...
IF RECORD+X1-X2 NOT = X'FF' THEN ...
```

Fig. 126: SET instruction with indexed addressing

Index registers are initialized with zero.

The content of the index registers can be changed by the following **short form index register instructions** (no multiplication or division).

X*n*+*m*	add value *m* to index register X*n*
X*n*-*m*	subtract value *m* from index register X*n*
X*n*=*m*	load index register X*n* with value *m*
X*n*+X*m*	add index register X*m* to X*n*
X*n*=X*m*	load index register X*n* with value of X*m*

Fig. 127: Short form index register instructions

An index register can also be handled like any data field, using the `SET` instruction

```
SET X1 = X1 * 2
```

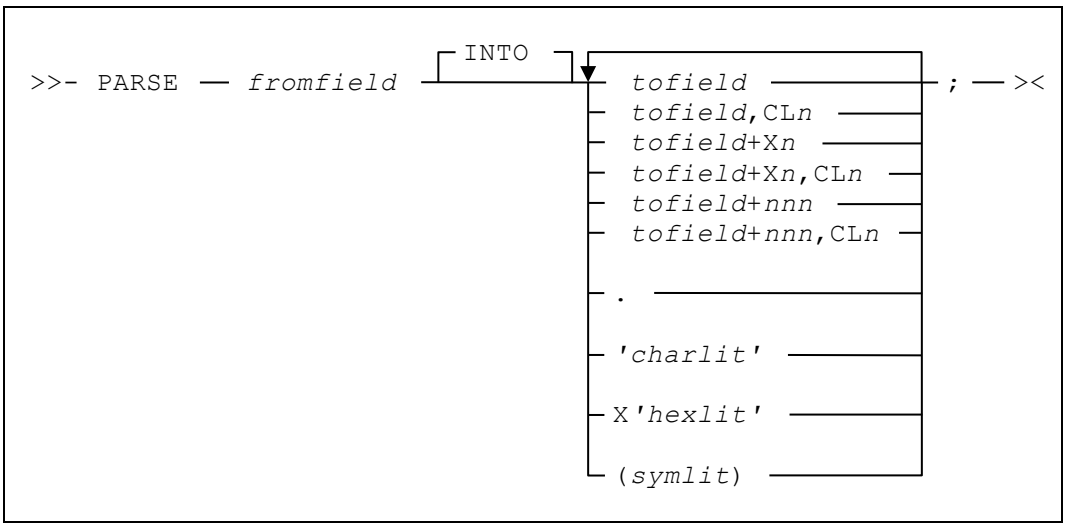
Fig. 128: SET instruction and index registers

When loading values from the input or work areas into the index registers, the values must be numeric. An index register has a `BL4` format (4 binary bytes).

```
SET X1 = ZONEDFIELD,ZL2
SET X1 = BINARYFIELD,BL2
```

Fig. 129: Loading index registers by use of the SET instruction

The PARSE Instruction



The PARSE instruction extracts parts of a sending field into following defined receiving fields. The parts of the sending field can thereby be distributed according to delimiters defined within the instruction. If a delimiter is not explicitly defined then blank is assumed.

Attention: The receving fields must be previously defined. The PARSE instruction does not automatically defined them.

fromfield	sending field, whose content is distributed to the receiving fields.
INTO	optional key word, for documentation only.
tofield	receiving field According to its position within the PARSE instruction the corresponding part (string) of the sending field is stored into it. Be aware of any delimiter that can be explicitly defined following the *tofield*.
tofield,CL*n* *tofield*+X*n* *tofield*+X*n*,CL*n* *tofield*+*nnn* *tofield*+*nnn*,CL*n*	the receiving field can be address modified according to QPAC rules. If the receiving field is a VARCHAR field the attribute V is replaced by C.
.	Dot A dot signalizes that the corresponding part (string) of the sending field has to be skipped.
'charlit'	character literal as a delimiter The preceding part of the sending field up to this delimiter is stored into the receiving field that is preceding the delimiter definition.
'hexlit'	hexadecimal literal as a delimiter The preceding part of the sending field up to this delimiter is stored into the receiving field that is preceding the delimiter definition.

(symlit)	a defined field symbol in brackets whose content is taken as a delimiter The preceding part of the sending field up to this delimiter is stored into the receiving field that is preceding the delimiter definition.

If fewer receiving fields are defined than parts (strings) do exist within the sending field, then any rest is also stored into the last receiving field.

If more receiving fields are defined than parts (strings) do exist within the sending field, then any remaining receiving field is cleared according to its attribute.

Chapter 8. Logic Control Commands

QPAC-Batch recognises condition definitions by IF and DO structures. These are either relation conditions, whereby two operands are compared with each other, or, a simple examination of a status condition, for example to examine a specific error condition.

The IF THEN ELSE Instruction

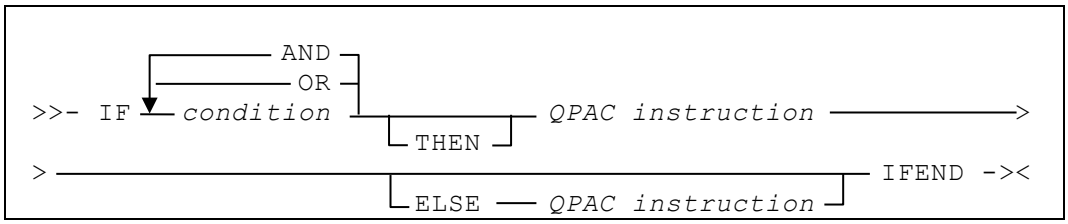


Fig. 130: Diagram basic format condition definition

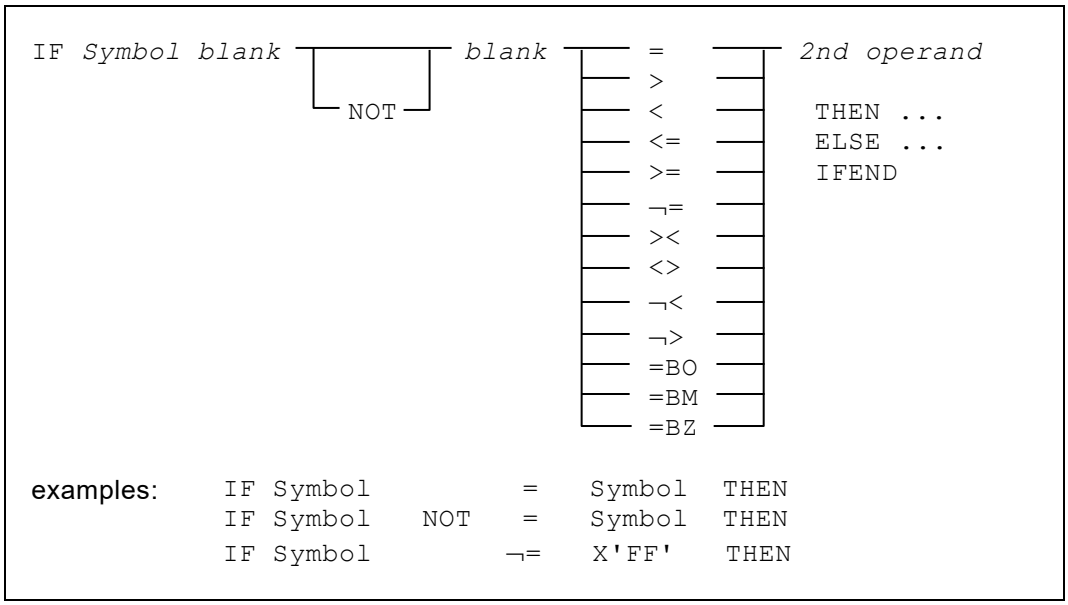


Fig. 131: Diagram relation condition

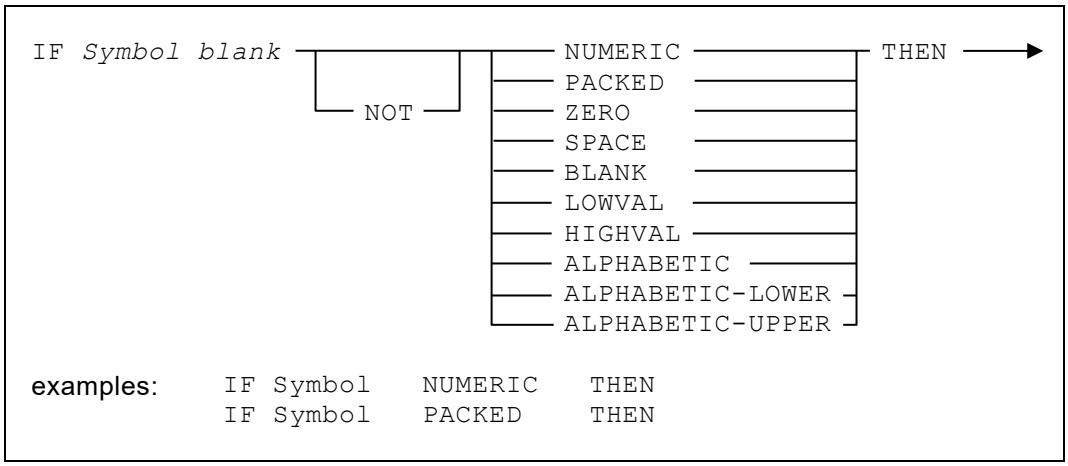


Fig. 132: Diagram status condition

The individual logical members of a condition definition are separated from each other by a blank. A single condition normally consists of two operands, brought together by the comparison operator.
Both operands can be defined by a symbol, which must follow the symbol rules as stated in a previous chapter.
Operand-2 could also be a literal, a character string within apostrophes defining a character literal, an X preceding the first apostrophe defining a hex literal, with a simple number being a numeric literal or a B preceding the apostrophe defining a bit constant.
A negative numeric literal is defined by the minus sign preceding the number:

```
IF ACCU0 = 150 THEN ...
IF CHAR  = 'ABCDE' THEN ...
IF SWITCH =BO B'00000001' THEN ...
IF SWITCH =BZ X'01' THEN ...
IF NUMBER = -1
```

Fig. 133: Symbols and literals within condition definitions

The following signs can be used as comparison operators:

=	comparison to equal	<=	comparison to less or equal
>	comparison to greater	=<	comparison to less or equal
<	comparison to less than	>=	comparison to greater or equal
¬>	comparison to not greater	=>	comparison to greater or equal
¬<	comparison to not less		
¬=	comparison to not equal	=BO	test bit ones
<>	comparison to not equal	=BM	test bit mixed
><	comparison to not equal	=BZ	test bit zero

Comparison operators preceded by the keyword **NOT** are negated:

NOT =	not equal
NOT >	not greater than
...	...

Logical and arithmetic field formats may not be mixed as operand pairs, as no valid conversion is possible. Such a combination is rejected as an error by the compiler at conversion time.

When comparing variable character fields the content of the preceding length field is considered.

If two logical operands that are to be compared have different lengths, the comparison is in the **length of the smaller field** as long as both lengths are not larger than 256 bytes and these are not variable fields.

Arithmetic field formats can be mixed amongst each other:

```
IF X1 < ACCU0
```

Fig. 134: Comparison of different arithmetic formats

A figurative constant can be defined as the comparison operator. In this case, *operand-2* is not defined.

These figurative constants are supported in this shorthand:

SPACE	X'40'
BLANK	X'40'
ZERO	X'F0'
LOWVAL	X'00'
HIGHVAL	X'FF'
PACKED	Is the content of the field packed?
NUMERIC	(zoned format) Is the field content purely arithmetic?
ALPHABETIC	Combination of A to Z, a to z, and space?
ALPHABETIC-UPPER	Combination of A to Z, and space?
ALPHABETIC-LOWER	Combination of a to z, and space?
ENTERED	*(QPAC-Online map fields only)*
ERASED	*(QPAC-Online map fields only)*

These figurative expressions can also be negated by NOT:

```
IF QUANTITY PACKED
IF QUANTITY NUMERIC

IF QUANTITY NOT PACKED
IF QUANTITY NOT NUMERIC

IF NAME ALPHABETIC
IF NAME NOT ALPHABETIC-UPPER
```

Fig. 135: Figurative constants within condition definitions

Comparison operands with variable length are possible for character fields.

```
IF ANYFIELD,CLXn  = OTHERFIELD
IF ANYFIELD       = OTHERFIELD,CLXn
```

Fig. 136: Comparison operands with variable length

Condition definitions consisting of two operands to be compared with each other, are called relation conditions.

Such relation conditions can be logically joined together. In order to do this, QPAC recognizes the two **Boolean operators** AND and OR.

Joining simple relation conditions by Boolean operators results in combined conditions.
Any number of combined conditions, in any variation can be defined using the AND and OR operators. But the rule for the solution of such expressions must be followed. This rule states: **on the same hierarchical level, the AND combinations will be solved first, in their procedural order**.

```
IF    c1    AND    c2    OR    c3    AND    c4    THEN
```

solves in the first step to:

```
IF          c12          OR    c3    AND    c4    THEN
```

solves in the second step to:

```
IF          c12          OR          c34          THEN
```

Fig. 137: Example of the solution rule of combined conditions

The QPAC condition definition also allows for bracketed expressions, thereby avoiding some redundant relation definitions. It can influence the outcome under the rule for solving expressions with Boolean operators, by changing the hierarchy.

The use of brackets is nothing other than an extended format of the combined condition. QPAC recognizes the following elements of the combined condition definition:

a)	simple-condition	(relation condition)
b)	AND	(Boolean operator)
c)	OR	(Boolean operator)
d)	(	(combined structure element)
e)	)	(combined structure element)

The following table shows the rule concerning the allowed sequence for elements in a combined condition:

combined condition element	allowed as first definition on left?	what can precede this element?	what can follow this element?	allowed as last definition on right?
simple condition	yes	AND OR (	AND OR)	yes
AND OR	no	)	simple condition (	no
(	yes	AND OR (	simple condition	no
)	no	simple condition)	AND OR)	yes

Fig. 138: Rule for combined conditions

Brackets must never be defined if only AND or only OR is used in a combined condition:

```
IF    c1    OR    c2    OR    c3    OR    c4    THEN
IF    c1    AND   c2    AND   c3    AND   c4    THEN
```

Brackets are not necessary if the definition of the combined condition corresponds to the solution rule, in that the AND combinations will be solved first:

```
IF    c1    AND    c2    OR    c1    AND    c4    THEN
IF    c1    OR     c2    AND   c3    AND    c4    THEN
```

Brackets can be used to avoid redundant 'simple conditions':

```
IF    c1    AND    c2    OR    c1    AND    c4    THEN
IF    c1    AND  ( c2    OR                 c4 )  THEN
```

When brackets are defined, the left and right sides must correspond one to one:

```
IF  ( ( c1  OR  c2 )  AND ( c3  OR  c4 )  AND  c5 ) THEN
```

Bracketing up to a depth of 10 levels is allowed. As long as the hierarchical structure depth is not reached, the number of pairs of brackets is not limited.

Visual Examples of the Process

Condition statement with or without an alternative:

A condition statement begins with the question `IF` followed by a comparison operation, and logically ends the **true case** with `THEN`, or the **false case** with `ELSE`:

```
IF I1POS1 = '10'  THEN  SET O1POS1 = I1POS1,CL2
                  ELSE  SET O1POS1 = '??'       IFEND
```

The keyword `IFEND` ends the logic and states the physical end of the `THEN` and `ELSE` branches:

```
IF I1POS1 = '10'  THEN  SET O1POS1 = I1POS1,CL2 IFEND

IF I1POS1 = '20'  THEN
                  ELSE  SET O1POS1 = '??'       IFEND
```

Multiple comparison operations can be joined by `AND` and `OR`:

```
IF I1POS1  = '10'   AND
   I1POS12 = X'00'  OR  I1POS1  = '10'  AND
                        I1POS12 > X'20' THEN ...
IFEND
```

The same condition can be formulated using brackets:

```
     IF  I1POS1  = '10'   AND
       ( I1POS12 = X'00'  OR  I1POS12 > X'20' ) THEN ...
     IFEND
```

Multiple conditions can be nested up to 15 levels.

```
IF I1POS10 = '1' THEN
|  IF I1POS20 = '2' THEN
|  |  IF I1POS30 = '3' THEN
|  |  |  SET OUTPUT_RECORD = INPUT
|  |  IFEND
|  |  ACCU20 = + 1
|  IFEND
|  ACCU10 = + 1
IFEND
```

Fig. 139: Example 1 without alternatives

```
IF I1POS10 = '1' THEN ──────────────────────────────────────────────────────┐
|                                                                           |
|  IF I1POS20 = '2' THEN ─────────────────────────────────────────────────┐  |
|  |                                                                      |  |
|  |  IF I1POS30 = '3' THEN ──────────────────────────────────────────┐   |  |
|  |  |                                                               |   |  |
|  |  |  IF I1POS40 = '4' THEN ──────────────────────────────────┐    |   |  |
|  |  |  |                                                       |    |   |  |
|  |  |  |  IF I1POS50 = '5' THEN ──────────────────────┐        |    |   |  |
|  |  |  |  |                                           |        |    |   |  |
|  |  |  |  |  SET OUTPUTFIELD1 = '11'                  |        |    |   |  |
|  |  |  |  |  SET OUTPUTFIELD2 = '22'                  |        |    |   |  |
|  |  |  |  |  SET INDICATOR    = '5'                   |        |    |   |  |
|  |  |  |  └──────────────────────────────────────────┘        |    |   |  |
|  |  |  |  ELSE ──────────────────────────────────────┐        |    |   |  |
|  |  |  |  |                                          |        |    |   |  |
|  |  |  |  |  SET INDICATOR    = '4'                  |        |    |   |  |
|  |  |  |  |                                          |        |    |   |  |
|  |  |  |  IFEND ─────────────────────────────────────┘        |    |   |  |
|  |  |  └──────────────────────────────────────────────────────┘    |   |  |
|  |  |  ELSE ──────────────────────────────────────────┐            |   |  |
|  |  |  |                                              |            |   |  |
|  |  |  |  IF I1POS60 = '6' THEN ──────────────┐       |            |   |  |
|  |  |  |  |                                   |       |            |   |  |
|  |  |  |  |  SET OUTPUTFIELD3 = '33'          |       |            |   |  |
|  |  |  |  |  SET OUTPUTFIELD2 = '22'          |       |            |   |  |
|  |  |  |  |  SET OUTPUTFIELD4 = '44'          |       |            |   |  |
|  |  |  |  └──────────────────────────────────┘       |            |   |  |
|  |  |  |  ELSE ──────────────────────────────┐       |            |   |  |
|  |  |  |  |                                  |       |            |   |  |
|  |  |  |  |  SET INDICATOR    = '8'          |       |            |   |  |
|  |  |  |  |                                  |       |            |   |  |
|  |  |  |  IFEND ─────────────────────────────┘       |            |   |  |
|  |  |  |                                            |            |   |  |
|  |  |  IFEND ──────────────────────────────────────┘              |   |  |
|  |  |                                                            |   |  |
|  |  ELSE ──────────────────────────────────────────────┐         |   |  |
|  |  |                                                  |         |   |  |
|  |  |  SET INDICATOR = '1'                             |         |   |  |
|  |  |                                                  |         |   |  |
|  |  IFEND ─────────────────────────────────────────────┘         |   |  |
|  |                                                              |   |  |
|  IFEND ──────────────────────────────────────────────────────────┘   |  |
|                                                                     |  |
IFEND ──────────────────────────────────────────────────────────────────┘
```

Fig. 140: Example 2 with alternatives

ELSEIF Case Structure

Condition Statement with Several Alternatives

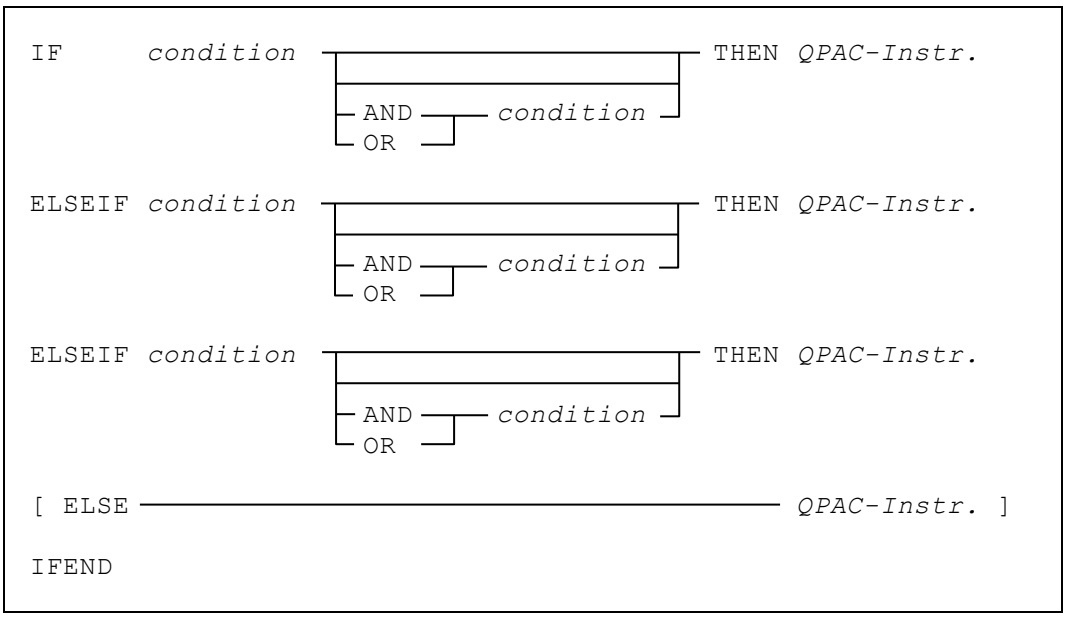


Fig. 141: Diagram ELSEIF case structure

The use of the `ELSEIF` keyword leads to a simple form of a condition statement with many possible alternatives, where further conditions can be stated in the false case.

```
IF      I1POS1,PL4 = 1  THEN  CALL-'ROUTINE1'
ELSEIF  I1POS1,PL4 = 2  THEN  CALL-'ROUTINE2'
ELSEIF  I1POS1,PL4 = 3  THEN  CALL-'ROUTINE3'
ELSE    WTO-'DATA ERROR - PROGRAM TERMINATED'
IFEND
```

Fig. 142: Usage of ELSEIF case structure

The above example would look as follows, using nested standard condition statements:

```
IF   I1POS1,PL4 = 1  THEN  CALL-'ROUTINE1'
ELSE IF   I1POS1,PL4 = 2  THEN  CALL-'ROUTINE2'
     ELSE IF   I1POS1,PL4 = 3  THEN  CALL-'ROUTINE3'
          ELSE
               WTO-'DATA ERROR - PROGRAM TERMINATED'
          IFEND
     IFEND
IFEND
```

Fig. 143: Traditional nesting of IF statements

DO Loop Instructions

The DO instruction enables the specification of loops with either a stated, or conditional number of repeats.

DO-*nn*	absolute number of repeats
DO-X*n*	absolute number of repeats in index register
DO-WHILE	positive condition number of repeats
DO-UNTIL	negative condition number of repeats
DO-FOREVER	endless loop
DOBREAK	jump to the beginning of the DO loop
DOQUIT	immediately leave the DO loop
DOEND	end of loop

Fig. 144: DO-Loop instructions overview

The DO-nn Loop Instruction

Basic format of loop instruction with **fixed number of repeats**.

Fig. 145: Diagram of loop instruction with fixed number of repeats

nn is an absolute value which defines the number of repeats.
DOEND defines the end of the loop.

Nesting up to **10 levels** is allowed.

```
X1=0
DO-10
   SET WPOS7000+X1,PL8 = 1
   X1+8
DOEND
```

Fig. 146: Usage of loop instruction with fixed number of repeats

Thereby, 10 table fields of 8 bytes each are initialized. (See also indexed addressing).

The DO-Xn Instruction

Basic format of loop instruction with **fixed modifiable number of repeats**.

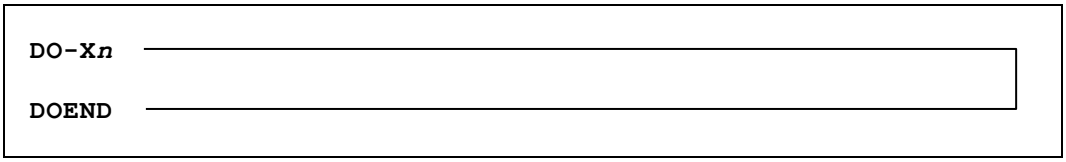


Fig. 147: Diagram of loop instruction with fixed modifiable number of repeats

`Xn` is an index register whose content defines the number of repeats. This index register is reduced by 1 before each loop. It can thereby be used for relative addressing within the `DO` loop. It can also be changed within the `DO` block, which will change the number of repeats.
`DOEND` defines the end of the loop.

```
X10=80
DO-X10                  (the first time X10 contains 79)
   SET OPOS80-X10 = IPOS1+X10,CL1
DOEND
```

Fig. 148: Usage of loop instruction with fixed modifiable number of repeats

The 80 character input record is transferred from dataset 1 to dataset 2 in reverse order.

The DO-WHILE Instruction

Basic format of loop instruction with **positive condition repetition "while"**.

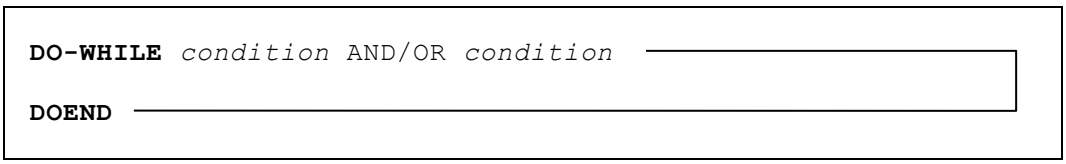


Fig. 149: Diagram of loop instruction with positive condition repetition

Condition is a processing dependency. As long as the condition is met, the loop is processed. The format and usage of the condition correspond exactly to that of the `IF` statement, with the exception that there is no `THEN` following the last condition.

The loop is ended by `DOEND`.

```
DO-WHILE I1POS1 = '10' AND I1POS12 = X'00'
   O1POS1 = I1POS1,CL4  PUT-O1
   GET-I1 AT-EOF DOQUIT ATEND
DOEND
```

Fig. 150: Usage of loop instruction with positive condition repetition

As long as positions 1-2 in the record are equal to '10', and position 12 equals X'00', the article number is transferred, and the next record read. A loop exit occurs if the condition is not fulfilled, or end of file is reached.

The DO-UNTIL Instruction

Basic format of loop instruction with **negative condition repetition "until"**.

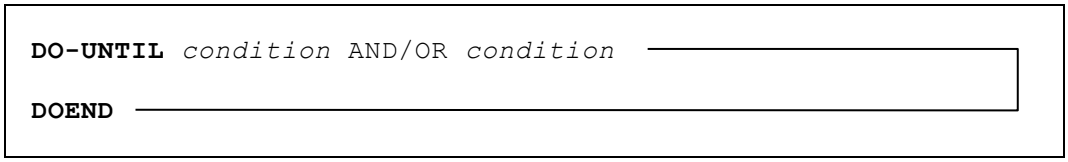


Fig. 151: Diagram of loop instruction with negative condition repetition

Condition is a processing dependency. As long as the condition is not met, the DO loop is processed. The format and usage of the condition correspond exactly to that of the IF statement, with the exception that THEN does not follow the condition.

The loop is ended with DOEND.

```
GET-I1
DO-UNTIL I1POS1 = X'FF' OR I1POS1 = '100'
   ...
   GET-I1
DOEND
```

Fig. 152: Usage of loop instruction with negative condition repetition

A record will be read within the loop until a record with the identification 100 in columns 1-3 is found, or until end of file.

The DO-FOREVER Instruction

Basic format of loop instruction with the **endless cycle**.

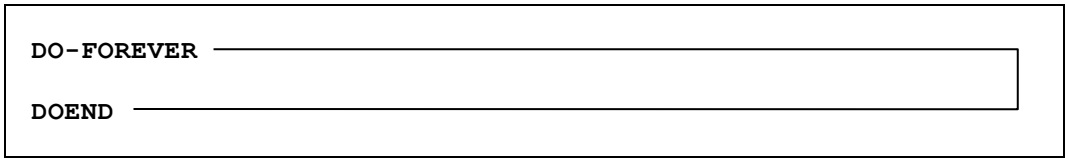


Fig. 153: Diagram of loop instruction with endless cycle

The DO command contains no condition or value that can be altered within the cycle. The loop never ends. The loop can be left by means of a DOQUIT or a GO command.

```
DO-FOREVER
   ...
   GET-I1 AT-EOF DOQUIT ATEND
   ...
DOEND
```

Fig. 154: Usage of loop instruction with endless cycle

Extended Logic Commands for Loop Instructions

The DOBREAK Instruction

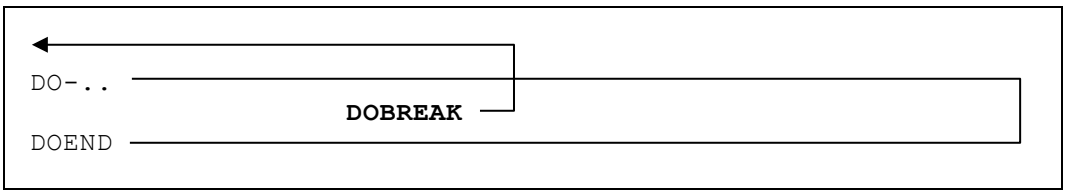


Fig. 155: Diagram of DOBREAK instruction

Stops processing in sequence and jumps to the beginning of the DO loop.

```
X1=80
DO-X1
   ...
   IF WPOS7000+X1 = ' ' THEN DOBREAK IFEND
   ...
DOEND
```

Fig. 156: DOBREAK branches to the beginning of the loop

In the above example, the first non-blank character from the right is looked for.

The DOQUIT Instruction

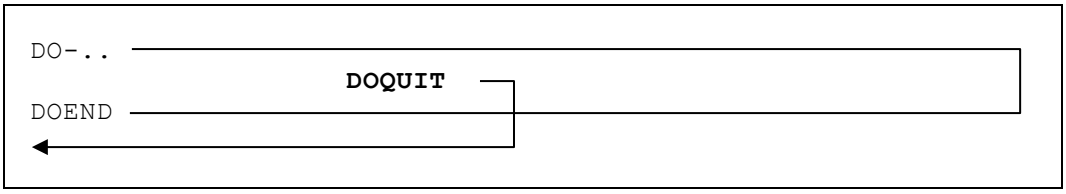


Fig. 157: Diagram of DOQUIT instruction

The DO block is exited from immediately, and processing continues following the DOEND.

```
DO-WHILE ...
  IF ... THEN DOBREAK IFEND
  DO-5
  ...
     DO-WHILE ...
        IF ... THEN DOQUIT IFEND
     DOEND
  DOEND
DOEND
```

Fig. 158: DOQUIT immediately leaves the loop

Chapter 9. Subroutines and External Programs

Internal Subroutine CSUB

An independently defined subroutine can be called using CSUB (call subroutine). The subroutine name can be 1 to 30 alphanumeric characters in length. Any number of CSUB commands can call the same subroutine, and the CSUB command can be given anywhere within the QPAC program, as long as it is logically correct.

CSUB-*name*	=	CALL subroutine
SUB-*name*	=	subroutine header definition
SUBEND	=	subroutine end definition

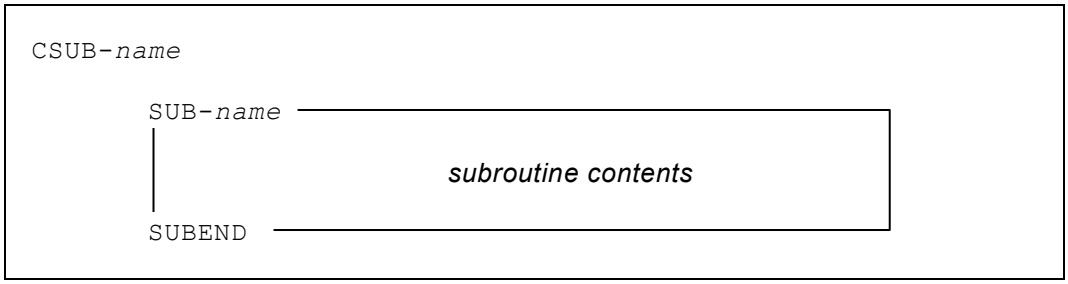


Fig. 159: Basic format of subroutine usage

SUB-*name* defines the beginning of the subroutine, where *name* is the identification by which the subroutine is called by CSUB. All processing and logic instructions can be used within the subroutine, including calls to other subroutines, i.e. nesting is allowed.

SUBEND defines the end of the subroutine. SUB and its corresponding SUBEND must be on hierarchical level 0, i.e. they may not be within an IF or DO structure block etc..

A subroutine is only processed if called by CSUB. If defined within the procedural sequence, the subroutine will be jumped over if sequentially met and not called by CSUB.

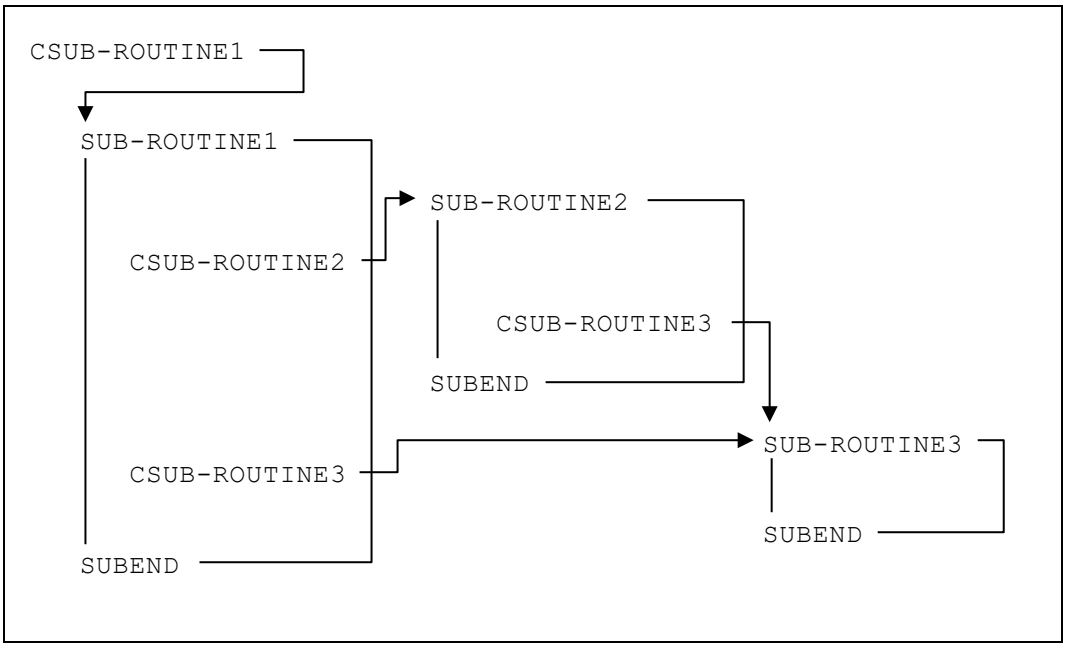
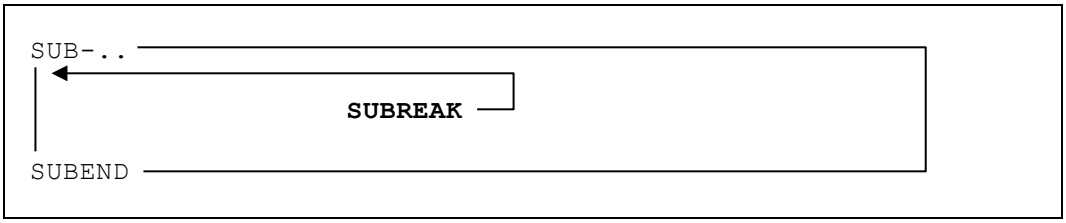


Fig. 160: Graphical example of subroutine nesting

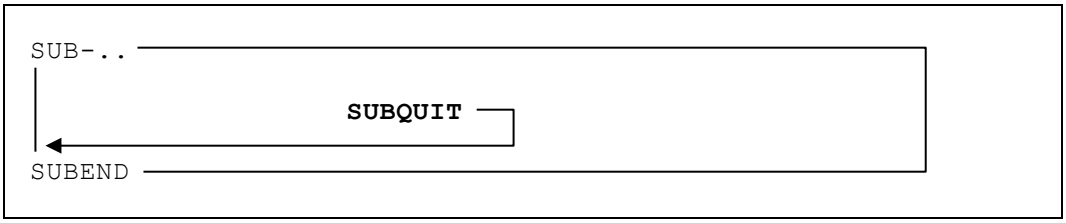
Additional Control Commands for Subroutines

The SUBREAK Instruction



Causes an immediate end to processing within the subroutine flow, and a direct jump back to the start of the subroutine.

The SUBQUIT Instruction



Causes processing to immediately leave the subroutine i.e. a direct jump to `SUBEND`.

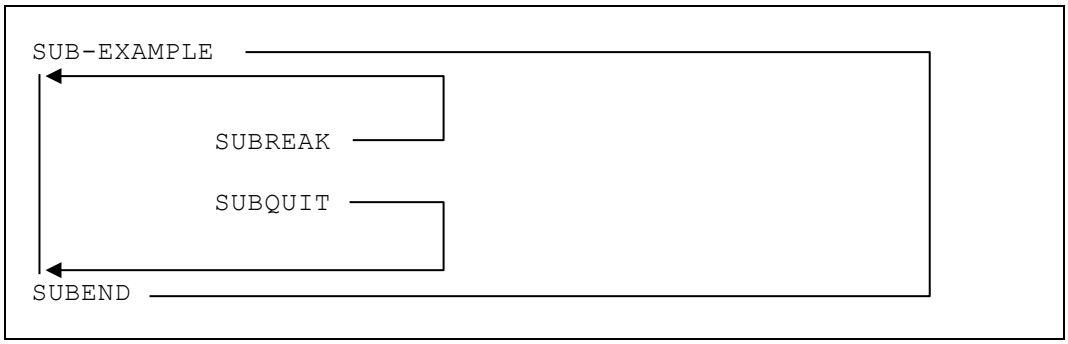


Fig. 161: Subroutine instructions SUBREAK and SUBQUIT

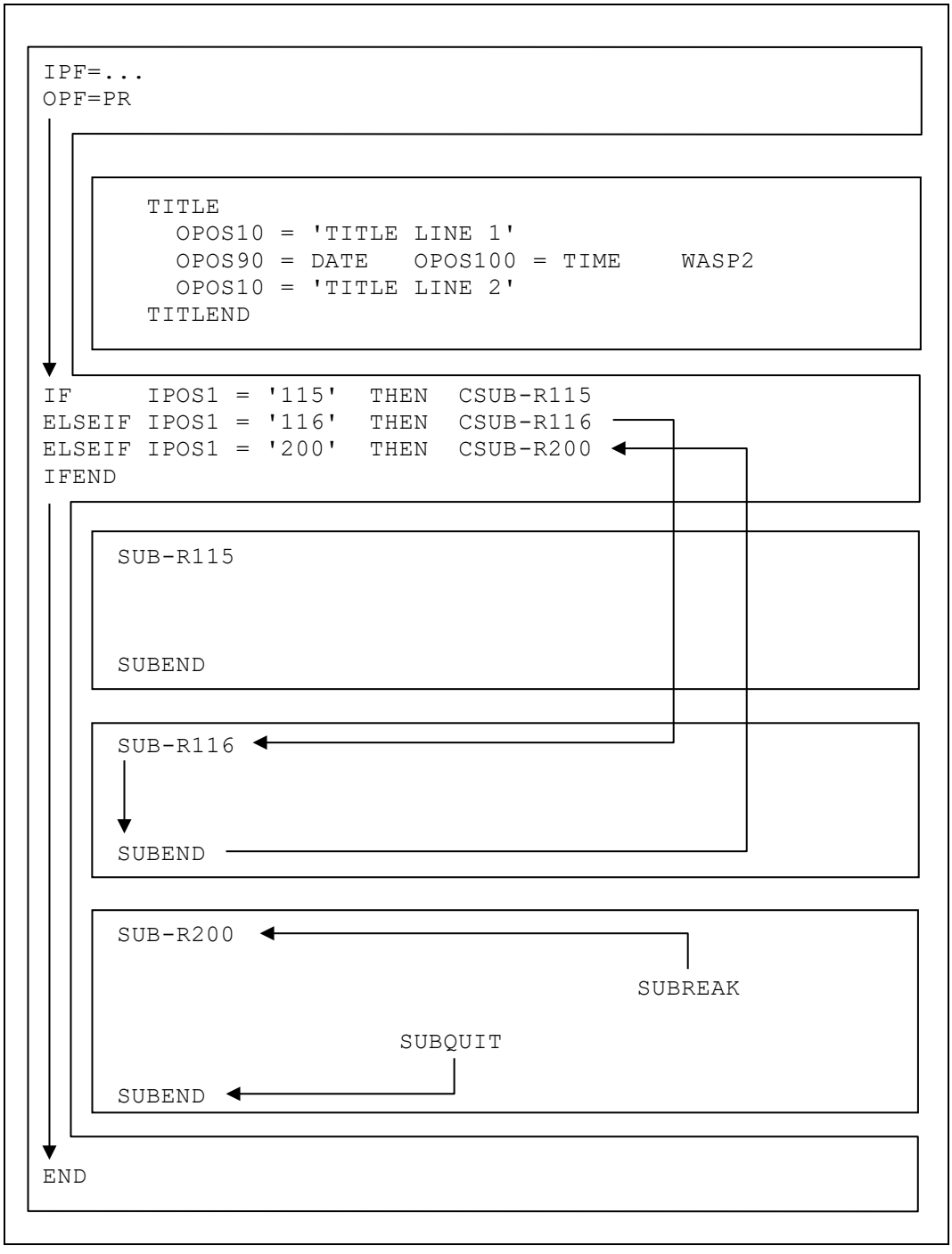


Fig. 162: Subroutine processing

External Subroutines (CALL Exit Routines)

```
CALL='loadmodule'   [ ,parameter,parameter ]
```

Specific **external processing routines** can be called, using the `CALL` function, from anywhere in the QPAC coding. `CALL`ed routines are completely independent from QPAC, they are loaded for processing as a load module from a link library (z/OS).

The design of external routines must comply with official linkage conventions, such as starting with the SAVE macro and ending with a RETURN.

By using a parameter address, the QPAC internal working storage is at the disposal of the routine.

On entry into the routine:	Register 15	routine's basic value
	Register 14	return address
	Register 1	parameter list address
	Register 13	save area address

Registers 2-12 must be saved, if they are used in the routine (SAVE/RETURN).

QPAC internally transfers COBOL or PL/I routines to Language Environment (LE) where they are executed. Therefore, the LE-function PIPI is used. PIPI knows two possibilities how to treat these routines. They are defined with the type of initialization: The SUBroutine mode or the MAINprogram mode.

The differences of these two modes are:

1. MAINprogram
   The WORKING STORAGE SECTION of COBOL programs is initialized with each program launch.
   The COBOL program must be compiled with the option RENT.
   The PL/I program is compiled with the procedure option MAIN.

2. SUBroutine
   The WORKING STORAGE SECTION of COBOL programs keeps its content from a previous call when it is called again.
   The PL/I program must be compiled with the procedure option FETCHABLE.

These two modes may not appear concurrently within the same QPAC program. Therefore, the mode to be used is specified by a QPAC `PARM` option: `PARM=CALL=SUB` or `PARM=CALL=MAIN`. If an explicit `PARM` option is missing `CALL=SUB` is assumed.

Assembler format of the linkage convention for the exit routine:

```
CALL-'routine',wadr
```

Fig. 163: Format of CALL instruction for Assembler

wadr addresses the QPAC internal working storage.

```
               USING *,15
ENTRY          SAVE (14,12)

               ST 13,SAVEAREA+4
               LR 12,13
               LA 13,SAVEAREA
               ST 13,8(12)            Backward chain
               L 1,0(1)               Load QPAC int.
                                      Storage Address


EXIT           L 13,SAVEAREA+4
               RETURN (14,12)
SAVEAREA       DS 18F
```

Fig. 164: Sample Assembler exit routine

Independent exit routines can be supplied with parameters.

```
CALL-'loadmodule',wadr [,wadr,... ,PCBn,PCB-name ...] [,RC]
```

The standard linkage conventions, as previously described, are applied.

If work area addresses are defined, the default value, as described under the basic CALL function, no longer applies; only the defined addresses are communicated to the routine.

```
CALL-'SUBROUT',WPOS5000,WPOS5010,WPOS6990,WPOS7000
CALL-'ROUTINE',WPOS5000,PCB1,PCB2
```

Fig. 165: Passing work areas to external programs

In addition, please note that the high order bit in the high order byte of the last address in the parameter address list is set to ON.

DL/I PCB addresses can also be communicated:

PCB*n*	=	the *n*th PCB in the PSB
PCB-*name*	=	PCB of the DB-Name (DBN=)

Work area positions in the CALL command **cannot be indexed**.

```
CALL-'SUBROUT',WPOS5000+XI
```

Fig. 166: Indexed working storage addresses are NOT ALLOWED

RC can be defined as the last parameter, whereby, a return code from an external routine can be referred to within QPAC.

In case the operands for the CALL statement can not be fully defined in one statement, it is possible to continue the definition in the following statement by using a comma followed by a blank as a delimiter. The first non-blank position in the following statement constitutes the continuation:

```
CALL-'ROUTINE',WPOS5000,WPOS7000,
       PCB1,PCB2
```

Fig. 167: Continuation lines of CALL instruction

The return code, either supplied by an external routine (or set within the QPAC (RC=)), should be examined as follows:

```
CALL-'ROUTINE',WPOS5000,RC
IF RC <> 0 THEN
```

Fig. 168: Examining the return code RC

RC is a special register name, and as such is internally defined.

External Subroutines (LINK Exit Routines)

```
LINK-'loadmodule'  [ ,parameter,parameter ]
```

The LINK command corresponds in its functionality to the CALL command with the difference that the routine is newly loaded every time and released after the call.

The definitions of the operands are documented under the CALL command and can be read there.

```
LOAD-'loadmodule',ptrfield[,Xn]

LOAD-fieldname,ptrfield[,Xn]
```

With this load command external tables (load modules in z/OS) can be loaded that are thereafter being processed in the QPAC program. As operands a pointer field and an index register must be defined. The pointer field contains afterwards the storage address where the table begins and the index register contains the table length (module length).
Tables that are loaded this way are afterwards being processed with a based structure. For this the structure must correspond to the format of the table elements.

```
00B=PTRFIELD,PTR
01B=TABLE_KEY,CL3
11B=TABLE_TEXT,CL77
...
...
  LOAD-'ANYTABLE',PTRFIELD,X10
  ...
  IF TABLE_KEY = '100' THEN found
  ELSE SET PTRFIELD = + 80            *. next element
```

Fig. 169: Example Load Table Based Structure

The end of the table may be recognized by an end element or by cumulating the individual element lengths and comparing them with the maximum length in the index register.

Instead of directly specifying the module name the name of a field may be defined where the module name will be stored at the time of execution just before the `LOAD` command is executed.

If the loaded module is an assembler program it may be afterwards called with `CALL-Ptrfield,....`

```
0S=MODULENAME,CL8
0B=MODULEADDRESS,PTR

SET MODULENAME = 'ASSPROG'
LOAD-MODULENAME,MODULEADDRESS
...
NORMAL
...
CALL-MODULEADDRESS,WPOS1,WPOS500
...
```

Fig. 170: Example dynamic loading of a module

Deletion of Loaded Tables or Sub Routines

```
DELMOD-'loadmodule'
DELMOD-symbolname
```

With this command modules which have been previously loaded with the LOAD command can be selectively deleted from the memory.

QPAC as Subroutine (Called from User Main Program)

From an Assembler or Cobol program a QPAC-Batch program may be called up as a subroutine as follows:

```
LOAD   ..QPAC..                  load QPAC nucleus and
ST     R1,QPACENTR               save ENTRY address
```

Initial Call

Initial call of the loaded QPAC nucleus serves for assembling the QPAC-Batch program. A parameter area is transferred in which QPAC PARM options can be defined. The option **SUBINIT** must be present in the PARM area as first operand. This says that QPAC is called up as a subprogram from a higher-ranking program. Afterwards further options can be defined which concern QPAC itself, e.g. NOLIST,NOLOGTIT,WORK=*nnn*

The source code of the QPAC program is normally read in via //QPACIN DD * under z/OS. If this is not possible, reading in can be controlled via the PDS data set //QPACPGM under z/OS with the PARM option QPGM=*progname*. *Progname* is the member name under which the source code is stored.

Another possibility comprises the assembled form as load module, which is controlled by the PARM definition QMOD=*progname*. These possibilities are described in Chapter 1: Introduction under PARM Option.

A third possibility is having the QPAC program code in the internal work area. If the length of the first two bytes is greater than 100 it is automatically assumed that the QPAC program itself follows the SUBINIT statement in 80 bytes elements. The length then contains the program size which is terminated by an END statement.

This way QPAC can load and call itself (LOAD-'QPAC'...
...CALL-*pointer*,...).

If QPAC is called as a sub program from a QPAC main program the listing of the sub program can be separated from the main program by specifying the DD statement //QPACSUBL DD SYSOUT=*.

As a further parameter address pair, the address of an area called External Area and its length can be transferred if required. These are called up in the QPAC program by field symbols, see DSECT in Assembler or LINKAGE SECTION in COBOL.

QPAC recognizes whether such an area is planned from the set high-order bit, or through a DUMMY definition (zero value). An existing area is not moved into the QPAC program. The length can be up to 3 MB.
In a subroutine QPAC this area can be called up directly by implicit symbol

definitions XPOS*nnn*, or by allocating own symbol names according to the rule *nn*X=SYMBOL.

```
SET XPOS10,CL5 = 'ABCDE'
or
10X=SYMBOL,CL5
SET SYMBOL = 'ABCDE'
```

Fig. 171: Addressing the external area

```
         L     R15,QPACENTR                              1)
         CALL  (15),(PARMS),VL                           2)
or       CALL  (15),(PARMS,DUMMY)                        3)
or       CALL  (15),(PARMS,AREALEN,AREADATA),VL          4)
         LTR   R15,R15                                   5)
         BNZ   INITERR                                   6)
         ...
PARMS    DC    F'100',CL100'SUBINIT,NOLIST,QPGM=xxxx,...'
AREALEN  DC    F'32000'
AREADATA DC    32000CL1' '
DUMMY    DC    F'0'                                      7)

or

PARMS    DC    F'800',CL80'SUBINIT,XREF'
         DC    CL80'PARM=LIST          '
         DC    CL80'IPF1=VSAM,DYNAMIC  '
         DC    CL80'                   '
         DC    CL80'ALLOC-I1           '
         DC    CL80'...                '
         DC    CL80'END                '

1) load entry address
2) initialization call without area
3) without VL, without area
4) with area
5) RC=0?
6) error found?
7) length value is 0
```

Fig. 172: QPAC as a subroutine: Initial call

Subsequent Calls

Subsequent calls always mean an execution of the QPAC-Batch program. This is best conceived as a subroutine. The QPAC program can be equipped with additional work areas as required. These are transferred on call into the internal QPAC work area, where they are available for the QPAC program.

When the CALL command is given, 3 pieces of information are supplied:
1. Position of internal work area to which the area is being transferred
2. Length of area
3. Address of area.

Before return, such areas are transferred back again and are then available again for the calling program.
When the CALL statement is made, a `PARM` field is defined containing the key word **`SUBEXEC`**. This signals the QPAC program that a call to program execution is involved. Whether additional data areas are transferred is indicated by the high-order bit or DUMMY field (zero value). The same applies to the number of defined areas, or the end of the operands.

```
         L     R15,QPACENTR
         CALL  (15),(PARMS),VL                          1)
or       CALL  (15),(PARMS,DUMMY)                       2)
or       CALL  (15),(PARMS,WPOS,WLEN,WAREA,...),VL
or       CALL  (15),(PARMS,WPOS,WLEN,WAREA,...,DUMMY)   3)
         LTR   R15,R15
         BNZ   CALLERR
         .
         .
PARMS    DC    F'7',C'SUBEXEC'
WPOS     DC    F'10001'
WLEN     DC    F'4096'
WAREA    DC    CL4096' '
WPOS2    DC    F'00001'
WLEN2    DC    F'2000'
WAREA2   DC    CL2000' '
DUMMY    DC    F'0'
```

1) execute program without areas
2) without VL, without areas
3) without VL, with areas

Fig. 173: QPAC as a subroutine: Subsequent calls

Final Call

Final call must occur to delete from the memory all areas dynamically installed by QPAC-Batch as well as the assembled program. When the `CALL` statement is made, a `PARM` field is defined containing the key word SUBTERM (PARM option)**`SUBTERM`**. This signals the QPAC program that deletion of all dynamically installed areas and loaded load modules is involved. Afterwards the QPAC program no longer exists in the memory.

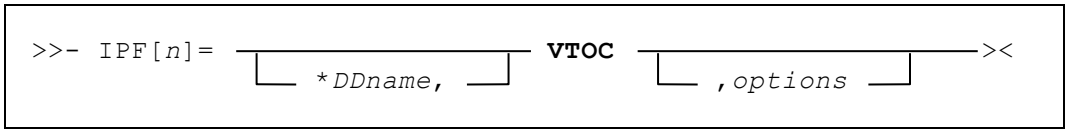
```
         L     R15,QPACENTR
         CALL  (15),(PARMEND),VL                          1)
or       CALL  (15),(PARMEND,DUMMY)                       2)
         LTR   R15,R15
         BNZ   ENDERR
         ...
PARMEND  DC    F'7',C'SUBTERM'
DUMMY    DC    F'0'
         ...
QPACENTR DC    A(0)                                       3)

1) with VL
2) without VL
3) entry address
```

Fig. 174: QPAC as a subroutine: Final call

Chapter 10. System Libraries and System Components

VTOC

Basic Format of VTOC File Definition



DDname	explicitly defined DDname. If this definition is missing, `IPF` is taken as DDname.
options	additional options as described in Chapter 2: Input/Output Definitions
`F4`	If `F4` is specified as an option the first VTOC record that is output will be the DSCB F4 (format 4) record.

To read the VTOC, a DD statement indicating the required volume, is required. A dataset name (DSN=) is not necessary.

```
//IPF1 DD UNIT=SYSDA,VOL=SER=XXXXXX
//IPF2 DD DSN=QPAC.LIBRARY,DISP=SHR
```

Fig. 175: DD statement for z/OS VTOC records

General Hints on VTOC Usage

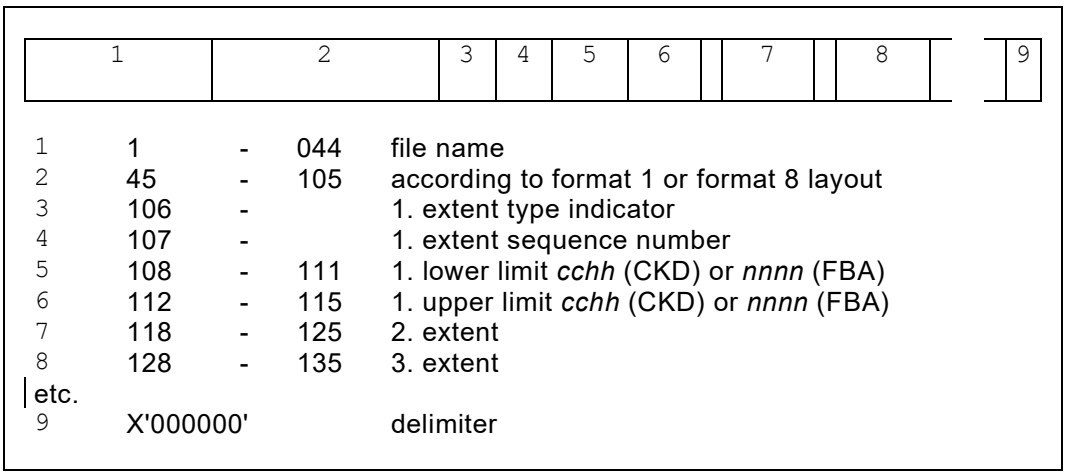
The VTOC of the addressed disk is read logically. Format 1 records and the corresponding extent data and also format 8, format 9 and format 3 records are made available to the user.

After the QPAC open of the VTOC file, the volume serial number is in the FCA, with a displacement of 2.

Positions 1 to 135 of the format 1 records remain unchanged. Starting with position 136, QPAC appends to the format 1 record all the extent information for the corresponding format 3 record.
Thereby all the necessary information is available through a single read command.

10 bytes per extent are used. Extent data is delimited by X'0000'.

For **EAV volumes** also format 8 and format 9 records plus if appropriate the corresponding format 3 records are presented. Complete records (140 bytes) are presented. The record layout is described in the IBM manual "DFSMSdfp Advanced Services“. Position 45 contains the format oft he record.



1	1	-	044	file name
2	45	-	105	according to format 1 or format 8 layout
3	106	-		1. extent type indicator
4	107	-		1. extent sequence number
5	108	-	111	1. lower limit *cchh* (CKD) or *nnnn* (FBA)
6	112	-	115	1. upper limit *cchh* (CKD) or *nnnn* (FBA)
7	118	-	125	2. extent
8	128	-	135	3. extent
etc.				
9	X'000000'			delimiter

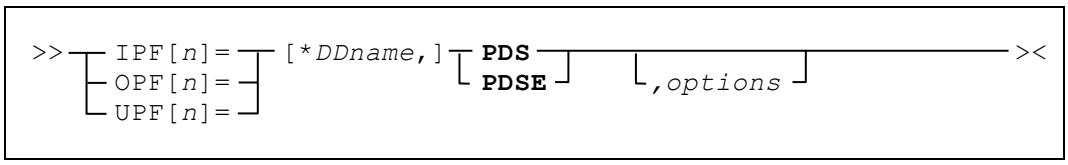
Fig. 176: VTOC layout returned by QPAC

```
//IPF1 DD UNIT=SYSDA,VOL=SER=...
//IPF2 DD
//OPF1 DD DSN=...
//EXEC QPAC
 IPF1=VTOC,WP=5000
 IPF2=VTOC,WP=5000
 OPF1=SQ,200,3600,WP=5000
 FILE1: GET-I1 AT-EOF GOTO FILE2 ATEND
        ...
        PUT-O1 GO TO FILE1
 FILE2: GET-I2 AT-EOF GOEND ATEND
        ...
        PUT-O1 GO TO FILE2
        ...
 END
```

Fig. 177: Example reading VTOCs

z/OS-Libraries (Partitioned Data Sets)

Basic Format of z/OS-Library File Definition



DDname — explicitly defined DD name.
If this definition is missing, `IPF` is taken as DDname.

options — additional options
as described in Chapter 2: Input/Output Definitions

`DIR` — directory only
directory information is given instead of data records. The directory-definition can also be dynamically defined using the reserved field symbol `..DIR`. Therefore the field `..DIR` has to be filled with a "D" before the OPEN.
e.g. `SET I1DIR = 'D'`

`FCA=....` — the FCA option is an important one. The FCA is dynamically assigned if not explicitly defined.

`MN=....` — member name
single members or groups of members can be selectively read. The definition follows the 'pattern matching' principle, whereby the following two wildcards can be used:

* represents **one or more characters**. Only one asterisk per definition is allowed, and this must be either at the beginning or the end.

\+ **marks a position** and represents **one valid character**. This marker can appear anywhere within the definition.

This member selection can also be dynamically defined using the reserved field symbol `..MN`. Therefore the field `..MN` must be filled with the member name or any generic part before the OPEN.

`NOE` — After `SETGK/SETEK` no E status code will be returned in the FCA.

EOM — end of member
The end of each member is displayed. The field `..RC2` returns an "E" after the last member record without containing a record in the record area..

MN=*name*	individual member name
MN=++++T	all member names with a length of 5 bytes ending in T
MN=*T	all member names ending in T
MN=T*	all member names, beginning with T
MN=*Z+	all member names with a Z in the second to last position

Fig. 178: Member selection for PDS library file

OPF[*n*]=PDS allows to insert new members into an existing PDS. The member records are written by the PUT instruction. Finally the member name has to be put into the FCA field ..MEMNM; the member is inserted into the PDS directory by the STOW instruction.
The FCA field ..STOWID signals whether an existing member may be replaced ('R') or only an addition may be done ('A').

UPF[*n*]=PDS allows the content of existing members to be modified and new members to be added.
With the PUTA instruction new members are inserted. Such members have to be completed with the STOW instruction (as previously described under OPF[*n*]=PDS).
With the instructions sequence GET - PUT existing member records may be modified but no records may be added or deleted.
The number of member records may be modified by "copying" them with the instructions sequence GET - PUTA.

Members can be deleted by the PUTD instruction. Before executing the PUTD instruction the FCA field ..MEMNM must be filled with the member name. After execution the return code 'G' is returned in the FCA field ..RC2 if the member does not exist.

General Hints on z/OS Libraries

The PDS libraries can have either fixed record lengths or be 'undefined'. In both cases the actual record length will be stored in the FCA with a displacement of 12 (4 bytes binary).
With an 'undefined' PDS it should be remembered that the QPAC internal record area length, which is the default, is 32760 bytes; this can lead to a WRONG LENGTH situation, which can be remedied by an RL= definition:

```
IPF=PDS,RL=32767
OPF=PDS,RL=32767
```

Fig. 179: Record length definition for PDS

A partitioned dataset can be read as a full library file or as a member file. In either case the FCA is needed for two-way communication.
The actual number of data bytes read from the PDS record is stored in the field ..LENG in binary format. If the PDS has an undefined format, and the record length may exceed 32760 bytes, the operand RL=*nnnnn* must be defined in the file definition. This operand overwrites the default length, which is required for addressing the internal work area.

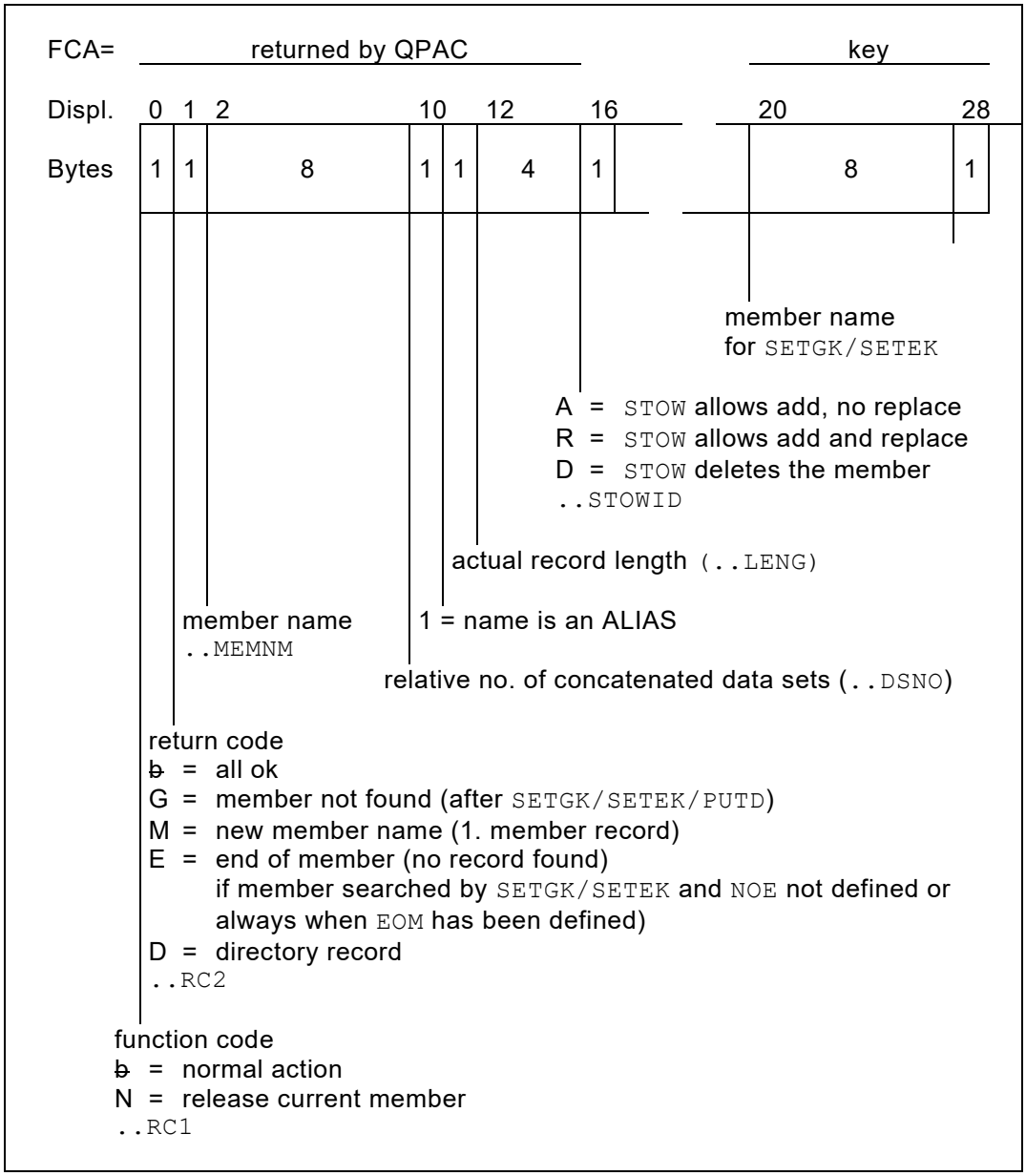


Fig. 180: FCA for PDS library file access

PUT for output files and PUTA for update files now update statistical information in the directory record. Therefore, the new fields STOWVV, STOWMM or STOWUSER can be filled before the STOW command in addition to the STOWID field.

..STOWVV,BL1	Version for the PDS directory statistics
..STOWMM,BL1	Modification level for the PDS directory statistics
..STOWUSER,CL7	UserId for the PDS directory statistics

Âfter the first GET command in a member (M in the ..RC2 field) the FCA contains additional information from the directory statistics record in a formatted format. This information can be collected by using the following reserved field symbols:

..MEMDIRVV,BL1	Version in the member directory statistics record
..MEMDIRMM,BL1	Modification in the member directory statistics record
..MEMDIRCRDT,PL5	Creation date in the member directory statistics record
..MEMDIRCHDT,PL5	Changed date in the member directory statistics record
..MEMDIRTIME,PL4	Time hhmmss in the member directory statistics record
..MEMDIRSIZE,BL2	Number of records in the member directory statistics record
..MEMDIRINIT,BL2	Initial size in the member directory statistics record
..MEMDIRUSER,CL7	UserId in the member directory statistics record

The ..RC2 code "G" after a SETGK command signalizes that the member name in the predefined key field does not exist. During the following GET command the next higher member is accessed or EOF is given if not additionally EOM has been defined.

If after a code "G" after a SETEK command a GET is used to read on the first member of the PDS is accessed.

The ..RC2 code "D" after a GET command signalizes that a directory record has been read.

SCAT (z/OS System Catalog)

Basic Format of the z/OS System Catalog File Definition

```
>>- IPF[n]=SCAT ─┬──────────┬─┬──────────┬─┬──────────┬─ ><
                 └,SCATNM=──┘ └,SDSNM=──┘ └,options──┘
```

`SCATNM=` catalog name
if this operand is missing the whole catalog and all the alias catalogs will be read. This could take a long time! The current catalog name is shown in the FCA.

`SDSNM=` selected data set name
With this operand data sets can be generically selected according to "pattern matching" rules.:

e.g.

```
,SDSNM=SYS1.*
,SDSNM=SYS1.**.PROCLIB
```

options Additional options (that make sense)

`FCA=....` The FCA may be laid into the internal working storage area. If this option is missing the FCA is dynamically allocated and can only be addressed by predefined symbol names.

`VOLID` Only DSN, type and VOLID are returned from the catalog. This will essentially consume less system resources and time.

`DIR` All information available in the catalog is returned.

`FULL` All information available in the catalog, completed with VTOC and VSAM, are returned.

`--` If `VOLID`, `DIR` or `FULL` is missing, information for VSAM is not returned, only from the catalog and from the VTOC.

The FCA for the z/OS system catalog has the following format:

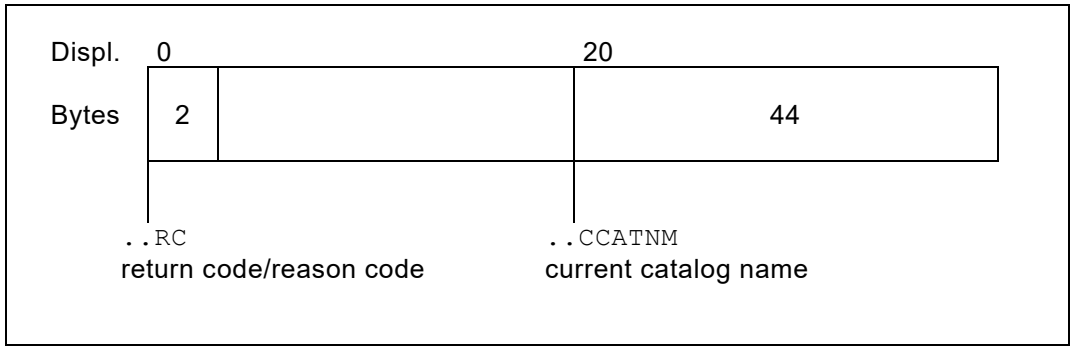


Fig. 181: FCA for z/OS System Catalog

Return code and reason code can be found in the message manual volume 3 under message IDC3009I.

During execution the field `..CCATNM` current catalog name contains the catalog

name that is currently being read.
Data set information is made available according to the following record structure. One record per `GET` instruction.

01-44	DSN data set name	
45-	data set type:	A = Non-VSAM B = Generation Data Group C = VSAM Cluster D = VSAM Data Component I = VSAM Index Component G = VSAM Alternate Index R = VSAM Path H = Generation Data Set U = User Catalog Connector X = Alias
46-51	VOLID oder `??????` (if non-existent)	
52-53	DSORG	PO = PDS, PS = SAM, VS = VSAM
54-56	RECFM	F = Fixed, FB = Fixed blocked, etc.
57-61	BLKSIZE / CISIZE	
62-66	LRECL	
67	Allocation Type	C = Cylinders, T = Tracks, B = Blocks
68-72	Primary Space	
73-77	Secondary Space	
78-80	*reserved*	
81-88	Creation date	YYYYMMDD
89-96	Last referenced date	YYYYMMDD
97-104	Expiration date	YYYYMMDD
105-108	GDG limit	
109-118	VSAM HARBA value	(high allocated RBA)
119-128	VSAM HURBA value	(high used RBA)
129-136	DATACLAS	
137-144	MGMTCLAS	
145-152	STORCLAS	
153-160	Last backup date	

Fig. 182: SCAT record structure

All fields are in character format. If individual fields have a valid content depends on the actual file type.

The catalog name addressed and/or the data set names to be selected may also be dynamically put into the reserved fields `..SCATNM` and `..SDSNM` before the `OPEN`. This requires an explicit file definition.

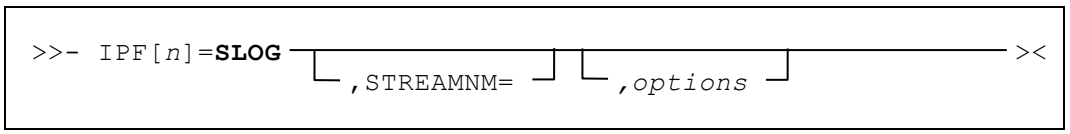
e.g.

```
IPF1=SCAT
...
SET I1SCATNM = 'TEST.CATALOG'
SET I1SDSNM =  'KTEST.*'

OPEN-I1
...
GET-I1
...
```

Fig. 183: Example SCAT

SLOG (z/OS System Logger)

Basic Format of the z/OS System Logger File Definition



STREAMNM=	log stream name the stream name is necessary. It may also be dynamically stored in the reserved field I*n*STREAMNM before the OPEN.
e.g.	`SET I1STREAMNM = 'SYSPLEX.OPERLOG'`
options	additional options
GMT=NO|YES	defines the timestamp being local (NO) or Greenwich mean time (YES). Default is NO.
ACTIVE|ALL	ACTIVE specifies that only active data are returned from the log stream. ACTIVE is the default. ALL specifies that active and inactive data are returned.
start position:	
TIMESTAMP=	yyyymmddhhmmsstt The timestamp parameter defines the start position from where we read. The direction of reading is dependant from additional operands. This parameter should be replaced by TIMESTAMPFROM=. If TIMESTAMP= is used, warning message QPAC182W will be displayed.
TIMESTAMPFROM=	yyyymmddhhmmsstt The TIMESTAMPFROM parameter defines the start position from where we read. The direction of reading is dependant from additional operands.
TIMESTAMPTO=	yyyymmddhhmmsstt The TIMESTAMPTO parameter defines the end position until where we read.
YOUNGEST OLDEST	the youngest data block is read the oldest data block is read Default is OLDEST.
direction:	
OLDTOYOUNG YOUNGTOOLD	reading goes from oldest to youngest data block reading goes from youngest to oldest data block. Default is OLDTOYOUNG.
FCA=....	the FCA may be laid into the internal working storage area. If this option is missing the FCA is dynamically allocated and can only be addressed by predefined symbol names.

`RC=YES`

return code / reason code
If this parameter is specified any return or reason code is returned in its FCA field and processing continues.

Please note: In this case X'0846' and X'0848' have to be explicitly tested for.

The FCA for the z/OS system logger has the following format:

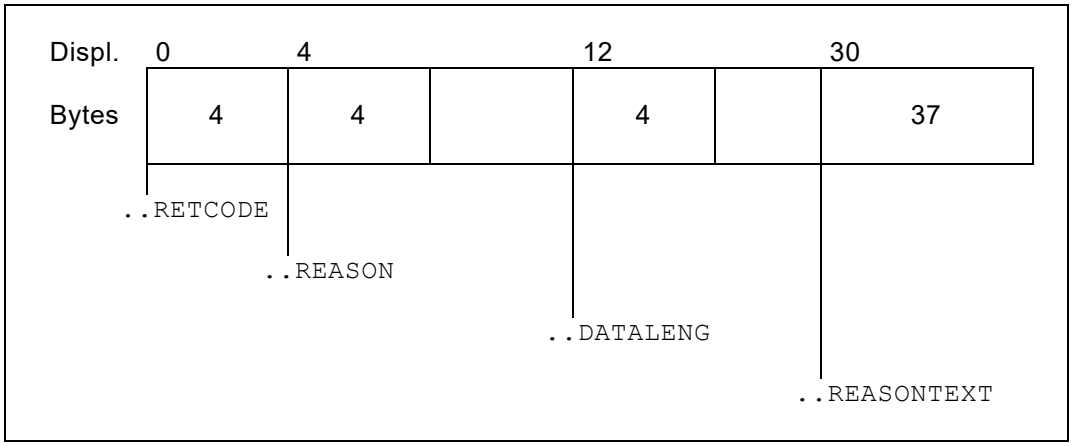


Fig. 184: FCA for z/OS System Logger

Return code and reason code can be looked up in the manual Authorized Assembler Services Reference Volume 2 under IXGBRWSE macro.

The structure of the stream records is dependant from the stream name.

Two reason codes are not handled like errors by QPAC:
The reason code X'0846' means: The log stream is empty and interpreted as EOF, if `RC=YES` has not been defined.
The reason code X'0848' means: No more data exists in the log stream and is interpreted as EOF, if `RC=YES` has not been defined.

```
IPF1=SLOG,ACTIVE,RC=YES,YOUNGEST,OLDTOYOUNG

SET I1STREAMNM = 'SYSPLEX.OPERLOG'

GET-I1  X1+1
IF I1RETCODE = 8 AND I1REASON,CL4 = X'00000848'
  THEN SETIME(WAIT=0500)                    *. WAIT 5 SECS
    IF X1 > 10 THEN GOEND ELSE GOBACK IFEND  *. CONTINUE
IFEND
PRINTR(*)
END
```

Fig. 185: Example 1 SLOG

```
IPF1=SLOG,STREAMNM=SYSPLEX.OPERLOG,GMT=NO,
     TIMESTAMPFROM=2005070117000100
GET-I1
PRINTR(*)
END
```

Fig. 186: Example 2 SLOG

Chapter 11. Integrated Functions (Function Box)

Functions Overview

BINTABS( )	Binary Table Search.
CALENDAR( )	Universal date conversion routine. Converts standard date, Julian date, week, day of week
CHANGEF( )	Changing character strings in files
CHANGER( )	Changing character strings in records
CHANGEW( )	Changing character strings in internal working area
COMPAREF( )	Compares two files
COMPARER( )	Compares two record areas
IDCAMS( )	VSAM catalog functions.
IEBCOPY( )	z/OS utility functions.
PRINTF( )	Print-out of a file in character/hex format
PRINTR( )	Print-out of a record in the record area in character/hex format
PRINTW( )	Print-out of the work area, hiper space or external area in character/hex format
SCANF( )	Scans a file for a given character string
SCANR( )	Scans a record area for a given character string
SCANW( )	Scans a work area for a given string
SEQCHK( )	Sequence check of a file
SETIME( )	Allows the setting of time intervals
SNAP( )	Takes a snapshot of the contents of QPAC fields like symbols, index registers, accumulators etc.
SORTF( )	Sorts input files onto one output file
SORTR( )	Sorts received records and returns them
SORTW( )	Sorts internal work area segments (tables)

Fig. 187: Integrated functions overview

The following specifications apply to all functions:

a) A function is defined by a left parenthesis appended immediately after the key word, which is the function name.

```
COMPAREF(parameters)
```

Fig. 188: Basic format of function routines

b) Parameters within parentheses must be defined without blanks.

c) A function definition together with its parameters must be complete in one statement.
Continuations over more than one statement are not supported.

d) Parameters can be defined or communicated to the function routine in two different ways:

- the definition is directly within the parentheses:

```
COMPAREF(1,5,A)
```

Fig. 189: Parameter specifications for function routines

- the definition can be stored in the internal QPAC work area, and a reference to this area is made within the parentheses. Therefore, the keyword `PARM=` is used:

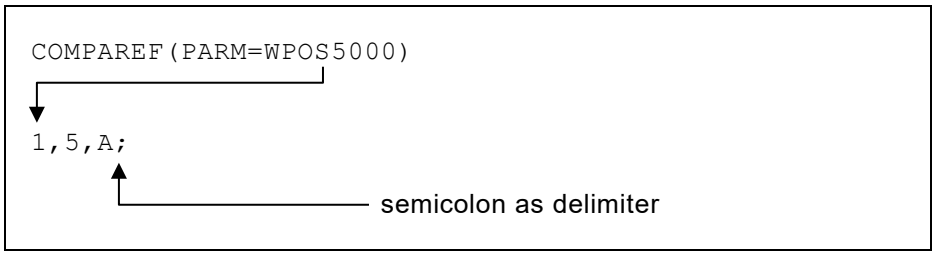


Fig. 190: Parameter specifications in the internal working storage

The second method allows for dynamic build-up and communication of parameters.

e) The parameter definitions are first validated by the function itself, but not until the function is called, and not at QPAC translation time. The end of a set of parameters is marked by a semicolon (;).

f) Generally, there is no predetermined sequence for parameters. However, parameters that depend on each other must be coded in the sequence of their interpretation, left to right. For example, file identification coding must precede record position coding:

```
COMPAREF(IPF2,IPF3,20-80)
```

Fig. 191: Parameter sequence

The relevant information for each function, is described in the detail descriptions for the individual functions.

g) For functions handling entire files, e.g. `COMPAREF()` (compare file), the following rules apply:

- if the input file is opened before calling the function, processing starts with the last record read.
- if the input file is not opened, the function will open it.
- an output file (printer), if not yet opened, will be opened by the function. On function end, the output file remains opened, except if otherwise stated in the detail description of the function.
- input files are closed after function end.

h) Any required printing is normally carried out on the internal system printer using `PUTLST` operations. If the function parameters contain the definition of an output file, printing occurs on the defined output file.

```
OPF=PR       COMPAREF(OPF)
```

Fig. 192: Redirecting print output

i) If the functions find parameter errors at interpretation time, a message is given out and processing ends abnormally. The keyword `NOABEND` prevents the abnormal ending of processing. In such a situation, the function return code will be set to 12, which can later be tested.

```
PRINTF(IPF1,IPF2,NOABEND)
IF FC = 12 THEN ...(any parameter error)
```

Fig. 193: Testing the function return code

j) If functions are used together with internal subroutines, it may be necessary to exchange control information to, and from the functions. In order to do this, a special register `FC` (function code) exists.
`FC` is a 4 bytes binary register, and can be accessed as such.

k) If a file identifier is to be given as a parameter (e.g. `IPF1` or `ODB2`), it is of course possible to use the short form of this identifier (e.g. `I1` or `O2`).

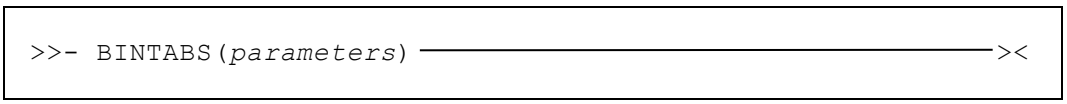
l) The internal processing and result of some function routines are the consequence of a comparison. Examples of this are the compare functions, whereby any inequality is printed.
With all these comparison dependent functions, it is possible to process the available records or areas instead of an in-line subroutine printing the result.

This subroutine steering is defined by the keyword parameter `EQ=`, `NE=` etc., where the subroutine name is included. The individual keyword parameters differ according to the function.

Function routines **must not be called recursively**. Within an entered subroutine the same function routine must not be recalled.

BINTABS( ): Binary Table Search

This function allows large tables in the internal work area or hiper space to be searched. The function expects the tables to be sorted in ascending sequence with fixed element lengths.
The function works according to the principle of finding the “middle”.



The information required by the function is given to it through parameters:

parameters:

[WK=]WPOS*nnnn*:X*n* HPOS*nnnn* *Symbol*:X*n*	Work area *from-to* definition as a dynamic specification of dimensions. The contents of the index register (*to*) is the displacement and is added by the function to the *from* value to determine the end of the table.
RL=*nnn* X*n*	Length of an element in the table, as a direct value or as an index register.
KP=*nn-mm* *nn,l*	Key position within the table element. This sort position may be defined as *from-to* or as *from,length*. It is assumed that the table is sorted in ascending sequence according to those positions.
ARG=WPOS*nnnn*:X*m* *Symbol*:X*m*	Argument to be searched in the table. The length of the argument field must correspond to the length of the key field (KP=). This is assumed. The offset of the element found is returned in the index register.
EQ=Subroutine	Subroutine to be called if the element is found in the table. In this case the index register X*n* contains the offset of the element found relative to the beginning of the table.
NE=Subroutine	Subroutine to be called if the element is not found in the table.

The definition of subroutines is optional. In every case after returning from the function the function register FC contains 0 (null), if the element has been found and 1 if it has not been found.

```
BINTABS(WK=WPOS10001:X1,KP=1-5,ARG=FIELD:X2,EQ=SUBEQ,RL=80
```

The table in the internal work area starting from position 10001 is searched for the value in the argument field `FIELD`.
If the element is found the index register X2 contains the offset relativ to the beginning of the table and the subroutine `SUBEQ` is called. After returning from the subroutine processing is continued behind the function `BINTABS`.

```
BINTABS(TABSTART:X4,KP=5,9,ARG=ARGFELD:X9,EQ=SUBEQ,RL=100)
```

The table begins at the field `TABSTART` and is within the hiper space. It is sorted according to positions 5-13.

```
BINTABS(HPOS1:X5,KP=1-9,RL=50,ARG=ARGUMENT:X6)
IF FC = 0 THEN found IFEND
```

This example shows a query, whether the element does exist within the table or not, without any subroutine being defined.

Fig. 194: BINTABS() examples

CALENDAR( ): Date Conversion

This function allows the date to be easily converted into different formats.

Six main formats are supported:

• standard date	*DDMMYY*
• standard date with century	*DDMM**CC**YY*
• julian date	*YYDDD*
• julian date with century	***CC**YYDDD*
• week format date	*YYWWD*
• week format date with century	***CC**YYWWD*

The **standard date** is the most commonly used format and is available in three different sequences of the elements year, month, day:

- `YMD` for year month day
- `CYMD` for century year month day
- `MDY` for month day year
- `MDCY` for month day century year
- `DMY` for day month year
- `DMCY` for day month century year

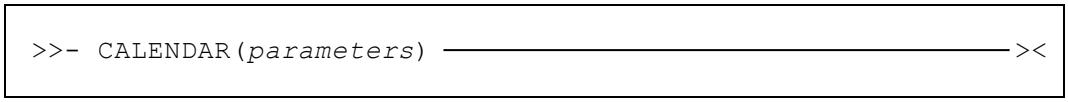
The **Julian date** expresses the days as a number calculated consecutively since January 1 in the format [*CC*]*YYDDD* (*DDD*=day within year).

The **week format** expresses the week as a number, starting each January 1 anew, and a day number within a week, Monday being the first day.
[*CC*]*YYWWD* (WW= week within year, *D*= day within week).

In addition a number of days, months or years can be added to or subtracted from a date.

Also the calculation of the difference between two calendar dates is supported.
In this case the result is written in number of days or as "date duration yyyymmdd" into the output work position (OWP).

All values of the different input, output and arithmetic formats are always an 8 bytes packed field within the work area.



Certain information must be given to the function through parameters:

parameters:

IWP=*datef*=*wpos*

IWP=*datef*=*wpos*
the input work position defines the original format of the date, and the position within the work area, where an **8 bytes packed field** contains the date.

When using the indicator format (IYMD= and IJUL=) the indicator 0 is interpreted as 1900 and the indicator 1 is interpreted as 2000.

datef = date format:

YMD = YYMMDD
MDY = MMDDYY
DMY = DDMMYY
JUL = YYDDD
YWK = YYWWD
CYMD = CCYYMMDD
IYMD = 0IYYMMDD
DMCY = DDMMCCYY
CJUL = CCYYDDD
CYWK = CCYYWWD
IJUL = 0IYYDDD

wpos = work area position

+WP=*datef*=*wpos*
-WP=*datef*=*wpos*

Date arithmetic work area position (optional)
the value of the 8 bytes packed field (wpos) will be added (+WP=) to or subtracted (-WP=) from IWP. The result is written to OWP. The values in wpos can also be negative. Per CALENDAR() function this parameter is only valid once.

When calculating the difference between the IWP date and the -WP date the lower will be subtracted from the higher. If the IWP date is lower then the result with OWP=DAY or OWP=DUR will be negative.

datef = date format:

DAY = number of days
MON = number of months
YEAR = number of years
CYMD = CCYYMMDD (only valid with "-WP=")
DMCY = DDMMCCYY (only valid with "-WP=")

wpos = work area position

OWP=*datef*=*wpos*

The output work area position defines the resultant format of the date and the position within the work area of an **8 byte packed field** for the output.

In addition all reserved CALDR… fields are filled with their corresponding content. This is not true if DUR or DAY has been specified for *datef* .

datef = date format desired (as for IWP), plus additionally:

DAY = difference between IWP-date and –WP-date in number of days

DUR = difference between IWP-date and –WP-date in the format yyyymmdd

wpos = work area position

WP=*datef*=*wpos*

The converted date will be returned into its input field. The format of the converted date is derived as follows:

YMD results in JUL
MYD results in JUL
DMY results in JUL
JUL results in YMD

The routine is a simple exchange procedure converting the date from the common to the julian format, and vice versa.

NOABEND

No abend if the date is invalid.
If the input date is invalid 0 is returned in the output field if NOABEND has been specified.
Additionally, the function code (FC=) is set to 8.

WINDOW=nn|50
WDOW=

If the century is equal 00 or is not specified a window with a year boundary of 50 is taken as the default.
This means that years before 50 are calculated with century 20 and the years after and including 50 are calculated with century 19.
With this parameter, the default value of 50 can be overwritten.

`WINDOW=nn|50`
`WDOW=`

If the century equals 00 or is not specified a window with the year limit set to 50 is assumed by default. This means that the years before 50 are calculated with century 20 and the years before and inclusively 50 with century 19.
With this parameter the default value of 50 may be overwritten.

```
CALENDAR(IWP=YMD=WPOS6000,OWP=JUL=WPOS6010)
```

The date will be converted from format YYMMDD to YYDDD.
Input is in work area pos. 6000 - 6009.
Output in pos. 6010 - 6019.

```
CALENDAR(IWP=JUL=WPOS6010,OWP=YWK=WPOS6900)
```

The date will be converted from format YYDDD to YYWWD.
Input is in an 8 byte packed field starting at work area pos. 6010.
Output in an 8 byte packed field starting at pos. 6900

```
CALENDAR(IWP=YMD=WPOS6000,OWP=DMY=WPOS6000)
```

The date will be converted from format YYMMDD to DDMMYY.
The input and output areas are the same.

```
CALENDAR(IWP=CYMD=WPOS6000,+WP=DAY=WPOS6010,OWP=DMCY=WPOS6020)
```

The number of days at position 6010 are to be added to the date at position 6000. The result is written to position in the format DDMMCCYY.

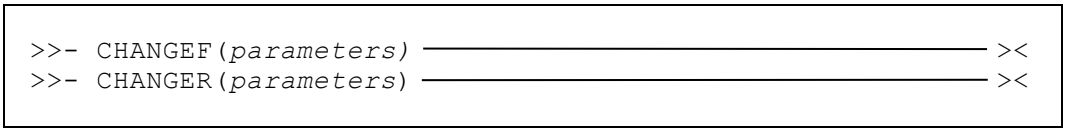
```
CALENDAR(IWP=DMCY=WPOS6040,-WP=CYMD=WPOS6050,OWP=DAY=WPOS6060)
```

The difference between the IWP date and the -WP date is to be calculated. The number of days are written to position 6060 as an 8 bytes field. With „OWP=DUR=WPOS6060“ the result would be stored as date duration (yyyymmdd).

Fig. 195: CALENDAR() examples

CHANGEF( ) / CHANGER( ): Replacing Character Strings

With this function character strings in data files can be searched and replaced. The organization of the data set must allow updating. This means that it must reside on disk and that its organization must be SAM or VSAM.
`CHANGEF()` reads the whole data set until EOF and writes the changed records back.
`CHANGER()` processes only the record currently in the I/O area and does not write it back.



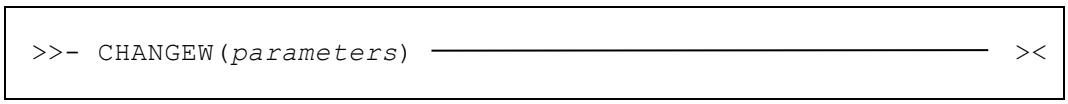
The information required by the function is given to it through parameters:

parameters:

UPF*n*	File ident to be processed. It must be defined as an update file. For testing purposes, the file can also be defined as IPF*n*. This causes any changes being simulated only and not being written back.
DLM='	Delimiter for the character string definitions. An apostrophe is taken as default. Any other character may be defined.
'*oldvalue*'*newvalue*'	Old string to be searched and new string to replace the old one. If the old string is shorter than the new one the following part of the record is shifted to the right. If the old string is longer than the new one the following part of the record is shifted to the left and appended to the new string.
EQ=Subroutine	A subroutine can be defined to be called if the old string is found within the record.
NE=Subroutine	A subroutine can be defined to be called if the old string is not found within the record.
OPF	A printer file ident can be specified. In this case any changes are printed to this print file and not to QPACLIST (SYSLST).

CHANGEW( ): Replacing Character Strings in Work Area-Tables

With this function character strings in work area tables can be searched and replaced. The tables must consist of elements with fixed lengths. `CHANGEW()` starts at the beginning of the table and searches every element for the defined character string and if found replaces it.



The information required by the function is given to it through parameters:

parameters:

[WK=]WPOS*nnnn*:X*n* HPOS*nnnn* WPOS*nnnn*-WPOS*nnnn* WK=*symbol*:X*n*	Work area *from-to* definition as a dynamic specification of dimensions. The contents of the index register (*to*) is the displacement and is added by the function to the *from* value to determine the end of the table.
RL=*nnn* X*n*	Length of an element in the table, as a direct value or as an index register.
DLM='	Delimiter for the character string definitions. An apostrophe is taken as a default. Any valid character may be specified here.
'*oldvalue*'*newvalue*'	Old string to be searched and new string to replace the old one. If the old string is shorter than the new one the following part of the record is shifted to the right. If the old string is longer than the new one the following part of the record is shifted to the left and appended to the new string.
X*n*	X*n* as the first standalone index register contains – when entering a sub routine - the relative offset of the element to the beginning of the table. This parameter is optional.
X*m*	X*m* as the second standalone index register contains – when entering a sub routine – in the EQ (equal) case the offset of the value found relative to the beginning of the element. This parameter is optional.
X*c*	X*c* as the third standalone index register contains – when entering a sub routine – the element number. This parameter is optional.
EQ=*Subroutine*	A sub routine may be defined to be called when the old string has been found within the element.
NE=*Subroutine*	A sub routine may be defined to be called when the old string is not found within the element.

OPF[*n*]	A printer file may be defined. In this case all the changes will be printed to this print file. Otherwise the output is sent to QPACLIST (SYSLST) if no sub routine has been defined.

```
CHANGEW(WK=WPOS10001:X1,RL=020,DLM=/,/Value1/Value2/,
                                          X2,X3,X4,EQ=SUBEQ)
```

The table in the internal work area starting from position 1001 is searched for the value1. If the value1 is found in an element a change to value2 and the call of sub routine SUBEQ occurs. Then the index register X2 contains the offset of the element relative to the table's beginning, X3 contains the offset to the changed value2 and X4 contains the element number.

```
CHANGEW(TABSTART:X4,RL=100,'Value1'Value2',NE=SUBNE,X8)
```

The btable starts at the field TABSTART and resides in the hiper space. The element length is 100. If there are elements where value1 is not found then the sub routine SUBNE is called.

```
CHANGEW(HPOS1-HPOS5001,RL=X5,'Oldvalue'Newvalue',OPF)
```

This example shows a table in the hiper space. Any elements where the value is found are listed to the print file OPF.

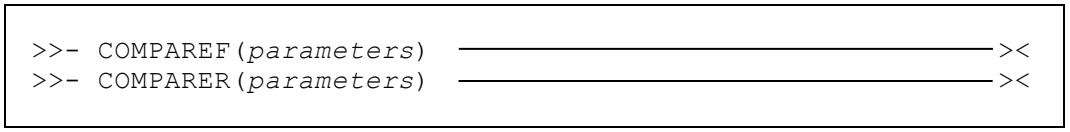
Fig. 196: CHANGEW() examples

COMPAREF( ) / COMPARER( ): Compare Files / Record Areas

This function compares the contents of the data records of two defined files and logs discrepancies on printer output, or transfers control to an inline subroutine.

The function `COMPAREF()` processes the two files in their entirety, up to EOF, including opening and closing. After exiting the function, the files are in a closed status.

The function `COMPARER()` does not itself read the files. It only compares the record areas. This function uses parameters the same way as the `COMPAREF()` function.



Control information can be given to the function by parameters.

parameters:

`(*)`	no specific parameters given. The comparison is made for the entire record length of both files on a 1 to 1 basis.
`IPF`*n*`,IPF`*m*	Defines explicitly which input files are to be compared. If file-id definitions are missing the function itself searches for the first two input files and compares them. The file names that were compared are logged after function end.
`OPF`*n*	Defines the file for print output (`OPF=PR`) if the internal system printer is not to be used. If both `OPF`*n* and any subroutines (`EQ=`, `NE=` etc.) are defined then the subroutines are called and the print output is written. If within a subroutine `NE`, `R1` or `R2` no print output should be written a function code 4 (`FC=4`) can be set.
`SRT=`*s*`,`*l*`,A [,...]` *s*`,`*l*`,D`	Defines the fields on which the files are to be sorted. The function can therefore determine if records are missing. Up to 20 sort fields can be defined. If no `SRT=` sequence specification is defined both files are read one by one.
`NODUPREC`	Records with equal sort keys are suppressed. Only the first record is taken.
from-to `[,...]`	Defines which record segments are to be compared; can be used if the two files have different record lengths but the segments to be compared have identical start and end positions. Up to 20 segment fields can be defined.
`CHR`	Print out only in character format.
`EQ=`*Subname*	When the two records are equal, control is transferred to this subroutine.
`NE=`*Subname*	When the two records are unequal, control is transferred to this subroutine.

R1=*Subname* — When the sequence control shows that only the record from file 1 is available, a branch to this subroutine occurs. If R1 is not defined, control goes to the subroutine defined by NE.

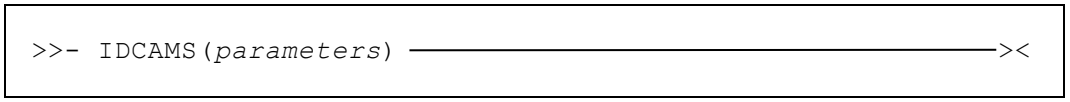
R2=*Subname* — When the sequence control shows that only the record from file 2 is available, a branch to this subroutine occurs. If R2 is not defined, control goes to the subroutine defined by NE, as long as NE is defined.

`COMPAREF(*)`	The first two input files are compared on a one-to-one basis.
`COMPAREF(IPF5,IPF9)`	Input files I5 and I9 are compared.
`COMPAREF(SRT=1,4,A)`	The first two input files are compared. Both files are sorted in ascending order according to positions 1 - 4.
`COMPAREF(SRT=1,4,A,10,2,D,1-80)`	The first two input files are compared. Both files are sorted in ascending order according to positions 1 - 4, then in descending order according to positions 10 - 11. Only positions 1 - 80 are compared.
`COMPAREF(IPF1,IPF3,OPF,CHR)`	Input files I1 and I3 are compared. Print output appears on output file OPF in character format only.
`COMPARER(IPF1,IPF3,OPF,CHR)`	The record areas of input files I1 and I3 are compared. Print output appears on output file OPF in character format only.
`COMPAREF(IPF1,IPF3,NE=SUBNE)`	The two files I1 and I3 are compared. An inequality causes a branch to the routine SUBNE.
`COMPARER(IPF1,IPF3,NE=SUBNE)`	The record areas of the files I1 and I3 are compared. An inequality causes a branch to the routine SUBNE.

Fig. 197: COMPAREF() / COMPARER() Examples

IDCAMS( ): VSAM Catalog Functions

With this function IDCAMS utility functions can be executed directly and integrated from within a QPAC program. The corresponding definition statements are passed to the function from the internal work area.
This function is only available in z/OS.



The information required by the function is given to it through parameters. Therefore two ways are available to pass the statements to IDCAMS:

1. The statements are being passed in form of a table in the internal working storage.
2. The statements are individually passed in an input subroutine. The routine is called as many times until the function code 8 (`FC=8`) is set.

The result from IDCAMS can also be received in two different ways:

1. The output defaults to IDCAMLST if no output subroutine is specified. If the IDCAMLST DD statement is missing it will be dynamically allocated with output class A. Any other output class may be assigned with the parameter `SYSOUT=x`.
2. The output can be received by way of an output subroutine. This subroutine is called per line until the end is reached.

When returning from the function any occurring IDCAMS MAXCC condition code is returned in the function return code field `FC`.

parameters:

`STMT=WPOS`*nnnn* `HPOS`*nnnn*	Work area position where the statements for the IDCAMS function are stored. The statements correspond exactly to the syntax expected by the official IDCAMS utility. The individual statements must be stored in elements with a length of 80 bytes. The end is marked by an /* or a blank.
`SYSOUT=`*x*	If an output subroutine definition is missing an output class may be assigned to the dynamically allocated DD statement.
`ISU=InSubroutine`	Name of the input subroutine. The IDCAMS function is calling this subroutine per individual statement. The statement to be passed is at `WPOS`*nnnn* which is define d by the parameter `IWP=WPOS`*nnnn*. The statements correspond exactly to the syntax expected by the official IDCAMS utility. With a length of 80 bytes. The end is signaled by passing the function code 8 (`FC=8`).
`IWP=WPOS`*nnnn*	Work area position where the statement must be put which must be passed by the input subroutine. It has a length of 80 bytes.

`OSU=OutSubroutine` Output subroutine name. This routine is called per individual result line. The result line is at `WPOS`*`nnnn`* which is defined by the parameter `OWP=WPOS`*`nnnn`*.
The individual line has a length of 121 bytes with leading ASA control character.

`OWP=WPOS`*`nnnn`* Work area position that contains the result line, which is received by the output subroutine, in a length of 121 bytes.

```
IDCAMS(IWP=WPOS7001,ISU=INSUB,OWP=WPOS5200,OSU=OUTSUB)

  SUB-INSUB
    IF X1 = 0 THEN
      SET WPOS7001,CL80 = ' LISTCAT ALL ' X1+1
    ELSE FC=8
    IFEND
  SUBEND

  SUB-OUTSUB
     PUTLST
  SUBEND
```

Fig. 198: Example 1 IDCAMS()

```
SET WPOS7001,CL80 = ' LISTCAT ALL '
SET WPOS7081,CL80 = '/*'

IDCAMS(STMT=WPOS7001,OWP=WPOS5200,OSU=OUTSUB)

  SUB-OUTSUB
     PUTLST
  SUBEND
```

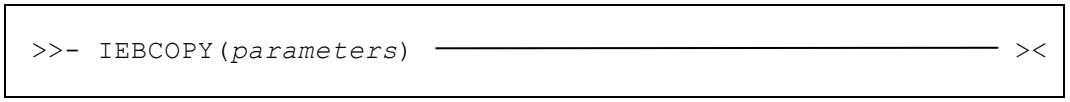
Fig. 199: Example 2 IDCAMS()

```
SET WPOS7001,CL80 = ' LISTCAT ALL '
SET WPOS7081,CL80 = '/*'

IDCAMS(STMT=WPOS7001,SYSOUT=T)
IF FC NOT = 0 THEN ...internal MAXCC code ... IFEND
```

Fig. 200: Example 3 IDCAMS()

IEBCOPY( ): z/OS Utility Functions

This function allows the IEBCOPY utility functions to be directly executed from within any QPAC program. The definition statements necessary are given to the function in the internal working area.
This function is only available under z/OS.



The information required by the function is given to it through parameters.
There are two ways for doing this:

1. The statements are prepared as a table in the working storage area (`STMT=...`).

2. The statements are individually prepared in an input sub routine. This sub routine is called until the function code 8 is set (`FC=8`).

A //SYSIN DD .. statement must NOT be present when using this function. The QPAC program definition statements must be read in via the //QPACIN DD ... statement.

The results from IEBCOPY can also be received in two ways:

1. The output is returned over the SYSPRINT statement if nothing special is defined and a //SYSPRINT DD .. statement does exist.

2. The output can be received through an output sub routine. This sub routine is called per line until the end has been reached. In this case NO //SYSPRINT DD.. statement must exist.

When returning from the function any occurring IEBCOPY return code is returned in the function return code field `FC`.

parameters:

`STMT=WPOS`*nnnn*
`HPOS`*nnnn*
: work area position where the statements for the IEBCOPY function are stored. The statements correspond exactly to the syntax that the official IEBCOPY utility is expecting. The individual statements are to be stored in a length of 80 bytes. The end has to be marked by a /* or blank line.

`ISU=`*InRoutine*
: name of the input sub routine. The IEBCOPY function does call this sub routine per statement. The statement to be passed is at `WPOS`*nnnn*, which is defined by the `IWP=WPOS`*nnnn* parameter. The statements correspond exactly to the syntax that the official IEBCOPY utility is expecting, with a length of 80 bytes.
The end is signalled by setting the function code 8 (`FC=8`).

`IWP=WPOS`*nnnn*
: work area position where the statement given by the input sub routine has to be put, with a length of 80 bytes.

`OSU=`*OutRoutine*
: name of the output sub routine. This routine is called per result line. The result line is at `WPOS`*nnnn*, defined by the `OWP=WPOS`*nnnn* parameter. The individual line has a length of 121 bytes with a leading ASA control character.

`OWP=WPOS`*`nnnn`*

work area position where the result line from the output sub routine is put, in a length of 121 bytes.

`SVC2..`

If the program IEBCOPY is APF protected the SVC of an SVC type 3 routine can be specified with this parameter to temporarily switch to APF mode. See the example of such a routine that comes with the product.

```
IEBCOPY(IWP=WPOS7001,ISU=INSUB,OWP=WPOS5200,OWP=OSU)
SUB-INSUB
  IF X1 = 0 THEN
    SET WPOS7001,CL80 = '   COPYMOD INDD=IN,OUTDD=OUT'
    X1+1
  ELSEIF X1 = 1
    SET WPOS7081,CL80 = '   SELECT  MEMBER=ANYNAME      '
    X1+1
  ELSE FC=8
  IFEND
SUBEND

SUB-OSU
  PUTLST
SUBEND


SET WPOS7001,CL80 = '  COPY INDD=SYSUT1,OUTDD=SYSUT2  '
SET WPOS7081,CL80 = '/*                               '
IEBCOPY(STMT=WPOS7001,OWP=WPOS5000,OSU=OUTSUB,SVC235)

SUB-OUTSUB
  PUTLST
SUBEND


SET WPOS7001,CL80 = '  COPY INDD=INPUTDD,OUTDD=OUTPUTDD'
SET WPOS7081,CL80 = '/*                                 '
IEBCOPY(STMT=WPOS7001)
IF FC NOT = 0 THEN ...internal IEBCOPY return code ... IFEND
```

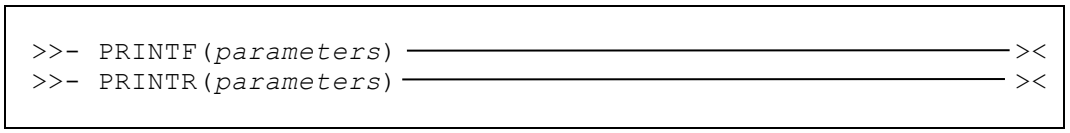
Fig. 201: IEBCOPY() examples

PRINTF( ) / PRINTR( ): Print File / Record Areas

The function `PRINTF()` prints the entire input file, update file or MQS*n* in character/hexadecimal format in an easy to read edited form. The function analyzes the record format considering the file definitions for fixed, variable, or undefined record format. For MQS*n* the GEToptions browse and truncation are internally set.

The function `PRINTR()` prints the current contents of the record area of the input file, which is either defined by parameter, or found by the function itself. The contents of the record area are not changed.

The function `PRINTR()` itself does not read records, but assumes that read commands are issued prior to the calling of the function. This function uses parameters in the same way as `PRINTF()`.



Control information can be given to the function through parameters.

parameters:

`(*)`	no detail parameters given. Print-out is according to default values i.e. the function takes the first input file, update file or MQS*n* it finds in the definition, and prints it in character/hex format.
`IPF`*n* `| UPF`*n* `| MQS`*n*	defines explicitly the file to be printed.
`OPF`*n*	defines the file for print output if the internal system printer (QPACLIST, SYSLST) is not to be used.
`CHR`	print-out only in character format.

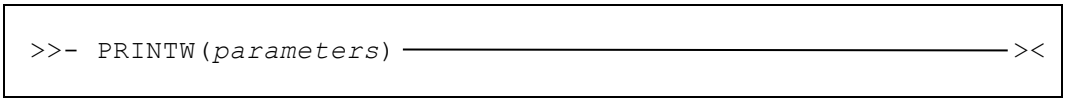
`PRINTF(*,OPF)`	The first input file that is found in the file definitions is printed to `OPF`.
`PRINTF(IPF5)`	Input file `IPF5` is printed
`PRINTF(OPF3)`	The first input file that is found in the file definitions is printed on output file `OPF3=PR`
`PRINTF(I1,CHR)`	Input file `IPF1` is printed in character format only.
`PRINTR(I1,CHR)`	The record area of input file `IPF1` is printed in character format only.

Fig. 202: PRINTF() / PRINTR(*) examples

PRINTW( ): Print Work Area, Hiper Space or External Area

This function prints the contents of the work area, the hiper space or the external area in hexadecimal and character format. It is possible to print all or only a part of the area.

The function does not alter the contents of the work area.



Within the parameter definition a part area can be specified, as can further instructions for the function.

parameters:

(*)	no detail parameters. The whole work area will be printed on the system printer (QPACLIST, SYSLST)
OPF*n*	the output channel OPF*n* should be used instead of SYSLST
[WK=]WPOS*nnnn*-WPOS*nnnn* HPOS*nnnn*-HPOS*nnnn* XPOS*nnnn*-XPOS*nnnn*	print from area position *nnnn* to area position *nnnn*, instead of the whole area
[WK=]WPOS*nnnn*:X*n* HPOS*nnnn*:X*n* XPOS*nnnn*:X*n*	print from area position *nnnn* up to the position defined by index register X*n*. X*n* contains a displacement value, and this value is added to the start value to determine the area end position.

PRINTW(*)	The whole work area will be printed.
PRINTW(OPF,WPOS5000-WPOS9000)	The first 4000 bytes of the work area will be printed on file OPF
PRINTW(HPOS10000:X1)	Index register 1 contains a displacement value which is added to 10000 to give the end limit up to which the work area is to be printed.

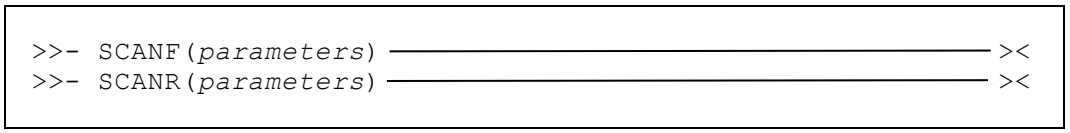
Fig. 203: PRINTW() examples

SCANF( ) / SCANR( ): Scan File / Record Area

The function SCANF() scans an input file for a given character string. Records that contain the string are printed.

The function SCANR() scans the current contents of the record area of an input file, the file being either previously defined or found by the function itself. The contents of the record area will not be altered.

The function SCANR() itself does not read records, but assumes that a read command has been executed. This function corresponds to the SCANF() function in its use of parameters.



Control information must be given to the function through parameters.

parameters:

DLM='	delimiter character This parameter is optional and defines the string delimiter character. Default is ' (apostrophe).
IPF*n*	explicitly defined input file to be scanned. If missing, the function itself searches for the first file in the file definitions.
'...' X'...'	character or hex string between the delimiters, to be used for the scan. String size cannot exceed 100 characters. If a hex string is defined the parameter DLM= is invalid.
from-to [,...]	record scan area. String is only scanned for in the record area delimited by the from and to positions.
CAPS=OFF|ON	When CAPS=OFF is defined the text is searched for lower- and upper-case letters. If the scan string is defined as a hex string CAPS=ON is automatically set. Otherwise, the text is searched as defined. CAPS=OFF is the default.
CHR	records containing the string are printed out in character format only.
NOPRINT	records are not printed out, but as in all cases, the total number of records fulfilling the scan condition is output after the function exit.
EQ=*Subname*	control is passed to this subroutine if the sought-after string is found. The first index register, if specified, contains the offset to that string.
X*nn*	the first index register, if specified, contains the offset of the search argument.
X*nn*	if the string is found more than once a second index register (counter) contains the number of hits.

NE=*Subname*	control is passed to this subroutine if a record is found not containing the defined search value.

`SCANF('CHARACTERS')`	Scans the first found input file for the value CHARACTERS.
`SCANF(DLM=*,*CHARACTERS*)`	Same as above, except the delimiter character is specified as * .
`SCANF(IPF5,'CHARACTERS',CHR)`	Input file `IPF5` is scanned for the value CHARACTERS. Found records are printed out in character format only.
`SCANF(IPF5,'CHARACTERS',NOPRINT)`	Same as above but records are not printed. (The total number of found records is output after function end).
`SCANF(DLM=:,:CHARACTERS:,20-80)`	Only positions 20-80 of each record are scanned.
`SCANR('CHARACTERS',EQ=SUBR,X9,X10)`	A branch to the routine ROUTINE occurs for every record that contains the term CHARACTERS. Index register `X9` contains the offset.

Fig. 204: SCANF() / SCANR() examples

SCANW( ): Scan Work Area Table

This function scans the elements of a table which exists in the working storage area. The contents of the area will not be altered.

```
>>- SCANW(parameters) ──────────────────────────────────────── ><
```

parameters:

DLM='	delimiter character, Default is ' (apostrophe)
[WK=]WPOS*nnnn*-WPOS*nnnn* WPOS*nnnn*:X*n* *field1-field2* *field1*:X*n*	table area in working storage, hiper space or single field. The table end may be specified by an index register which contains the offset.
'...' X'...'	character or hex string between the delimiters, to be used for the scan. String size cannot exceed 100 characters. If a hex string is specified, the parameter DLM= is invalid.
RL=*nnnn*	table element length
CAPS=OFF|ON	When CAPS=OFF is defined the text is searched for lower- and upper-case letters. If the scan string is defined as a hex string CAPS=ON is automatically set. Otherwise, the text is searched as defined. CAPS=OFF is the default.
CHR	print output in character format
EQ=*Subname*,Xn_1,Xn_2	control is passed to this subroutine if the sought-after string is found. The first index register contains the element number, and the second index register contains the offset within the element that contains the searched string.
NE=*Subname*,Xn_1	subroutine to be branched to if a record NOT containing the string is read.
OPF*n*	printer output

```
SCANW('CHAR',WK=WPOS1-WPOS1000,RL=100)
```

Searches the table elements with a length of 100 for the string CHAR.

```
SCANW(DLM=*,*CHARACTERS* ...)
```

The asterisk is the delimiting character.

```
SCANW('CHAR',EQ=SUBR,X1,X2 ...)
```

The subroutine SUBR will be branched to every time an element is found containing the string CHAR.

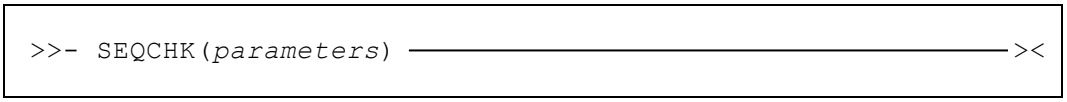
Fig. 205: SCANW() examples

SEQCHK( ): Sequence Check

This function reads the entire input file and checks the sequence. The necessary information to process this function is obtained from parameters.

The function opens the file (if not already open) and processes it up to end of file.

Sequence errors are reported by a printout of the two records in error, the last read record forming the basis for continuation. Records with **equal** sort values are considered to be valid.



Various elements of control information must be given to the function by parameters:

parameters:

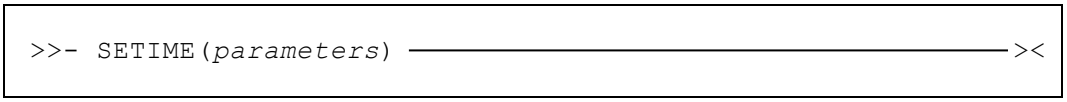
IPF*n*	explicit specification of the file to be checked. If this definition is present, it must precede the sort parameters.
SRT=*s*,*l*,A [,....] *s*,*l*,D	This parameter is compulsory. It defines the fields on which the sequence check must operate. s = sort field position within the record l = length of the field in bytes A = ascending sequence order D = descending sequence order Up to 5 fields can be defined.
OPF*n*	Defines the output file (printer) for error records, replacing the internal system printer.
NOPRINT	Print-out of error records is suppressed. In all cases the number of found errors is reported at function end.
EQ=*Subname*	A branch to this routine occurs if two records with the same sort criteria are found.
LT=*Subname*	A branch to this routine occurs if the sort criteria of the first record is less than that of the second record.
GT=*Subname*	A branch to this routine occurs if the sort criteria of the first record is greater than that of the second record.

`SEQCHK(SRT=1,4,A)`	The first input file that is found in the file definitions is sequence checked in ascending order according to the positions 1 - 4 of the record.
`SEQCHK(IPF5,SRT=10,3,A,2,5,D)`	Input file IPF5 is sequence checked: - position 10, length 3, ascending - position 2, length 5, descending
`SEQCHK(SRT=15,9,D,NOPRINT)`	The first input file that is found in the file definitions is sequence checked in descending order according to the positions 15 - 23 in the record. Error records are not printed out.
`SEQCHK(SRT=1,4,A,EQ=ROUTINE)`	Records are tested to see if they have the same sort criteria.

Fig. 206: SEQCHK() examples

SETIME( ): Set Time Interval

This function can set a time interval, after which processing jumps to a pre-defined routine, or for the duration of which, processing remains in a waiting state.



Certain control information must be given to the function through parameters:

Parameter:

INTERVAL=*nn* IV=*nn*	This defines the time interval. Processing continues after the interval. Current processing will be stopped, and processing in the pre-defined subroutine will commence. After leaving the subroutine, processing recommences at the interrupted point.
WAIT=*nn*	This defines the time interval. Processing halts and waits until the end of the interval. Processing then resumes within the pre-defined subroutine (if defined). After leaving the subroutine, processing continues with the instructions directly following the function. In z/OS the time interval is defined in **hundredths of seconds**.
SU=*Subname*	Defines the in-line subroutine to which processing jumps when the time interval has elapsed. The definition of the subroutine is optional. However, IV= without a subroutine is meaningless.

SNAP( ): Snapshot of QPAC Fields and Registers

With this function it is possible to take a snap shot of workfields at any time of QPAC execution. This information is very useful for tracing and problem analysis. In case of a program check QPAC automatically performs this function.

```
>>- SNAP(parameters) ------------------------------------------><
```

The information required by the function is given to it through parameters:

parameters:

(*)	no detail parameters given. All types of fields that are used and have a value not equal to zero are printed on the system line printer.
OPF*n*	defines the file for print output if the internal system printer (QPACLIST, SYSLST) is not to be used.
XR	print only the contents of all used index registers whose contents is not equal to zero.
ACCU	print only the contents of all used accumulators whose contents is not equal to zero.
IO	print only the I/O areas of all used file definitions whose contents is not equal to zero.
SYM	print only the contents of all used symbols.
SYM=*field1/field2* =*field** =*f+eld1*	print only the contents of the field symbols specified. * represents one or more positions not to be tested + represents one position not to be tested

```
SNAP(XR,ACCU,IO)


          INDEX REGISTER DISPLAY
          ALL ZERO

          ACCU FIELDS DISPLAY
          ALL ZERO

          I/O AREAS DISPLAY
I00 001 = D8D7C1C340E3C8C540D6D5C540C1D5C440D6D5D3E8404040
    051 = E2D6C6E3E6C1D9C540D7C1C3D2C1C7C540C6D9D6D440
O00 001 = D6E2E8E240C7D9C1D5C440D3E4C5404040404040404040
    051 = 40404040404040404040404040404040404040404040404040
    101 = 40404040404040404040404040404040
```

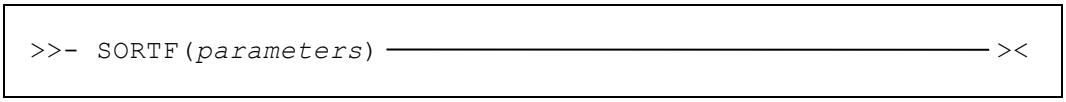
Fig. 207: Sample SNAP() output

SORTF( ): Sort File

This function sorts one or more input files into one output file during the execution of a QPAC program.

The function internally uses the IBM SORT program (or the compatible sort program of other manufacturers).

The input and output files are opened and closed by the function. After function execution they are in closed status.



A minimum of necessary control information must be given to the function by parameters.

parameters:

`IPF[n], UPF[n]` explicitly defines the input files to be sorted. If this parameter is missing, the function uses all the input files.

`OPF[n]` defines explicitly the output file for the sorted file. If this parameter is missing, the function itself searches for the first defined output file.

`SRT=s,l,A [,...]`
defines the fields on which the SORT must operate. This parameter cannot be omitted.
If no other parameters are defined, the following occurs:

- all present input files are assumed to be input files for the SORT
- the first output file definition is assumed to be the file definition for the sorted output file
- the format of the SORT fields is assumed to be `BI` (binary)
- the number of work files is 1

`SRT=s,l,f,A [,...]`
`s,l,f,D`
sort field definitions can contain field format attributes *f* which replace the default format `BI`.
Valid format attributes are:
`CH,ZD,PD,FI,BI,FL,Y2B,Y2C,Y2D,Y2P,Y2S,Y2Z.`

`NODUPREC` Records with equal sort keys are suppressed. Only the first record is taken (SUM FIELDS=NONE).

`MSG` indicates that all the usual SORT messages should be issued on SYSOUT. Per default only error messages are issued.

`NOABEND` the function will normally terminate once an error has occured. To avoid any early break in processing, the `NOABEND` parameter can be given, whereby, after an error has occurred, the function returns to its CALL point and sets the return code to 'not equal' 0. The return codereturn code can then be examined,

```
IF RC NOT = 0 THEN
```

and the function reCALLed with altered parameters.

`WORK=`*n* — it is assumed that no external work files are necessary, as the input can be sorted within main storage. However, when a large file is input, the number of required SORTWK*n* files can be defined by the `WORK=`*n* parameter. A corresponding number of DD statements (SOTWK1-9) must also be present.

`WORKNM=`*xxxx* — when work files are used, the default file name is SORTWK*n*. The `WORKNM=`*xxxx* parameter enables the first 4 characters of the file name to be altered:

```
WORKNM=QPAC
```

results in QPACWK1, QPACWK2 etc.

`Y2PAST=`*yyyy* — this parameter defines the actual century window. See the corresponding SORT literature.

`SORTF(SRT=1,3,A)`	All input files found by the function are sorted and output onto the first found output file.
`SORTF(SRT=1,3,PD,A)`	Same as above, but field format is packed decimal.
`SORTF(OPF5,SRT=10,4,D,1,5,A)`	All input files are sorted onto output file `OPF5`
`SORTF(I1,I5,O5,SRT=7,9,CH,A)`	The input files `I1` and `I5` are sorted onto output file `OPF5`. After the sort, `I1`,`I5` and `OPF5` are in a 'closed' state.

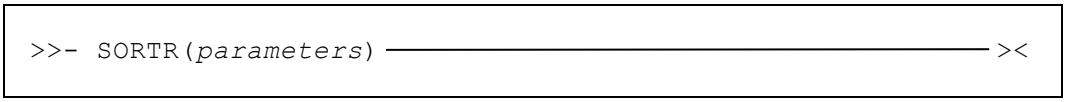
Fig. 208: SORTF() examples

SORTR( ): Sort Records

This function sorts records given to it by an internal subroutine. After the sort, the records are returned using another internal subroutine.

The function uses the IBM SORT internally (or the compatible sort of other manufacturers).

The function itself does not read the records but assumes that they will be supplied. The sort parameter rules are the same as for `SORTF()`.



The required control information must be given to the function through parameters.

parameters:

`ISU=`*name*	name of the input subroutine which will fetch the records for the sort, until `FC` (function code) = 8, which indicates that no further records will be supplied. The sort then takes place. Within the input subroutine no `GO....` commands are permitted. The return to the sort program assumes a valid `SUBEND`. To cancel any sort process the function code can be set to 16 within the input subroutine (`FC=16` instead of `FC=8`). After this no sorting is done and the `SORTR` function is left. `FC=16` remains and may be tested.
`OSU=`*name*	name of the output subroutine which accepts the records after the sort. This routine is called again and again, until all records are given back.
`SU=`*name*	instead of `ISU=` and `OSU=`, a common routine can be defined. The user must therefore control the input/output cycle in the subroutine himself.
`IWP=`*nnnn*	input work area position in which the function expects the records after the input routine is completed.
`OWP=`*nnnn*	output work area position into which the function returns the records for the output routine, after the sort.
`WP=`*nnnn*	instead of `IWP=` and `OWP=` a common area can be defined.
`RL=`*nnn*	length of the records to be sorted.
`RL=V`*nnn*	maximum length with variable record lengths
`NODUPREC`	see under `SORTF()`.
`SRT=`*s*`,`*l*`,A,...` *s*`,`*l*`,D` *s*`,`*l*`,`*f*`,A` *s*`,`*l*`,`*f*`,D`	sort field definitions The format of these definitions corresponds to those of SORT, with or without the field format attribute *f*. If no field format attribute is given, format `BI` will be assumed. Valid format attributes are : `CH,ZD,PD,FI,BI,FL,Y2B,Y2C,Y2D,Y2P,Y2S,Y2Z`.

`MSG`	this parameter specifies that SORT messages will be output via SYSOUT.
`WORK=`*`n`*	number of SORTWK files (see `SORTF()`).
`WORKNM=`*`xxxx`*	changes the first 4 positions of the SORTWK file name (see `SORTF()`).
`Y2PAST=`*`yyyy`*	This parameter defines the actual century window. See the corresponding SORT literature.

```
SORTR(ISU=INPUT,OSU=OUTPUT,WP=WPOS7001,RL=800,SRT=1,3,A)
...
   SUB-INPUT
      ...
       move record to position 7001
       or set  FC=8  (no following records)
       ....
   SUBEND

   SUB-OUTPUT
      ...
      read record from position 7001 and processing
      ...
   SUBEND
```

Fig. 209: SORTR() example with two defined subroutines

```
SORTR(SU=ROUTINE,WP=WPOS7001,RL=50,SRT=1,3,PD,A,MSG)
...
   SUB-ROUTINE
      ...
      IF X1 = 1 THEN
         ...                (output phase)
      ELSE
         ...                (input phase)
         ...
         FC=8  X1=1         (end of input)
      IFEND
   SUBEND
```

Fig. 210: SORTR() example with one defined subroutine

SORTW( ): Sort Work Area

This function will, for example, sort tables in the internal work area or in the hiper space.

```
>>- SORTW(parameters) ------------------------------------------------ ><
```

The required control information must be given to the function through parameters.

parameters:

[WK=]WPOS*nnnn*-WPOS*nnnn* HPOS*nnnn*-HPOS*nnnn*	*from-to* area definition, delimiting the area to be used. The *to* address element is not included in the sort.
[WK=]WPOS*nnnn*:X*n* HPOS*nnnn*:X*n*	*from-to* area definition - dynamically defined. This defined area will be sorted. When an index register limit is given, the contents of the register are taken as a displacement and added to the *from* value. The *to* address element is not included in the sort. The key word WK= is optional.
RL=*nnn* X*n*	length of one element in the area, as a direct value or as an index resgister.
SRT=*s*,*l*,A *s*,*l*,D	sort field definitions - up to 5 fields can be given. The field attribute is not allowed.
NODUPREC	Duplicates will be overwritten with Hex'FF' and be sorted to the end of the work area. A possible index register will be decreased accordingly.

```
SORTW(WPOS6000-WPOS7000,RL=10,SRT=1,8,A)
```

The fields in the internal work area starting at address 6000 are sorted in ascending order.

```
SORTW(WPOS5000:X1,RL=80,SRT=5,10,D)
```

Records with a length specified by X1 starting at address 5000 in the internal work area are sorted in descending order according to positions 5 to 14.

```
SORTW(HPOS1:X1,RL=80,SRT=5,10,A,1,4,D,50,1,A)
```

Records with a length specified by X1 starting at address 1 in the hiper space are sorted. Positions 5-14 in ascending order, 1-4 in descending order and position 50 in ascending order.

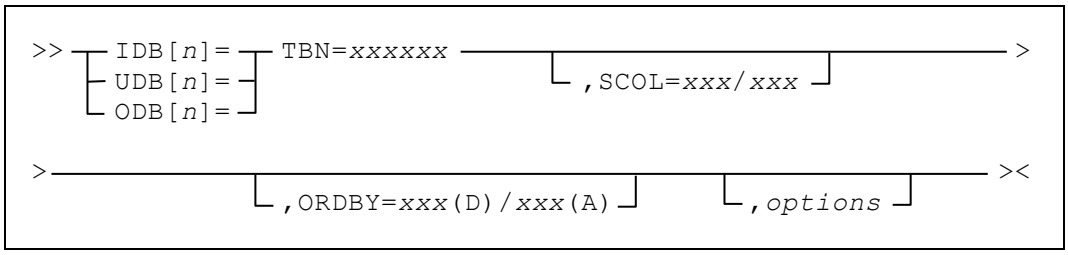
Fig. 211: SORTW() examples

Chapter 12. DB2 Support Feature

DB2 Data Base Definition

IBM DB2 is fully supported by this feature. It assumes, that DB2 is installed, and that there is access to the DB system.

General Format of DB2 DB Definition



TBN= Is the table or view to be processed.
The follwoing items may be specified:

- Location (for distributed databases)
- Creator
- Table name

```
IDBn=TBN=LOCATION.CREATOR.TABLENAME
```

Fig. 212: Unique specification of a table object

The table name can also be a synonym.
This definition must be the first operand. This tells QPAC that this data base is to be processed by DB2.

SCOL= selected column names.
Thereby, the required columns from the table row can be selectively processed.
It functions as though only these columns are present. Column names are separated from each other by a slash:

```
SCOL=COLUMN1/COLUMN10
```

Fig. 213: Selection of multiple columns

If there is not room for all the names within the statement, continuation on the following line is achieved by the definition of a slash followed by a space in the current statement. The next column name is then defined in the following statement:
Leading blanks are allowed.

```
SCOL=COLUMN1/
     COLUMN10
```

Fig. 214: Selection of multiple columns (cont.)

SCOL= can only be defined with IDB or UDB.

With ODB, the whole row must always be processed.

ORDBY= ORDER BY clause.
This defines the order into which the rows or selected columns are to be sorted. In this sort sequence, the rows are made available. Column names are separated from each other by a slash.

```
ORDBY=COLUMN1/COLUMN10
```

Fig. 215: Sort of rows and selected columns

If there is not room for all the names within the statement, continuation on the following line is achieved by the definition of a slash followed by a space in the current statement.
The next column name is then defined in the following statement. Leading blanks are allowed.

```
ORDBY=COLUMN1/
      COLUMN10
```

Fig. 216: Sort of rows and selected columns (cont.)

A column can also be sorted in descending order. The type of sort can be defined in brackets following the column name:

COLUMN1(**D**)/ means descending
COLUMN2(**A**)/ means ascending.
The default is (A).

```
ORDBY=COLUMN1/COLUMN10(D)/COLUMN20
```

Fig. 217: Sortieren von Columns aufwärts oder abwärts

ORDBY= can only be defined with IDB.

options The following additional options are available:

WP=*nnnnn* Work area position.
This definition states that the row should be written into, or read from, the general work area. The definition refers to the place within the work area that is to be overlaid by the row to be processed. If WP= is defined, there is no dynamic record area for the corresponding file definition.

FCA= File communication area.
This defines which position within the general QPAC work area should contain the information exchange area for the respective file definition. The default form of FCA definition is dynamic.

With the FCA the following implicit symbols are predefined and can be used as required:

```
..SQLCODE,BL4      = return code
..ROWLEN,BL4       = length of the row read
..TBN,CL18         = table name
```

`RC=YES` Return code.
With this definition, any error from DB2 will not result in a QPAC processing halt, but the original SQLCODE in the FCA field `..SQLCODE` will be transferred to the program.

```
-927
```

Fig. 218: SQL return code in FCA field ..SQLCODE

If there is no error, the return code in the FCA field `..SQLCODE` is 0.
It is important to check the return code in the FCA within the application, if this operand is defined.

```
IF I1SQLCODE = -927 THEN
IF I2SQLCODE NOT = 0 THEN
```

Fig. 219: Examining the SQL return code

`PRFX=YES` Prepare for symbol prefix.
This option can be defined when different tables with the same column names are given.
Thereby, to create a field symbol, the file identification (short form) is placed in front of all column names.
This avoids the 'duplicate symbol' problem.

CAF Support instead of the TSO Batch Program IKJEFT01

The PLAN name and the DB2 system id may be specified with a `PARM=` definition. This will internally initialize **CAF support** instead of using the TSO batch program IKJEFT01.

`PLAN=`*`planname`*
Plan name if QPAC is not called under the TSO batch program IKJEFT01.

`DB2ID=`*`sysid`* DB2 system id if QPAC is not called under the TSO batch program IKJEFT01.
See also under DB2 job control example.

CAF Return Codes and Reason Codes

CAF return and reason codes are to be found in the IBM manual *Application Programming and SQL Guide* under "CAF Return Codes and Reason Codes".

Usage of DB2 Data Bases

In **z/OS** the processing by QPAC is based on a DB2 DBRM called QPACBDB2, an application plan and the QPAC interface QPACBDB2.
The interface QPACBDB2 must be defined as a program within the plan. The tables can then be processed.

QPAC works on the basis of **dynamic SQL**. But, to a great extent, it finds its own bearings.

Within z/OS up to 39 DB2 file definitions can be made, and they must have either a file identification 1-39, or they must be defined by the implicit format.
Please note: Identification numbers 1-9 are declared **without "WITH HOLD"** and the implicit format and identification numbers 10-39 are declared **with "WITH HOLD"**.

```
IDB=     ODB=     UDB=
IDB1=    ODB8=    UDB3=
```

Fig. 220: DB2 file definitions

QPAC automatically defines and associates internally the original column names as field names, together with the respective lengths and format attributes.
These field names are attached in the order that they are present in the I/O area, as the individual columns in a row are defined.
Variable fields (VARCHAR, LONG VARCHAR) are defined with a **leading 2 byte binary field** in the I/O area. The column name, however, points to the beginning of the field. The leading 2 byte length field has its own name associated with it, consisting of the column name, followed by a **hash sign** (#).

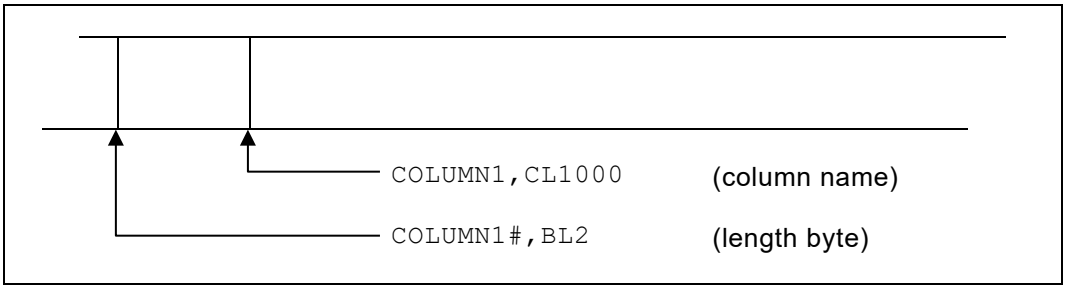


Fig. 221: Variable fields with 2 bytes length field

These automatically associated field names can be referenced in the processing.
The relative associations can be seen on the XREF cross-reference list.

Columns with attribute NULL YES, i.e. they can be NULL, have an additional field association:
A dynamically associated 2 byte binary field, consisting of the column name followed by an **at sign (@)**, is associated as a field name, and can be used to control the NULL condition. If this indicator field is set to -1 after a read operation, the content of the column is taken as NULL. If this indicator field is set to -1 before an output operation, the content of the column is output as NULL.

```
SET COLUMN10@ = -1
IF COLUMN10@ = -1
   THEN                              (the column content is NULL)
```

Fig. 222: Handling the NULL state

If the indicator field (@) for a column is set to -1 (NULL) following a read, and the column field contains a value, then the indicator field must be set to 0, in order to indicate a NOT NULL status. Otherwise, the NULL status would be present if a `REWRITE` were to be processed.

```
SET COLUMN10@ = 0
```

Fig. 223: Releasing the NULL state

Floating Point

Columns with the FLOAT attribute (floating point, short or long precision) can be processed by QPAC.

a) A field with the FLOAT attribute will automatically be converted into a displayable format by the `SET` instruction, and then be put into a character field.

```
SET WPOS7000,CL20 = COLUMN_FLOAT
```

Fig. 224: Processing of SQL floating point fields

The result is a floating-point number in the following, left justified, format:

```
+0.5925E+02
```

Fig. 225: Display format of an SQL floating point number

b) If a field with the FLOAT attribute is to be filled with a `SET` instruction, the sending part is a character field, containing a floating-point number, or a character literal with the format of a floating point number.
The floating point number in character format will automatically be converted and stored into the column field.

```
SET COLUMN_FLOAT = WPOS7000,CL20
SET COLUMN_FLOAT = +0.5925E+02
SET COLUMN_FLOAT = '+0.5925E+02'
```

Fig. 226: Displaying SQL floating point numbers

There is no further support of floating-point arithmetic within QPAC.

If the same table is stated in more than one file definition, a 'duplicate symbol' situation occurs, due to the automatic field symbol association of column names. The same thing would happen if the same column names were present in different tables.

This situation can be avoided by defining the operand `PRFX=YES` in the file definition. With this definition, each column name which is associated as a field symbol, is preceded by the file identification (short form). The symbol names are thereby clearly identified and unique.

```
IDB1=TBN=TABLE1,PRFX=YES
IDB2=TBN=TABLE1,PRFX=YES

IF I1COLUMN1 = ...
IF I2COLUMN1 = ...

  or

IDB1=TBN=TABLEX
IDB2=TBN=TABLEX,PRFX=YES

IF COLUMNX = ...
IF I2COLUMNX = ...
```

Fig. 227: PRFX=YES prevents from "Duplicate Symbol" situations

This prefix association is only for field symbols, and not for other column definitions, e.g. `SCOL=`, `ORDBY=` or `WHERE-D`*`n`* conditions etc.

The purpose of the FCA is for the exchange of any necessary information between the processing program and DB2. It is therefore mainly concerned with return codes.

The FCA for DB2 has the following format:

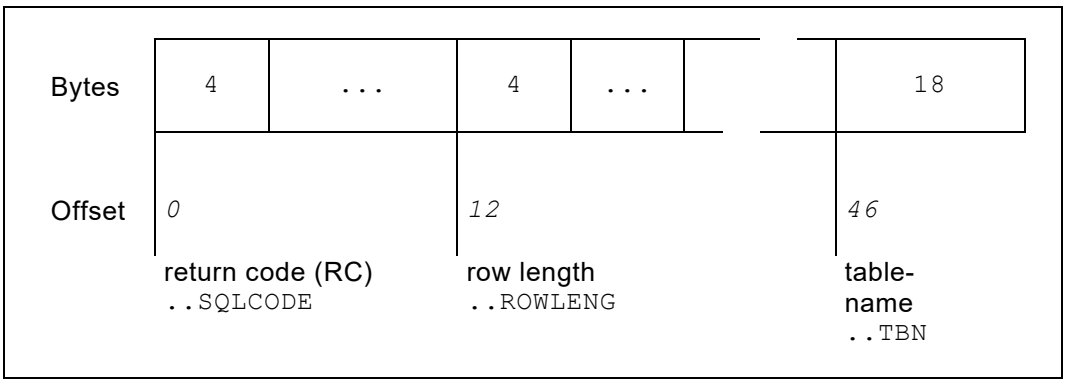


Fig. 228: FCA for DB2

RC	DB2 return code in binary format as a negative value. e.g. `-927` This return code is only placed in the FCA if the operand `RC=YES` is defined in the file definition. Otherwise, an SQL error automatically causes a processing halt. This would be the normal case. The field has the symbol `..SQLCODE,BL4` assigned to it.
row length	This is the length of the complete row, (all the columns together in the I/O area), including any length bytes for VARCHAR columns. The field has the symbol `..ROWLENG,BL4` assigned to it.
table name	This is the name of the table to be processed (see also `TBN=`). The field has the symbol `..TBN,CL18` assigned to it.

Basically, a table (or individual rows from a table), can be read (`IDB` with the `GET` command), modified (`UDB` with the `GET - PUT` command sequence), or written (`ODB` with the `PUT` command).

Additions of rows to, or **deletions** of rows from an existing table, are allowed in update mode (`UDB`).

Processing is carried out sequentially row-wise, in the order DB2 supplies them, as long as the ORDER BY clause (`ORDBY=`) is not defined.

The individual columns of a row are made available in a row together in the I/O area. The fields can be addressed using their column names. It is also possible, (but not necessarily recommended), to address the individual fields directly, e.g. using implicit symbol association:

```
I1POS1,CL80
```

Fig. 229: Not recommended: direct addressing of columns

If only `GETs` are used (`IDB` definition), processing is from the beginning to the end of the table. This corresponds internally to the following function sequence:

DECLARE .....,
OPEN table ..,
FETCH ...,
CLOSE table.

If instead of `IDB` (input DB2 data base table) `UDB` is defined (update data base table), the command sequence `GET PUT` corresponds to the following function sequence:

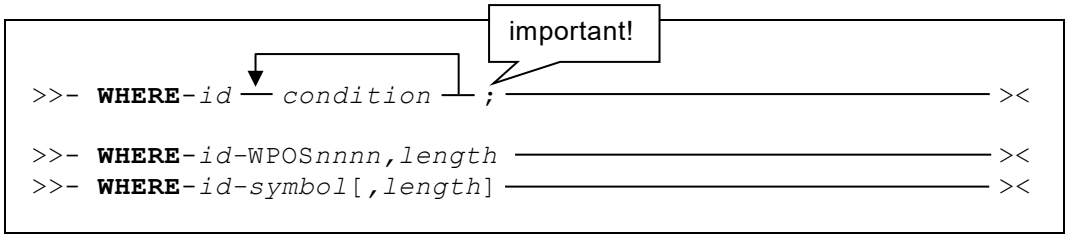
DECLARE ... FOR UPDATE OF ...,
OPEN ....,
FETCH ....,

UPDATE .... WHERE CURRENT OF .....,

CLOSE ...

The WHERE Instruction

A table can be prepared for **selective processing** by using the command `WHERE`. This corresponds in meaning to the WHERE clause in a SELECT statement for an SQL definition.
The `WHERE` instruction has the following two formats:



The conditions must have the exact format of the WHERE clause in the SELECT statement. A **semi-colon (;)** defines the **end of all the condition definitions**.
The condition definitions are internally accepted in the defined format, and are syntax checked by DB2 itself. An exception to this are host variables that are possibly present in a condition.
A **host variable** is originally identified by a preceding **colon (:)**. Any host variables present will have their content resolved when the command is executed. The host variable itself must be a defined field at the time the program is translated (syntax checked). Otherwise, the error 108 (undefined symbol) will occur.

The second format allows the WHERE conditions to be dynamically put into the defined field symbol or area during execution. These conditions must **NOT contain any host variables**. They can no longer pass any syntax analysis at this moment. Also, the terminating semicolon (;) is not allowed.

The `WHERE` command is a preparatory command. If more than one row is present, those fulfilling the selection criteria are made available individually by the following `GETs`.

```
WHERE-I1 COLUMN10 = 'name' AND
         COLUMN5  > :X28 ;

NORMAL
GET-I1   (only rows whose columns fulfill the
         above conditions appear here)
...
```

Fig. 230: Usage of the WHERE instruction (example 1)

```
WHERE-I1 COLUMN10 = 'name' AND
         COLUMN5  > :X28 ;

DO-FOREVER
   GET-I1   AT-EOF DOQUIT ATEND
            (only rows whose columns fulfill the
            above conditions appear here)
   ...
DOEND
```

Fig. 231: Usage of the WHERE instruction (example 2)

```
0S=WHEREAREA,CL100
SET WHEREAREA = 'COLUMN10 = "name" AND '/
                'COLUMN5 > 116          '
...
WHERE-I1-WHEREAREA
```

Fig. 232: Usage of the WHERE instruction (example 3)

The `WHERE` command contains an `OPEN` if the current file state is "closed".

The FETCH Instruction

Instead of the `GET` instruction also the `FETCH` instruction may be used. The difference is that in case of an EOF situation no automatic close is done and no GET-Block is skipped.

```
>>- FETCH-Id ──────────────────────────────────────────────────────────><
```

The PUTA Instruction

With the `PUTA` command (put addition) a **complete row** can be **inserted** into a DB2 table. The individual columns must be filled with data in the I/O area, prior to the `PUTA`, and must conform to the format and convention of DB2. Null values must be identified with the column name@ and a value of -1.

For VARCHAR fields, the actual string length must be given in the field column name# (2 bytes binary). The maximum field size is always assigned in the I/O area. A column following a VARCHAR field is therefore always placed next to the maximum possible length.

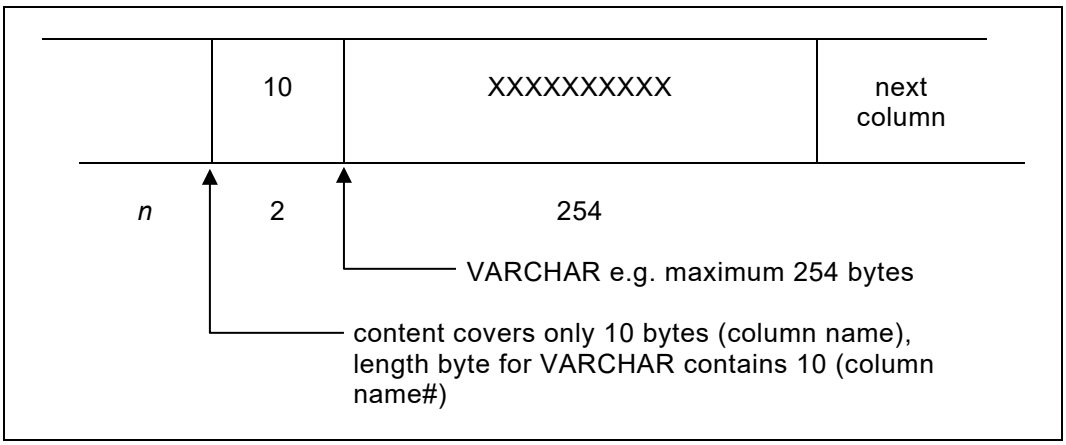


Fig. 233: String length specification for VARCHAR fields

`PUTA` is only allowed in update mode (`UDB=`).

The PUTD Instruction

Individual rows can be **deleted** from a table. The row to be deleted must be read by a `GET` prior to being deleted, i.e. the `PUTD` command deletes the last row read.

`PUTD` is only allowed in update mode (`UDB`).

`PUTD` ist nur im Updatemodus (`UDB=`) möglich.

The ODB= Definition (initial loadInitial Load)

The existing rows in a table are deleted when the table is opened. In this case, the `PUT` command inserts new rows into the table, without the old rows remaining. The individual columns in the row must be previously filled with data, as described under `PUTA`. Attention should be paid to the length bytes for VARCHAR columns, or the indicator field (@), to see if NULL values are to be set.

Under **z/OS**, QPAC, which processes the DB2 tables, is called and executed by the **TSO batch program IKJEFT01**.
The application corresponds to the official format for batch programs.

An example of the job control follows.

```
//         EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//QPACLIST DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DB2)
  RUN PROG(QPAC) PLAN(qpacplan)
  END
//QPACIN   DD *
PARM=XREF,LCT=66
IDB=TBN=table,SCOL=USER/NUMBER,ORDBY=USER(A)
OPF=PR
...
END
```

Fig. 234: DB2 job control (z/OS)

QPAC can also be called without the use of the TSO batch program IKJEFT01.
In this case the plan and the system id have to be defined by `PARM`.

```
//         EXEC PGM=QPAC
//QPACLIST DD SYSOUT=*
//QPACIN   DD *
PARM=PLAN=qpacplan,DB2ID=db2id
IDB=TBN=...
```

Fig. 235: Calling DB2 without IKJEFT01

Enhanced SQL Command Functions

The following examples are related to enhanced capabilities when working with DB2 tables. This new command group is highly flexible and nearly corresponds to the functionality of the IBM SPUFI functions.
Every command of this enhanced group is introduced by the key word **EXECSQL** and terminated by a **semicolon (;)**. The enclosed definition statements are - mostly unchanged - transferred to DB2. Only the relationship to the DB2 table definitions and any potential host variables - where allowed - are resolved in advance.

QPAC internally all these EXECSQL commands are based on **Dynamic SQL**. Dynamic SQL itself implies certain restrictions which cannot be circumvented by QPAC. These restrictions are only minimal and normally not lacked by developer's requirements.

Communication between QPAC and DB2 in connection with EXECSQL commands is handled by new reserved field symbol names. Among other things official identifiers of the SQLCA structure have thereby been implemented such as e.g. SQLCODE, SQLSTATE etc.
The contents of these SQLCA fields are available after every EXECSQL command and afterwards overwritten by any following SQL command.
The additional new reserved field symbols are listed beneath.

After an EXECSQL command any results or column contents are in the individual column fields of the corresponding DB table definition. In case of a so called "expression" result this result is available in a reserved field symbol which is built according to the type and format of the result. Such a field symbol always begins with with the stem **RESULTxn**, followed by the format (**B**=binary, **C**=character, **V**=varcharacter, **P**=packed) and the position as a number where the corresponding **column position** is located within the command string.

e.g.
```
EXECSQL SELECT COUNT(*), MAX (SALARY), MIN(SALARY)
  FROM IDB2 ;
```

results in RESULTB1, because COUNT within the SELECT statement is at position 1 and the result is a binary value, RESULTP2 because MAX (SALARY) is at position 2 and the result is a packed number and RESULTP3 because MIN(SALARY) is at position 3 and also a packed number.

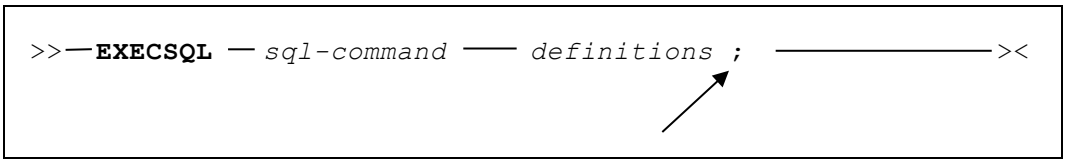
Reserved Field Names (only valid for EXECSQL)

SQLCODE	BL4	SQLCODE
SQLERRD1	BL4	error code
SQLERRD2 - 6	BL4	error code
SQLWARN0	CL1	warning code
SQLWARN1 - 9	CL1	warning code
SQLWARNA	CL1	warning code
SQLSTATE	CL5	status code
RESULTB*n*	BL4	binary expression
RESULTB*n*@	BL4	binary expression indicator
RESULTP*n*	PL16	packed expression
RESULTC*n*	CL256	character expression
RESULTV*n*#	BL2	variable character expression length field
RESULTV*n*	CL254	variable character expression

The EXECSQL Single Instruction

After the key word `EXECSQL` the ultimate SQL command follows which is terminated on its own by a **semicolon (;)**.
Within the SQL command any potential table definitions are not specified with the original table name but with the short identification of a preceding QPAC table definition.
Every table, except from tables in sub selects, that are to be processed have to be preliminary defined as a QPAC DB table definition. By this definition the column names are allocated as fields which then can be filled when required. Thereby any potential prefixes will be considered.

Column names within the command definition are not allowed to contain prefixes because they are directly delivered to DB2. The same rules apply as already described with the `WHERE-Id` instruction.

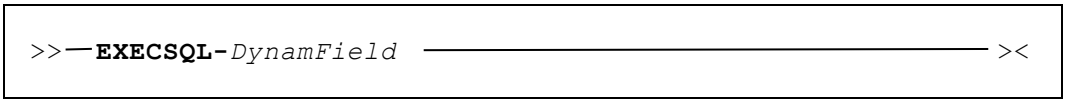


Supported are all SQL commands which can be dynamically prepared. A list of all supported SQL commands can be found in Appendix C of the SQL Reference Manual.

The *definitions* must exactly correspond to the format of the official SQL commands INSERT, UPDATE, DELETE, SELECT etc. except to the before mentioned table definition.
The table definitions are only specified with the QPAC DBId (e.g. `IDB2` or `I2`) which then at the same time connects to the preceding DB table definition and is replaced by QPAC at execution time by the original table name.
Host variables are marked with a preceding **colon (:)** according to SQL convention. The host variable itself must be a defined field symbol at the time of program execution. Otherwise, an error message `108` (undefined symbol) appears.
Details are presented in the following examples.

The `EXECSQL SELECT` command in the above show format corresponds to a so-called s**ingle SELECT**, i.e. only one single row will be made available. If more than one row of the same condition is available the SQLCODE –811 is returned together with the resulting row.

The `EXECSQL` instruction may also be used in the dynamic format. In this case the ultimate SQL command SELECT, INSERT, DELETE or UPDATE are prepared in a predefined area. The field name of this area is the appended to the `EXECSQL` command with a hyphen.



The dynamic format does not permit any host variables with colons **(:)** because they cannot be resolved at execution time. Additionally, the semicolon (;) is not permitted. It is part of the dynamic command itself.

Single Instruction Examples

```
* DB table definition 1
* ---------------------

IDB1=TBN=QPAC.QPACTAB1
 1=COLUMN_CHA,CL10
0S=COLUMN_CHA@,BL2
11=COLUMN_FLOATS,FL4
0S=COLUMN_FLOATS@,BL2
15=COLUMN_FLOATL,FL8
0S=COLUMN_FLOATL@,BL2
23=COLUMN_DATE,CL10
0S=COLUMN_DATE@,BL2
33=COLUMN_DEC,PL5
0S=COLUMN_DEC@,BL2
38=COLUMN_INT,BL4
0S=COLUMN_INT@,BL2
42=COLUMN_SMALLINT,BL2
0S=COLUMN_SMALLINT@,BL2
44=COLUMN_TIME,CL8
0S=COLUMN_TIME@,BL2
52=COLUMN_TIMESTAMP,CL26
0S=COLUMN_TIMESTAMP@,BL2
78=COLUMN_VARCHAR#,BL2
80=COLUMN_VARCHAR,CL254
0S=COLUMN_VARCHAR@,BL2

* DB table definition 2 prefixed
* ------------------------------

IDB2=TBN=DSN8710.EMP,PRFX=YES
 1=I2EMPNO,CL6
 7=I2FIRSTNME#,BL2
 9=I2FIRSTNME,CL12
21=I2MIDINIT,CL1
22=I2LASTNAME#,BL2
24=I2LASTNAME,CL15
39=I2WORKDEPT,CL3
0S=I2WORKDEPT@,BL2
42=I2PHONENO,CL4
0S=I2PHONENO@,BL2
46=I2HIREDATE,CL10
0S=I2HIREDATE@,BL2
56=I2JOB,CL8
0S=I2JOB@,BL2
64=I2EDLEVEL,BL2
0S=I2EDLEVEL@,BL2
66=I2SEX,CL1
0S=I2SEX@,BL2
67=I2BIRTHDATE,CL10
0S=I2BIRTHDATE@,BL2
77=I2SALARY,PL5
0S=I2SALARY@,BL2
82=I2BONUS,PL5
0S=I2BONUS@,BL2
87=I2COMM,PL5
0S=I2COMM@,BL2
```

Fig. 236: EXECSQL - Sample DB table definitions

```
* DB table definition 3 prefixed
* ------------------------------

IDB3=TBN=DSN8710.EMPROJACT,PRFX=YES

1=I3EMPNO,CL6
7=I3PROJNO,CL6
13=I3ACTNO,BL2
15=I3EMPTIME,PL3
0S=I3EMPTIME@,BL2
18=I3EMSTDATE,CL10
0S=I3EMSTDATE@,BL2
28=I3EMENDATE,CL10
0S=I3EMENDATE@,BL2

*
* DB table definition 4
* --------------------

IDB4=TBN=SYSIBM.SYSDUMMY1

1=IBMREQD,CL1
```

Fig. 237: EXECSQL - Sample DB table definitions (cont.)

Static Format:

```
* Example DELETE from DB2 table IDB1
* ----------------------------------

EXECSQL DELETE FROM IDB1 WHERE COLUMN_CHA =
                               'CHARACT400' ;
IF SQLCODE < 0 THEN  .. .. ..  IFEND
IF SQLCODE = 100 THEN    not found    IFEND
IF SQLERRD3 > 1  THEN    more than 1 row    IFEND
```

Fig. 238: EXECSQL - Example DELETE

```
* Example UPDATE of DB2 table IDB1
* --------------------------------

SET X1 = 2048
EXECSQL UPDATE I1
        SET COLUMN_INT   = :X1 ,
        COLUMN_DATE  = NULL
           WHERE COLUMN_CHA = 'CHARACT100' ;

IF SQLCODE < 0 THEN    .. .. ..    IFEND
```

Fig. 239: EXECSQL - Example UPDATE

```
* Example INSERT into DB2 table IDB1
* ----------------------------------

SET COLUMN_CHA      = 'CHARACT402'
SET COLUMN_DATE     = '1990-12-30'
SET COLUMN_DEC,CL5  = X'0000000000'
SET COLUMN_DEC@     = -1
SET COLUMN_INT      = 4096
SET COLUMN_SMALLINT = 2048
SET COLUMN_TIME     = '23.20.00'
SET COLUMN_VARCHAR  = 'VARIABLE CHARAC402'
SET COLUMN_VARCHAR# = 18

EXECSQL INSERT INTO IDB1
    (COLUMN_FLOATS,
    COLUMN_FLOATL,
    COLUMN_CHA,
    COLUMN_DATE,
    COLUMN_DEC,
    COLUMN_INT,
    COLUMN_SMALLINT,
    COLUMN_TIME,
    COLUMN_VARCHAR,
    COLUMN_TIMESTAMP)
    VALUES
       (1234E6,
        1234E6,
        :COLUMN_CHA,
        :COLUMN_DATE,
        NULL,
        :COLUMN_INT,
        :COLUMN_SMALLINT,
        :COLUMN_TIME,
        :COLUMN_VARCHAR,
        CURRENT TIMESTAMP) ;

IF SQLCODE < 0 THEN .. .. .. .. IFEND
```

Fig. 240: EXECSQL - Example INSERT

```
* Example 1 Single SELECT of DB2 table IDB1
* ----------------------------------------

EXECSQL SELECT * FROM I1
   WHERE COLUMN_CHA = 'CHARACT402' ;

IF SQLCODE NOT = 0 THEN   any error   IFEND
```

Fig. 241: EXECSQL - Example 1 Single SELECT static format

The first row that complies to the WHERE clause is filled into the column fields of table `IDB1`.
The **return code –811** signalizes that there are additional rows that satisfy this condition.

```
* Example 2 Single SELECT of DB2 table IDB1
* ----------------------------------------

EXECSQL SELECT COUNT(*) FROM IDB1 ;

IF SQLCODE NOT = 0 THEN    any error    IFEND
IF  RESULTB1   > 0 THEN
     ....
     ....
IFEND
```

Fig. 242: EXECSQL - Example 2 Single SELECT static format

The number of rows in table IDB2 are filled into the reserved field **RESULTB1** which has the format 4 bytes **binary** and **position number 1**, because after the SELECT command the function COUNT is at position 1.

```
* Example 3 Single SELECT of DB2 table IDB1
* ----------------------------------------

EXECSQL SELECT SUM( COLUMN_DEC ),
               MAX(COLUMN_DEC) ,
               MIN( COLUMN_DEC),
               FROM IDB1 ;

IF SQLCODE NOT = 0 THEN   any error  IFEND
IF  RESULTP1@  < 0 THEN     null     IFEND
IF  RESULTP1   > 0 THEN  .. .. .. .. IFEND
IF  RESULTP2@  < 0 THEN     null     IFEND
IF  RESULTP2   > 0 THEN  .. .. .. .. IFEND
IF  RESULTP3@  < 0 THEN     null     IFEND
IF  RESULTP3   > 0 THEN  .. .. .. .. IFEND
```

Fig. 243: EXECSQL - Example 3 Single SELECT static format

The results of these functions are 16 bytes packed. SUM is at position 1, MAX at position 2 and MIN at position 3. The field with the trailed alpha sign (@) signalizes a potential NULL value.

```
* Example 4 Single SELECT of DB2 table IDB2
* ----------------------------------------

0S=KEY,CL6
SET KEY = '000030'

EXECSQL SELECT LASTNAME FROM IDB2
           WHERE EMPNO = :KEY  ;

IF SQLCODE = 100   THEN    not found    IFEND
IF SQLCODE NOT = 0 THEN    any error    IFEND
```

Fig. 244: EXECSQL - Example 4 Single SELECT static format

Field I2LASTNAME of table definition IDB2 is filled.

```
* Example 5 Single SELECT of DB2 tables IDB2 and IDB3
* -------------------------------------------------

EXECSQL SELECT AA.EMPNO, SUM(EMPTIME)
            FROM IDB2  AA ,
                 IDB3  BB
            WHERE AA.EMPNO = '000040' AND
                  BB.EMPNO = '000040'
            GROUP BY AA.EMPNO ;

IF SQLCODE NOT = 0 THEN   any error   IFEND
```

Fig. 245: EXECSQL - Example 5 Single SELECT static format

In the example above correlation names are used. Such names cannot be DB Ids (e.g. `IDB1`) !
Correlation names must be defined with all columns names if the corresponding DB Id definition contains a correlation name.
The column names in the `SELECT` definition cannot contain the QPAC prefixes. In the `SELECT` only potential DB Ids and host variables with colon (:) are resolved before the string is handed over unchanged to DB2-SQL.

```
* Example 6 Single SELECT of DB2 table IDB3
* ----------------------------------------

EXECSQL SELECT CURRENT TIME, CURRENT DATE,
               CURRENT TIMESTAMP
           FROM IDB3  ;

IF SQLCODE NOT = 0 THEN   any error   IFEND

SET SAVETIME      = RESULTC1
SET SAVEDATE      = RESULTC2
SET SAVETIMESTAMP = RESULTC3
```

Fig. 246: EXECSQL - Example 6 Single SELECT static format

Results are in the reserved fields with format C (CHARACTER) and the position numbers 1-3. These character fields are 256 bytes long. See under "reserved field symbols" in this regard.

Dynamic Format:

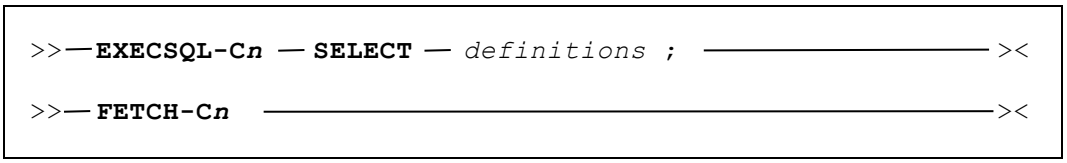
```
* Example DELETE from DB2 table IDB1
* ----------------------------------

1W=DYNAMAREA,CL800

SET DYNAMAREA = 'DELETE FROM I1     '/
'WHERE COLUMN_CHA = '/
''CHARACT400'' ; '

EXECSQL-DYNAMAREA

IF SQLCODE < 0 THEN .. .. .. IFEND
```

Fig. 247: EXECSQL - Example Dynamic format

The EXECSQL Cursor Instruction for SELECT Commands

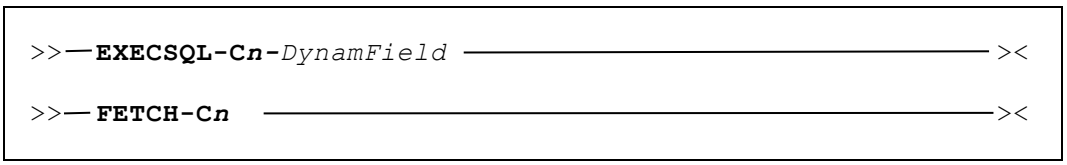
Beside the EXECSQL instruction for single SQL commands there is also the version where the EXECSQL command is associated with a cursor and thereby has the possibility to process multiple rows sequentially.

For this the additional command FETCH is used that fills one row after the other into the column fields. Regarding the SELECT definitions the same rules apply as previously described for the EXECSQL single instruction.



To the EXECSQL command an identification (C*n*) is appended with a hyphen. This can be C1 to C19. It is a reference id comparable with a file identification which on the one hand gives a hint that it is a single SELECT and on the other hand also must be specified with any following corresponding FETCH-C*n* command. It is also the cursor name which is internally assigned.
Please note: The cursors C1-C9 are internally declared **without "WITH HOLD"** and the cursors C10-C19 are internally declared **with "WITH HOLD"**.

The EXECSQL instruction with cursor may also be used in a dynamic format. In this case the area's field name is also attached to the Id with a hyphen. The complete SELECT command is then placed into this area before calling the EXECSQL, also having the previously mentioned restrictions for host variables with colons (:).



Sample SELECT Cursor Instruction

Static Format:

```
* Example 1 SELECT Cursor from DB2 table IDB2
* ----------------------------------------

EXECSQL-C1 SELECT * FROM IDB2 ;
IF SQLCODE NOT = 0 THEN  prep or open error  IFEND

DO-FOREVER
   FETCH-C1
   IF SQLCODE = 100   THEN DOQUIT       IFEND
   IF SQLCODE NOT = 0 THEN   any error  IFEND
   ...
DOEND
```

Fig. 248: EXECSQL - Example 1 Static format with Cursor

```
* Example 2 SELECT Cursor from DB2 tables and Subselect
* ----------------------------------------------------
*
EXECSQL-C8 SELECT AA.EMPNO, AA.LASTNAME,
                  'No project activities'
             FROM IDB2  AA
             WHERE NOT EXISTS
                (SELECT EMPNO FROM IDB3
                    WHERE AA.EMPNO = EMPNO)
           UNION
           SELECT AA.EMPNO, AA.LASTNAME, BB.PROJNO
             FROM IDB2  AA , IDB3  BB
             WHERE AA.EMPNO = BB.EMPNO ;

IF SQLCODE NOT = 0 THEN  any error  IFEND
...
FETCH-C8
```

Fig. 249: EXECSQL - Example 2 Static format with Cursor

In this example a sub select is shown. At the third position a text constant is defined. This text resp. `BB.PROJNO` stands in the result field `RESULTV3` after execution. This field has the attribute `V` (VARCHAR) and has a length of 254 bytes. The effective length is in the appropriate field `RESULTV3#`.

Dynamic Format with cursor:

```
* Example SELECT from DB2 table IDB1
* ----------------------------------

0S=ANYFIELD,CL6
0S=ANYVALUE,M'999.99'
1W=DYNAMAREA,CL800

SET ANYFIELD = '000030'
SET ANYVALUE = 55000

SET DYNAMAREA = 'SELECT * FROM IDB1  '/
'WHERE EMPNO <       '/
' ''' ! ANYFIELD ! '''/
' AND BONUS > ' !  ANYVALUE

EXECSQL-C5-DYNAMAREA

IF SQLCODE < 0 THEN .. .. .. IFEND

DO-FOREVER
   FETCH-C5
   IF SQLCODE = 100   THEN DOQUIT    IFEND
   IF SQLCODE NOT = 0 THEN any error IFEND
   ...
DOEND
```

Fig. 250: EXECSQL - Example Dynamic format with Cursor

The definition in the `DYNAMAREA` must not contain any host variables with colon (:). But using the `SET` instruction character fields may be concatenated whose contents is then taken at execution time.
The contents of `DYNAMAREA` is handed over unchanged to DB2-SQL after the DB Id `IDB1` has been resolved.

Additional Sample SQL Commands

```
EXECSQL  GRANT BIND  ON PLAN TESTQPAC TO PUBLIC;

EXECSQL  SET CURRENT SQLID = 'XYZ';

EXECSQL  COMMIT;

EXECSQL  ROLLBACK;

EXECSQL  DROP   TABLE  QPAC.QPACTAB3;

EXECSQL  CREATE TABLE  QPAC.QPACTAB3
        (NRSPR00       SMALLINT,
         NRVSO00       CHAR(6),
         NRKNS00       CHAR(5),
         TSMUT00       TIMESTAMP,
         AZVRNHI       DECIMAL (5,0)
        )  IN DSN8D71A.DSN8S71D;
```

Fig. 251: EXECSQL - Examples additional SQL commands

The same definitions are also possible in dynamic format:

```
0S=DYNAMICAREA,CL80

SET DYNAMICAREA = 'GRANT BIND ON PLAN TESTQPAC TO PUBLIC'
EXECSQL-DYNAMICAREA
```

Fig. 252: EXECSQL - Examples SQL commands in dynamic format

Auto Commit

The reserved field symbol `DB2COMMIT` is a counter field of 8 bytes. It can be used to automatically trigger a COMMIT when a specific number of INSERTs and UPDATEs through `EXECSQL` and `PUTA-U`*`n`* or `PUTD-U`*`n`* , `PUT-O`*`n`* basic commands has been reached.
This number can be specified in the `DB2COMMIT` field. When this value has been reached and a COMMIT has been done the counting begins again.

```
SET DB2COMMIT = 10

SET DB2COMMIT = 0
```

Fig. 253: DB2 auto commit

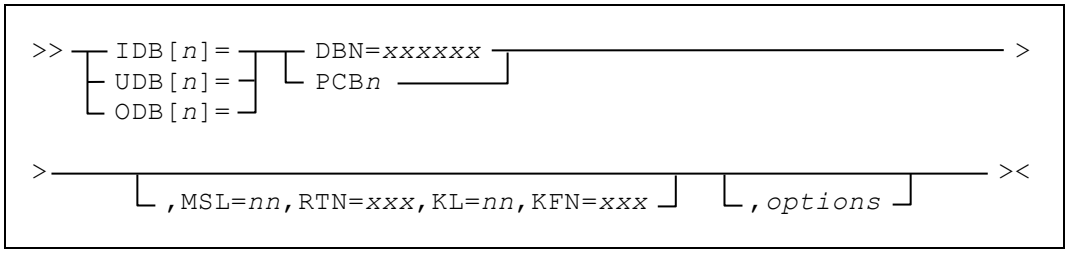
If the `DB2COMMIT` field is set to 0 (zero) the auto commit function is deactivated. Default is 0.

Chapter 13. DL/I Support Feature

DL/I Data Base Definition

IBM DL/I is fully supported by this feature. It assumes, that DL/I is installed, and that certain DL/I basic routines are available in the system.

Grundformat der DL/I DB Definition



`DBN=` `PCBn`	data base name. It is assumed that the PCB*n* accompanying PSB contains a DB definition under this name. Instead of the data base name a PCB number within the PSB may be defined.
`MSL=`	maximum segment length. If this definition is missing, the segment length for the I/O buffer will be taken from the internal DL/I blocks, **where possible**. This definition is required when the internal DL/I blocks are not available. This is the case when DL/I is located in another address area (QPACBMP, DBCTL under z/OS).
`RTN=`	root segment name
`KFN=`	name of the key field in the root segment
`KL=`	key length of the root segment This parameters are always required when DL/I data base management is located in another address area to that of QPAC program itself, and if the command `SETGK/SETEK` is to be used at the same time. QPACBMP or DBCTL always require this definition. In other cases, this information for QPACDLI is automatically taken from the internal blocks.
`options`	additional options according to the following descriptions:
`CLR=` `CLE=`	additional options as described in Chapter 2: Input/Output Definitions `CLR=` refers to an output area `CLE=` refers to an input area The rules for clearing areas as specified for other file organizations, also apply to DL/I. If no option is specified, the output area is cleared to X'00' and the input area to X'FF'.

FCA=

File communication area
This definition determines the address within the general QPAC working storage area which will be used as the communication area for the files. The default form of FCA definition is dynamic.

SSEG=

selected segments
If not all segments within the DBD should be made available, those required can be defined by this option, segment names being separated by a '/' .

```
SSEG=NAMEA/NAMEB/NAMEX
```

Fig. 254: Specification of multiple segment names

To continue onto a following QPAC statement, when defining several segment names, the first statement must end with '/' and the next one start with the segment name.

```
SSEG=NAMEA/NAMEB/
  NAMEX
```

Fig. 255: Segment names specification on the following statement

SSEG= is only allowed for input or update (IDB=, UDB=).

WP=*nnnnn*

Work area position
The segment should be written into, or read from, the defined general work area. The parameter defines the position within the work area reserved for the segment. When WP= is defined, there is no dynamic record area for the corresponding file definition.

NOGE

Suppresses the return code GE, which can occur when using SETGK or SETEK. (Refer to "Hints on Processing Logic").
It should also be noted that, in such a situation, instead of the return code, the next segment (root segment) will be passed over. At the same time the internal switch changes from 'record orientated' to 'DB orientated'.

NOII

Suppresses the return code II which can occur with a PUTA instruction when a segment with the same key already exists. IF NOII is specified and a segment cannot be inserted, processing halts. If NOII is not specified and the segment already exists, the return code II is present in the FCA following the PUTA.

RC=YES

return code
The DL/I return code is returned in the FCA field ..RC without QPAC being terminated in case of an error. This return code contains blank if no DL/I return code has appeared.

STAT

data base statistics
At end of processing (close time) data base statistics should be displayed.

Processing by QPAC with the GET instruction is sequential and is based on logical DBs as described by PSB definitions. Segment after segment is made available in its turn.

All common DB organizations are supported. QPAC does not contain components that contradict official DB and PSB definitions. QPAC organizes itself specifically to fit in with the defined PSB.

The FCA is used for the exchange of information between DL/I and QPAC, mainly for return codes. These have a logical meaning not necessarily connected to an error situation. The FCA is also used for key values when using the SETGK or SETEK functions.

The FCA for DL/I is structured as follows:

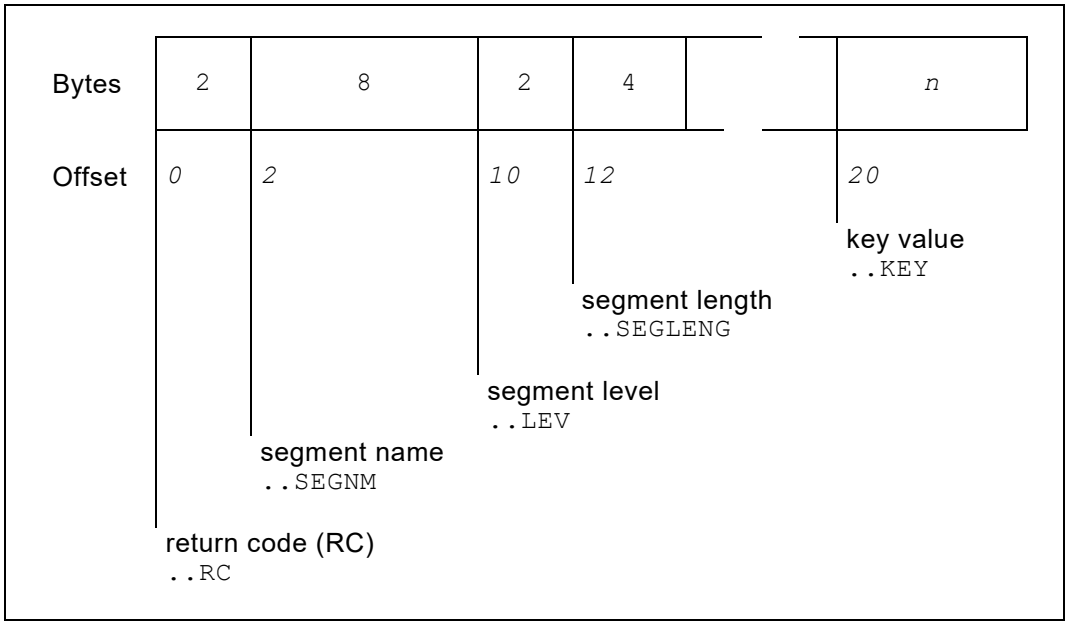


Fig. 256: FCA for DL/I

return code (RC)

return code (RC)
The follwoing return codes are returned by QPAC, if RC=YES was **not** specified:

for input/update:
~~bb~~ = normal situation without any conditions
GA = crossed hierarchical boundary
GE = segment not found
GK = different segment type at same level
II = segment can not be inserted by PUTA

for output:
~~bb~~ = normal situation without any conditions

Return codes which do not begin with G or a blank, always result in a processing halt: the exception is II.

The return code GB (end of a data base) results in an EOF situation and is only returned if RC=YES has been specified.

segment name	After each GET the segment name is available in the FCA. For output, the name of the segment to be written must be placed in the FCA before the PUT instruction.
segment level	segment level 2 bytes numeric EBCDIC.
segment length	in binary form, 1 fullword, after a GET command. It is only made available if the internal DL/I blocks are accessible, i.e. if the DL/I management is located in the same address area.
key value	used to start set generic key (SETGK) or set equal key (SETEK). See SETGK/SETEK description.
filler (IDB & UDB)	The unused right-hand part of the segment area available, (for a maximum segment length of 4096 bytes), is filled with X'FF'. If a different filler is required, it can be explicitly defined using the CLE= option in the file definition.

```
IDB=DBN=EDB888,CLE=X'40'
```

Fig. 257: Option for special filler

The DL/I support feature uses the official processing routines of the DL/I system, which must be present under their official names in the DL/I or system libraries.

The PARM parameter of the EXEC statement cannot be used when working with DL/I z/OS. The QPAC DL/I support feature therefore has a QPAC PARM statement:

This PARM statement has a fixed format, i.e. it must contain PARM= in **positions 1-5** and must be the **first statement** to be read in:

```
//       EXEC   PGM=   (QPACDLI     or
//DD1    DD             QPACBMP)
//DD2    DD
...
//QPACIN DD *
PARM=LIST,NOLOG,LINECNT=66
IDB=DBN=DBTEST,MSL=150
OPF=PR
...
END
```

Fig. 258: DL/I example for z/OS

The PARM options are equivalent to those described in Chapter 1: Introduction under PARM option.

The call to QPAC from the job control corresponds to DL/I conventions. There is therefore no difference between a PSB generated for PL/I and one for COBOL / ASSEMBLER.
In each case, for z/OS systems, one of the following interfaces must be called as the main program:

QPACDLI	for pure batch processing
QPACBMP	for DL/I online data bases in a separate address space (e.g. DBCTL)

(PSBTEST is generated for COBOL programs)

```
//EXEC PGM=DFSRRC00,PARM='DLI,QPACDLI,PSBTEST,15'
//EXEC PGM=DFSRRC00,PARM='DLI,QPACBMP,PSBTEST,..'
```

Fig. 259: DL/I call

A defined PSB can contain several PCB's. The required DB is selected according to the `DBN=` parameter in the DB file definition, and only this one is accessed by QPAC.

Instead of the `DBN` (data base name) definition, which provides the search basis for a PCB within a PSB, the PCB can also be defined by its place number. Thereby, `PCB1` is the first, `PCB4` the fourth PCB in the PSB.

Simple HISAM DBs can only be used with `SETGK`, if the `NOGE` option is defined. This definition excludes that of a parentage.

The segment length can only be returned in the FCA when QPACDLI is used, i.e. with pure batch processing. This information is not available when online data bases are accessed.

Hints on Processing Logic

Basically, a DB can be read (IDB), modified (UDB) or created (ODB).

Additions to, or **deletions** from an existing DB are supported in update mode (UDB).

Processing is by segments in their hierarchical order, top to bottom, left to right. Only sensitive segments are made available. The user must himself identify which segment is available at any given time. A segment-type change on the same level, or the crossing of a hierarchical boundary, is indicated by the return code. The appropriate segment name is available in the FCA.

If only GETs are used, processing is from beginning to end of the DB. This is equivalent to the GN (go next) function. If UDB is defined instead of IDB, the QPAC GET is equivalent to the function GHN, and a following PUT equivalent to REPL. A PUTA command corresponds to ISRT, and PUTD to the DLET.

The SETGK Instruction

The SETGK operation provides a further possibility. **SET GENERIC KEY** allows processing to start at a given **ROOT SEGMENT**.

SETGK itself does not make the root segment available, this is done by the following GET command.

Before issuing a SETGK command, the key value must be stored in the FCA. The key length is automatically given to QPAC through the DBD. The following GET corresponds to the function GHU and makes available the root segment with equal or next higher key value. If no such segment is found before end of DB, a return code of GE (no segment found) is given.

GETs following the first GET correspond to the function GNP (GHNP) until the end of the DB record is reached, when the return code GE (no segment found) is given.

```
SET WPOS9020,CL8 = 'KEY'
SETGK-I1  GET-I1
DO-UNTIL I1RC = 'GE'
...
...
GET-I1
```

Fig. 260: Sample SETGK for DL/I access

It is important to note, that by using a SETGK operation, processing is based **DB record mode**, i.e. on reaching DB record end, the GE is returned and no segment is made available. An update at this time, through a PUT (REPL) would be erroneous (return code DJ). If SETGK operations are not used, processing is in **DB mode**.

DB record mode: SETGK sets processing to a given root segment. The following GET commands correspond to the DL/I function GNP (get next within parent). The end of the DB record (before the following root segment) is indicated by the return code GE. A following GET command makes the following root segment available and automatically switches to DB mode processing, or possibly to EOF status (GB if RC=YES is defined) if the preceding DB record was the last one in the DB.

DB mode: With each GET command, the next sensitive segment is made available, until DB end. The GET command corresponds to the DL/I function GN (get next). The change from the end of a DB record to the next root segment is indicated by a return code of GA, as it is in the case of every boundary change to a higher level within dependent segments.

e.g.	QPAC operation	DL/I function	FCA return code	availability
a)	SETGK			(no segment)
	GET	GU		root segment
	GET	GNP		dependent segment
	GET	GNP		dependent segment
	GET	GNP	GE	(no segment)
	SETGK			(no segment)
	GET	GU		root segment
	.			
	.			
b)	SETGK			(no segment)
	GET	GU		root segment
	GET	GNP		dependent segment
	GET	GNP		dependent segment
	GET	GNP	GE	(no segment)
	GET	GN		root segment
	GET	GN		dependent segment
	GET	GN		dependent segment
	GN	GN	GA	root segment
	GET	GN		dependent segment
c)	SETGK			(no segment)
	GET	GHU		root segment
	PUT	REPL		
	GET	GHNP		dependent segment
	PUT	REPL		
	GET	GHNP		dependent segment
	GET	GHNP	GE	(no segment)
	PUT	REPL	DJ error	
d)	.			
	.			
	GET	GN		segment
	GET	GN		segment
	.			
	.			
	GET	GN	GB	end of data base if RC=YES has been defined

The SETEK Instruction

The SETEK operation provides a further possibility.

Set Equal Key allows processing to start at a given **ROOT SEGMENT** with the same key.

SETEK itself does not make the root segment available, this is done by the following GET command.

Before issuing a SETEK command, the key value must be stored in the FCA. The key length is automatically given to QPAC through the DBD. The following GET corresponds to the function GU (GHU) and makes available the root segment with the same key value. If no such segment is found, a return code of GE (no segment found) is given.

GETs following the first GET correspond to the function GNP (GHNP), until the end of the DB record is reached, when a return code of GE (no segment found) is given.

```
SET WPOS9020,CL8 = 'KEY'
SETEK-I1  GET-I1
DO-UNTIL WPOS9000,CL2 = 'GE'
...
...
GET-I1
```

Fig. 261: Sample SETEK for DL/I access

It should be noted that when using the SETEK operation, processing is in **DB record mode**, i.e. when an end of DB record is reached, a return code of GE is set, and no segment is made available. Thereby, in update mode, a following PUT (REPL) command would be in error, (return code DJ).

SETEK is otherwise no different to SETGK, and thereby what applies to DB mode processing under SETGK applies here too.

The PUTA Instruction

Segments can be **inserted** into a DB by the PUTA command (put add). The segment name must be placed in the FCA and the segment itself in the record area.

The return codes ~~bb~~ and II are both possible; the latter only if NOII is not defined as an option.

PUTA is only possible in update mode (UDB=), and only if the PSB allows 'inserts'.

The PUTD Instruction

Segments can be **deleted** from the DB by the `PUTD` command (put delete). The segment to be deleted must first be read by a `GET` command, i.e. `PUTD` deletes the segment last read.

`PUTD` is only possible in update mode (`UDB=`), and only if the PSB allows 'deletes'.

The ODB= Definition (Initial Load)

A database can also be created. Therefore a PSB with PROCOPT LS or L must be used. Before execution the segment name must be inserted into the FCA field `..SEGNM` (segment name). The segment record is contained in the I/O area.

DL/I Database Related Commands with SSAs.

A separate category of DL/I commands is available. But these commands use SSA fields, and their contents has to be defined additionally.
Up to 8 SSA fields (each has a max. length of 256 bytes) are available. The names are `..SSA1` to `..SSA8`. They must be used in number sequence. The number of SSA fields in use is specified in the `..SSAN` field (SSA numbers).

The contents of the SSA fields must adhere to the official DL/I format. QPAC expects the format to be correct.

This command category is valid for `IDB` and `UDB`, **NOT** for `ODB`.

`GU`	`-I[`*n*`]`	`-U[`*n*`]`	Get unique
`GHU`	`-I[`*n*`]`	`-U[`*n*`]`	Get hold unique
`GN`	`-I[`*n*`]`	`-U[`*n*`]`	Get next
`GHN`	`-I[`*n*`]`	`-U[`*n*`]`	Get hold next
`GNP`	`-I[`*n*`]`	`-U[`*n*`]`	Get next within parent
`GHNP`	`-I[`*n*`]`	`-U[`*n*`]`	Get hold next within parent
`REPL`	`-I[`*n*`]`	`-U[`*n*`]`	Replace
`DLET`	`-I[`*n*`]`	`-U[`*n*`]`	Delete
`ISRT`	`-I[`*n*`]`	`-U[`*n*`]`	Insert

Fig. 262: DL/I commands with SSAs

```
SET I1SSA1 = 'ROOTSEG *-(ROOTKEY = 100)'
SET I1SSAN = 1
GU-I1
IF I1RC = X'4040' THEN all ok IFEND
IF I1RC = 'GE' THEN not found IFEND
```

Fig. 263: Example of usage

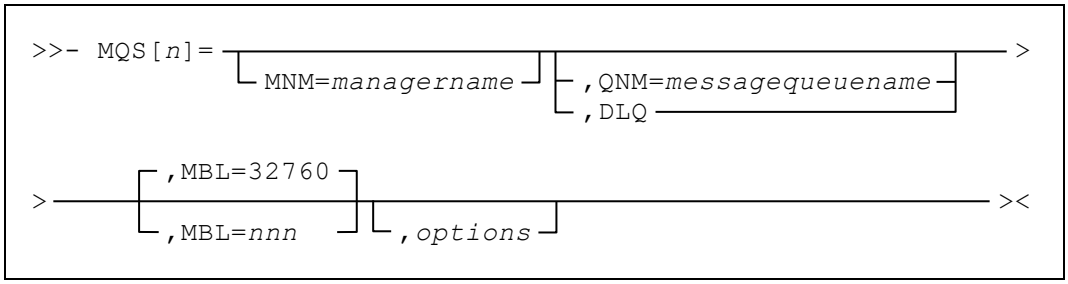
The official DL/I return code is available in the FCA field `..RC` (return code) after execution.
With this command category the option `RC=YES` is the default and the execution is therefore not terminated in case of any error.

Chapter 14. MQSeries Support Feature

MQSeries Single Message Queue Definition

IBM MQSeries for z/OS is fully supported by this feature. It is based on the assumption that MQSeries is installed, and the necessary authorization is given.

Basic Format of the MQSeries Message Queue Definition



`MNM=` Is the active queue manager name of the MQSeries system. The manager name can also be dynamically filled into the reserved field **`Qn`MGRNAME** before the `CONNECT` command is executed.

`QNM=` Is the message queue name of the queue that will be sent or read. This queue name can also be dynamically filled into the reserved field **`Qn`QNAME** before the `OPEN` command is executed.

`DLQ` Dead Letter Queue
This definition, instead of providing a message queue name (`QNM=`) specifies that this file definition has to be provided for reading the Dead Letter Queue.
QPAC then automatically determines and dynamically assigns the name of the Dead Letter Queue after the CONNECT process. With the `GET` commands the Dead Letter Header (DLH) is put by QPAC into the separate Dead Letter Area (`QnDLH_AREA`) and is then available to the user. The message is available in the buffer area.

`MBL=` With this key word the maximum buffer length is defined which is provided for the message queue. The specification can be done in **bytes or in megabytes**.
Megabytes are specified in sizes of **1M 2M 3M 4M**. 100M is the largest value allowed within z/OS.
If this definition is missing a default buffer length of 32760 bytes is assumed. This value cannot be dynamically assigned or increased but if necessary, it can be decreased by using the key word **`Qn`BUFFLENG**.

`options` The following options may be used additionally:

RC=YES

The completion code and the reason code are returned in the FCA and there will be no abend in case of an error. The assigned reserved field symbols are named **Q*n*COMPCODE** and **Q*n*REASON**.
In case of an error the current command is filled into the FCA field Q*n*CMDTEXT and the reason code clear text into the FCA field Q*n*REASONTEXT.

FCA=

File Communication Area
This definition specifies the address within the QPAC working storage of the file communication area for the corresponding MQSeries file definition. Normally the FCA is not explicitly bound to an address of the working storage. If not explicitly specified it is dynamically allocated and the assigned field symbols are used.
The structure of the FCA is shown graphically on one of the following pages.

CLR=X'00'
CLR=NO

This option specifies the clearing value with which the message queue area should be cleared after a PUT command. If this definition is missing, X'00' is assumed, i.e. the queue area will be cleared with Low Value. By specifying CLR=NO the clearing can be prevented.

WP=*nnnnn*

Working Storage Position
This definition specifies that the MQSeries message queue area is allocated within common working storage. The working storage area must have an adequate size which is specified using the PARM parameter **PARM=WORK=*nnnnn***. If WP= is specified an additional dynamically allocated queue area will not exist.

Processing of MQSeries Message Queues

Message queues are processed by QPAC using the commands **CONNECT**, **OPEN**, **PUT** resp. **PUT1**, **GET**, **COMMIT**, **CLOSE**, **DISCONNECT** and in special cases **INQUIRE** and **SET**.

Within a QPAC program multiple message queues may be defined simultaneously. The sum of all maximum addressable buffer lengths must not exceed 16 MB, i.e. for example 4 message queues each with a maximum buffer length of 4 MB (MBL=4M) could be defined.

Together with the message queue definitions all other file and database organizations such as VSAM, SAM, DB2 etc. may be defined in any combination. The declaration is always done following the same rule: The file ident key word is appended by a number from 1-99.

```
IPF1=...   IDB2=...   MQS3=...   OPF4=...
```

Fig. 264: Explicit file definitions

This number is part of the reference when used in input/output commands or implicit position symbols.
For example the position symbol Q3POS1 specifies the first position of the message queue area of the file declaration MQS3=. The same applies to the command CONNECT-Q3.

Before any message can be processed, read or sent, the connection to the message queue manager must be established using the **CONNECT** command. The CONNECT command must be executed for every single MQS*n* definition. QPAC internally verifies whether a connection to the same queue manager already exists and internally establishes its connection. The same applies to the **DISCONNECT** command.

Access to a message must be established using an OPEN command. According to the desired processing the "open option" must be set. The open option is stored into the predefined field **Q*n*OPENOPT**. Therefore, QPAC provides for the official symbols (as shown in the CMQA Macro/Book) which can be used as sending fields instead of the direct numbers.
For example the option OUTPUT can be set as follows to send messages:

```
SET Q3OPENOPT = MQOO_OUTPUT
```

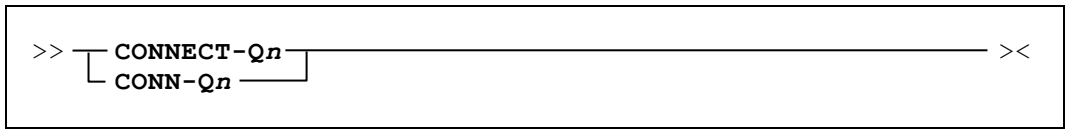
Fig. 265: Output option

If **RC=YES** is defined with a file definition the completion code must be examined. Here also the official symbols may be used according to CMQA macro.

```
IF Q3COMPCODE = MQCC_OK     THEN ...
IF Q3COMPCODE = MQCC_FAILED THEN ...
```

Fig. 266: Examples of the completion code examination

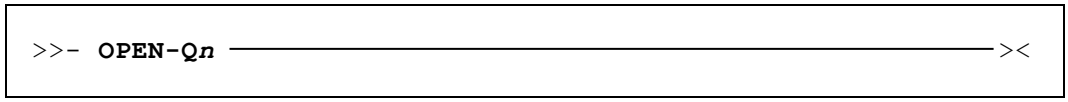
MQSeries Commands



Before the CONNECT command, a queue manager name may be inserted into the field **Q*n*MGRNAME** if not already defined in the file definition.

```
SET Q1MGRNAME = 'MANAGER'
CONNECT-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Fig. 267: Example MQSeries CONNECT command



Before the OPEN command the open option must be inserted into the field **Q*n*OPENOPT**. See the open option list under **'Values Related to MQOPEN** Open Options'. Additionally, a message queue name may be specified with the field **Q*n*QNAME** if not already defined in the file definition.

```
SET Q1OPENOPT = MQOO_INPUT_SHARED
OPEN-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error

or

SET Q1QNAME   = 'MESSAGE_QUEUE_NAME'
SET Q1OPENOPT = MQOO_OUTPUT
OPEN-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Fig. 268: Examples MQSeries OPEN command

```
>>- GET-Qn ------------------------------------------------------------><
```

Before the GET command the options listed in the "Get Message Options" may be inserted into the field Q*n*GMO_OPTIONS if necessary. Additional fields with can be looked up in the MQGET OPTIONS area.
See therefore the equates under **'Values Related to MQGMO'**.

After a GET command the message is put into the input/output area like the record of any conventional data set. The FCA field Q*n*DATALENG contains the current message length. This can be smaller than the buffer length, i.e. the maximum input/output area. If the FCA field completion code Q*n*COMPCODE does not contain 0 an error has occurred, and its reason is available in the FCA field reason code Q*n*REASON. Additionally a short clear text to the reason code is available in the FCA field Q*n*REASONTEXT.

A GET command is only allowed if the processing has been opened with an input open option, e.g. MQOO_BROWSE or MQOO_INPUT_AS_Q_DEF.

```
GET-Qn AT-EMPTY ... if message queue is empty ... ATEND
```

Fig. 269: Example MQSeries GET command

AT-EMPTY has the same meaning as AT-EOF with the only difference that no automatic CLOSE is done. AT-EMPTY is only available with MQS*n*= file definitions and may be specified if RC=YES has not been defined.

```
>>- PUT-Qn ------------------------------------------------------------><
```

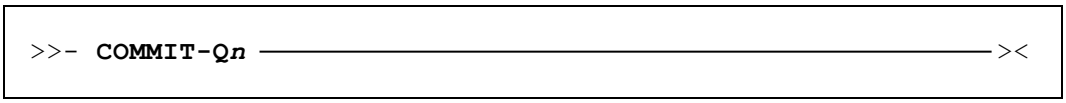
Before the PUT command the message must be stored into the output area.
The current length should be stored into the FCA field Q*n*DATALENG. If missing the maximum buffer length is assumed (MBL=).
Have a look at the PUT options if necessary and consult the equates under **'Values Related to MQPMO'**.

The PUT command is only allowed if processing has been opened with the open option MQOO_OUTPUT.

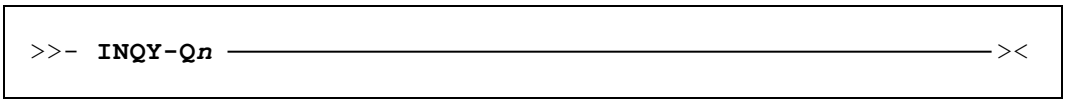
```
SET Q1POS1,CL100 = 'MESSAGE DATA'
SET Q1DATALENG   = 100
PUT-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Fig. 270: Example MQSeries PUT command

The completion code Q*n*COMPCODE has to be examined if RC=YES has been specified. If RC=YES is missing QPAC will terminate execution in case of any error.



If desired the COMMIT command may be specified after a PUT command. It's meaning is self explaining.



The INQUIRE command allows the collection of additional information.

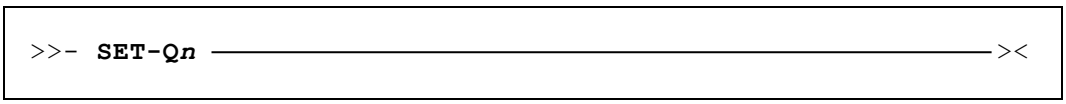
Before the command the SELECTOR values must be set. A maximum of 16 selector fields are available. They are predefined and are called **Q*n*SELECTOR1,BL4** to **Q*n*SLECTOR16,BL4**. The symbol **Q*n*SLECTORS,CL64** spans over all individual fields. There is an additional field **Q*n*SELCNT,BL4** into which the number of used selector fields must be filled. There are two groups, character attribute selectors and integer attribute selectors.

The result of the character attribute selectors is available in the predefined field **Q*n*CHARATTAREA,CL256** and the result of the integer attribute selectors in the field **Q*n*INTATTARRAY,CL64**. Q*n*INTATTARRAY are 16 fields in sequence of the format BL4. Each of those 16 integer attribute fields has its own name too: **Q*n*INTATT1** to **Q*n*INTATT16**.

Both areas also have a length and a counter field which are predefined. Their names are **Q*n*CHARATTLENG,BL4** and **Q*n*INTATTCNT,BL4**. In these fields the length of the used character attribute area resp. the number of used integer attribute fields must be set.

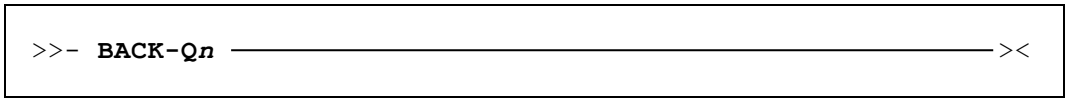
```
SET Q1SELCNT      = 1
SET Q1SELECTOR1   = MQCA_CREATION_DATE
SET Q1INTATTCNT   = 0
SET Q1CHARATTLENG = 12
INQY-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Fig. 271: Example MQSeries INQY command

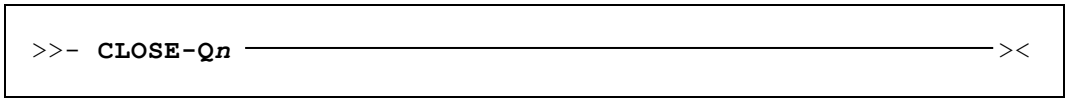


With the SET command new attributes can be set.
The SET command is principally the opposite of the INQY (INQUIRE) command.

Before the SET command - as opposed to the INQuire command - the selector fields, the character values to be stored and/or the integer attributes have to be set.
The same fields are to be used as documented under the INQY command.



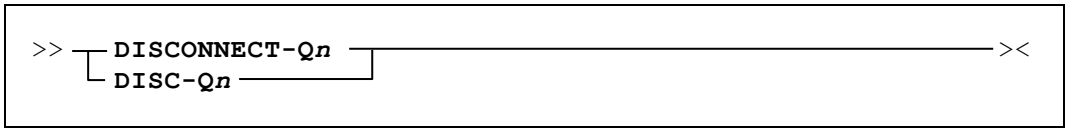
With this command a rollback can be done. All GET-Q*n* and PUT-Q*n* operations are backed out. This command is only effective BEFORE a CLOSE-Q*n*.



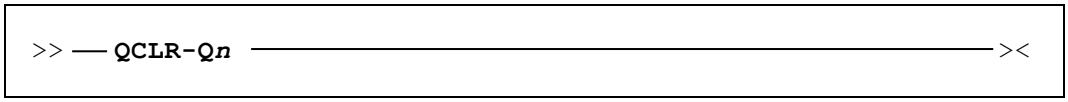
Before the CLOSE command the close option has to be set in the field Q*n*CLOSEOPT. See therefore the close option list under '**Values Related to MQCLOSE** close options'.

```
SET Q1CLOSEOPT = MQCO_NONE
CLOSE-Q1
```

Fig. 272: Example MQSeries CLOSE command



Before the DISCONNECT command normally no additional options must be defined. The DISCONNECT command definitely terminates the connection to the queue manager.



With this command the contents of the referenced queue can be deleted. The command internally works with the MQSeries utility CSQUTIL. The JCL statements //SYSIN DD .. and //SYSPRINT DD .. will be internally allocated and should therefore not be defined within the job step.

In case of an error the FCA field **Q*n*REASON** contains the reason code, the FCA field **Q*n*CMDTEXT** contains the command and the FCA field **Q*n*REASONTEXT** contains the corresponding reason code in clear text according to CMQA macro, for example MQRC_NOT_OPEN_FOR_OUTPUT.

The FCA for MQSeries has the following structure:

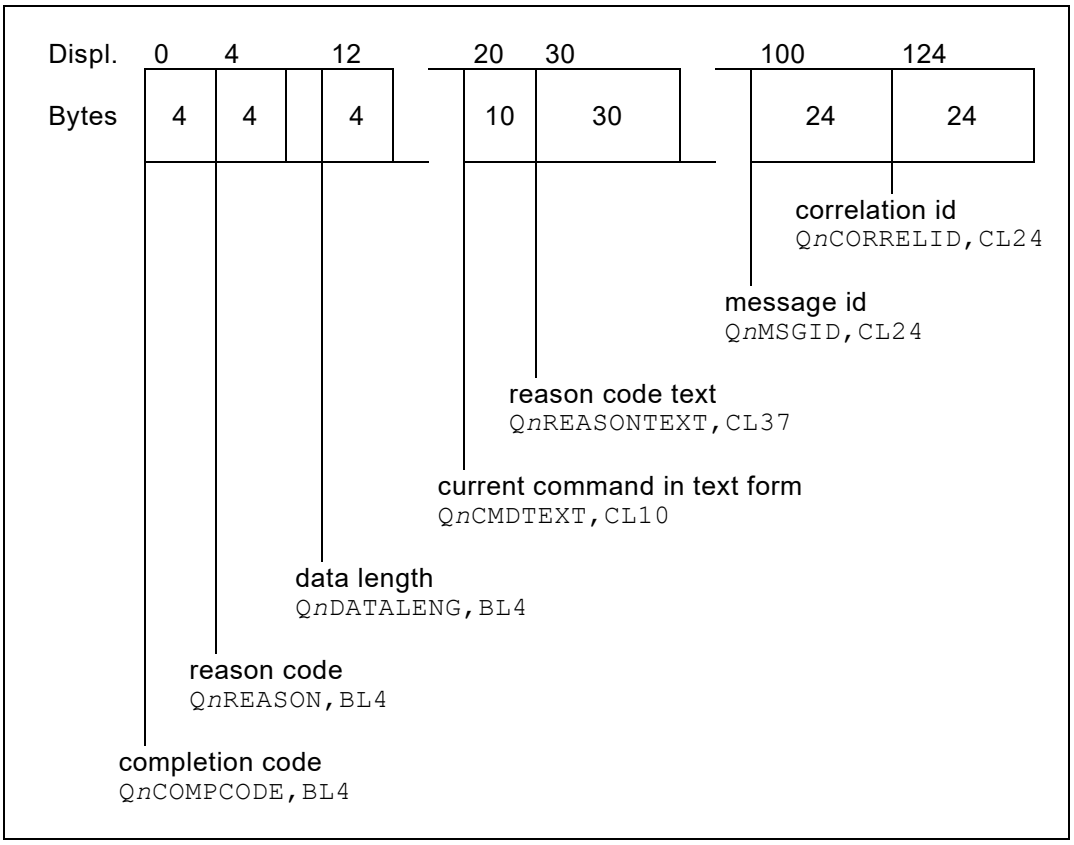


Fig. 273: FCA for MQSeries

Before a PUT command the current message length must be stored into the FCA field **Q*n*DATALENG**. If this length is missing or its value is binary 0 the maximum buffer length is taken as the message length (**MBL=***nnn*).

After a GET command the FCA field **QnDATALENG** contains the current message length received in the queue area. If the defined maximum buffer length is to small a buffer length error is returned.

Areas

The following reserved field symbols are dedicated to the internal areas which are assigned to the individual commands as parameters.
The symbol names correspond to the official ones as described in the macros/books CMQODA, CMQMDA, CMQGMOA, CMQPMOA.

Object Descriptor Area

Symbol	Offset	Length	Fmt	Description
Q*n*OD_AREA	0	168	C	Area name
Q*n*OD_VERSION	4	4	B	Version number
Q*n*OD_OBJECTTYPE	8	4	B	Object type
Q*n*OD_OBJECTNAME	12	48	C	Object name
Q*n*OD_OBJECTQMGRNAME	60	48	C	Object Q manager name
Q*n*OD_DYNAMICQNAME	108	48	C	Dynamic queue name
Q*n*OD_ALTERNATEUSERID	156	12	C	Alternate user identifier

Message Descriptor Area

Symbol	Offset	Length	Fmt	Description
QnMD_AREA	0	324	C	Area name
QnMD_VERSION	4	4	B	Version number
QnMD_REPORT	8	4	B	Report option
QnMD_MSGTYPE	12	4	B	Message type
QnMD_EXPIRY	16	4	B	Expiry time
QnMD_FEEDBACK	20	4	B	Feedback or reason code
QnMD_ENCODING	24	4	B	Data encoding
QnMD_CODECHARSETID	28	4	B	Coded char set identifier
QnMD_FORMAT	32	8	C	Format name
QnMD_PRIORITY	40	4	B	Message priority
QnMD_PERSISTENCE	44	4	B	Message persistence
QnMD_MSGID	48	24	C	Message identifier
QnMD_CORRELID	72	24	C	Correlation identifier
QnMD_BACKOUTCOUNT	96	4	B	Backout counter
QnMD_REPLYTOQ	100	48	C	Name of reply to queue
QnMD_REPLYTOQMGR	148	48	C	Name of reply to Q Mgr
QnMD_USERIDENTIFIER	196	12	C	User identifier
QnMD_ACCOUNTINGTOKEN	208	32	C	Accounting token
QnMD_APPLIDENTITYDATA	240	32	C	Appl data relating to ident
QnMD_PUTAPPLTYPE	272	4	B	Appltype that put the msg
QnMD_PUTAPPLNAME	276	28	C	Applname that put msg
QnMD_PUTDATE	304	8	C	Date when msg was put
QnMD_PUTTIME	312	8	C	Time when msg was put
QnMD_APPLORIGINDATA	320	4	C	Appldata relating to orig

Options that the MQGET Area

Symbol	Offset	Length	Fmt	Description
QnGMO_AREA	0	72	C	Area Name
QnGMO_VERSION	4	4	B	Version number
QnGMO_OPTIONS	8	4	B	Options that control the .
QnGMO_WAITINTERVAL	12	4	B	Wait interval
QnGMO_SIGNAL1	16	4	B	Pointer to signal
QnGMO_SIGNAL2	20	4	B	Signal identifier
QnGMO_RESOLVEDQNAME	24	48	C	Resolved nm of dest que

Options that the MQPUT Area

Symbol	Offset	Length	Fmt	Description
QnGMO_AREA	0	72	C	Area Name
QnPMO_VERSION	4	4	B	Version number
QnPMO_OPTIONS	8	4	B	Options that control the .
QnPMO_TIMEOUT	12	4	B	
QnPMO_CONTEXT	16	4	B	Object handle of input Q
QnPMO_KNOWNDESTCOUNT	20	4	B	
QnPMO_UNKNOWNDESTCOUNT	24	4	B	
QnPMO_INVALIDDESTCOUNT	28	4	B	
QnPMO_RESOLVEDQNAME	32	48	C	Resolved nm of dest que
QnPMO_RESOLVEDQMGRNAME	80	48	C	Resolved nm of dest qmgr

Dead Letter Queue Header Area

Symbol	Offset	Length	Fmt	Description
QnDLH_AREA	0	172	C	Area Name
QnDLH_VERSION	4	4	B	Version Number
QnDLH_REASON	8	4	B	Reason Message arrived
QnDLH_DESTQNAME	12	48	C	Original destination queue
QnDLH_DESTQMGRNAME	60	48	C	Original destination q-mngr
QnDLH_ENCODING	108	4	B	Numeric of data that follows
QnDLH_CODEDCHARSETID	112	4	B	Charset of data that follows
QnDLH_FORMAT	116	8	C	Format name
QnDLH_PUTAPPLTYPE	124	4	B	Type of application that put
QnDLH_PUTAPPLNAME	128	28	C	Name of applicaation that p
QnDLH_PUTDATE	156	8	C	Date when Msg was put
QnDLH_PUTTIME	164	8	C	Time when Msg was put

RFH Header Area

Symbol	Offset	Length	Fmt	Description
QnRFH_AREA	0	32	C	Area Name
QnRFH_VERSION	4	4	B	Version Number
QnRFH_STRUCLENGTH	8	4	B	Total length of MQRFH incl.
QnRFH_ENCODING	12	4	B	Numeric of data that follows
QnRFH_CODEDCHARSETID	16	4	B	Charset of data that follows
QnRFH_FORMAT	20	8	C	Format name
QnRFH_FLAGS	28	4	B	Flags

CICS Bridge Area

Symbol	Offset	Length	Fmt	Description
QnCIH_AREA	0	180	C	Area Name
QnCIH_VERSION	4	4	B	Version Number
QnCIH_STRUCLENGTH	8	4	B	Length of MQCIH struct
QnCIH_ENCODING	12	4	B	Reserved
QnCIH_CODEDCHARSETID	16	4	B	Reserved
QnCIH_FORMAT	20	8	C	Format name of data that
QnCIH_FLAGS	28	4	B	Flags
QnCIH_RETURNCODE	32	4	B	Return code from bridge
QnCIH_COMPCODE	36	4	B	Comcode or CICS EIBRESP
QnCIH_REASON	40	4	B	Reason or CICS EIBRESP2
QnCIH_UOWCONTROL	44	4	B	unit-of-work control
QnCIH_GETWAITINTERVAL	48	4	B	Wait interval for MQGET
QnCIH_LINKTYPE	52	4	B	Link type
QnCIH_OUTPUTDATALENGTH	56	4	B	COMMAREA data length
QnCIH_FACILITYKEEPTIME	60	4	B	Bridge facility release time
QnCIH_ADSDESCRIPTOR	64	4	B	Send/receive ADS descript
QnCIH_CONVERSATIONALTAS	68	4	B	wether task can be convers
QnCIH_TASKENDSTATUS	72	4	B	Status at end of task
QnCIH_FACILITY	76	8	C	BVT token value
QnCIH_FUNCTION	84	4	C	name of CICS EIBFN functi
QnCIH_ABENDCODE	88	4	C	Abend code
QnCIH_AUTHENTICATOR	92	8	C	Password or passticket

QnCIH_RESERVED1	100	8	C	Reserved
QnCIH_REPLYTOFORMAT	108	8	C	format name of reply msg
QnCIH_REMOTESYSID	116	4	C	Remote sysid to use
QnCIH_REMOTETRANSID	120	4	C	Remote transid to attach
QnCIH_TRANSACTIONID	124	4	C	Transaction to attach
QnCIH_FACILITYLIKE	128	4	C	Terminal emulated attributes
QnCIH_ATTENTIONID	132	4	C	AID key
QnCIH_STARTCODE	136	4	C	Transaction start code
QnCIH_CANCELCODE	140	4	C	Abend transaction code
QnCIH_NEXTTRANSACTIONID	144	4	C	Next transaction to attach
QnCIH_RESERVED2	148	8	C	Reserved
QnCIH_RESERVED3	156	8	C	Reserved
QnCIH_CURSORPOSITION	164	4	B	Cursor position
QnCIH_ERROROFFSET	168	4	B	Offset of error in message
QnCIH_INPUTITEM	172	4	B	Item number of last msg
QnCIH_RESERVED4	176	4	B	Reserved

EQUATES of the Different Options and Field Values

Official symbol names which are also used in QPAC are assigned to the different values that must be filled into the area fields according to the application needs. The most important ones are listed here. They correspond exactly to the specifications which can be found in the MQSeries manuals of IBM or in the copy books of the programming languages.
Their detailed meaning can be read in the MQSeries manuals of IBM.

Values Related to MQOPEN

Open Options

```
MQOO_INPUT_AS_Q_DEF                 EQU     1
MQOO_INPUT_SHARED                   EQU     2
MQOO_INPUT_EXCLUSIVE                EQU     4
MQOO_BROWSE                         EQU     8
MQOO_OUTPUT                         EQU     16
MQOO_INQUIRE                        EQU     32
MQOO_SET                            EQU     64
MQOO_SAVE_ALL_CONTEXT               EQU     128
MQOO_PASS_INDENTITY_CONTEXT         EQU     256
MQOO_PASS_ALL_CONTEXT               EQU     512
MQOO_SET_IDENTITY_CONTEXT           EQU     1024
MQOO_SET_ALL_CONTEXT                EQU     2048
MQOO_ALTERNATE_USER_AUTHORITY       EQU     4096
MQOO_FAIL_IF_QUIESCING              EQU     8192
```

Values Related to MQCLOSE

Close Options

```
MQCO_NONE                           EQU     0
MQCO_DELETE                         EQU     1
MQCO_DELETE_PURGE                   EQU     2
```

Values Related to MQGMO

Structure Version

```
MQGMO_CURRENT_VERSION               EQU     1
```

Get Message Options

```
MQGMO_WAIT                          EQU     1
MQGMO_NO_WAIT                       EQU     0
MQGMO_SYNCPOINT                     EQU     2
MQGMO_SYNCPOINT_IF_PERSISTENT       EQU     4096
MQGMO_NO_SYNCPOINT                  EQU     4
MQGMO_MARK_SKIP_BACKOUT             EQU     128
MQGMO_BROWSE_FIRST                  EQU     16
MQGMO_BROWSE_NEXT                   EQU     32
MQGMO_MSG_UNDER_CURSOR              EQU     256
MQGMO_BROWSE_MSG_UNDER_CURSOR       EQU     2048
MQGMO_ACCEPT_TRUNCATED_MSG          EQU     64
MQGMO_SET_SIGNAL                    EQU     8
MQGMO_FAIL_IF_QUIESCING             EQU     8192
MQGMO_CONVERT                       EQU     16384
MQGMO_NONE                          EQU     0
```

Wait Interval

```
MQWI_UNLIMITED                      EQU     1
```

Signal Values

```
MQEC_MSG_ARRIVED                    EQU     2
MQEC_WAIT_INTERVAL_EXPIRED          EQU     3
MQEC_WAIT_CANCELED                  EQU     4
MQEC_Q_MQR_QUIESCING                EQU     5
MQEC_CONNECTION_QUIESCING           EQU     6
```

Values Related to MQPMO

Structure Version

MQPMO_CURRENT_VERSION	EQU	1
MQPMO_CURRENT_LENGTH	EQU	128

Put Message Options

MQPMO_SYNCPOINT	EQU	2
MQPMO_NO_SYNCPOINT	EQU	4
MQPMP_NO_CONTEXT	EQU	16384
MQPMO_DEFAULT_CONTEXT	EQU	32
MQPMO_PASS_IDENTIY_CONTEXT	EQU	256
MQPMO_PASS_ALL_CONTEXT	EQU	512
MQPMO_SET_IDENTITY_CONTEXT	EQU	1024
MQPMO_SET_ALL_CONTEXT	EQU	2048
MQPMO_ALTERNATE_USER_AUTHORITY	EQU	4096
MQPMO_FAIL_IF_QUIESCING	EQU	8192
MQPMO_NONE	EQU	0

Values Related to MQOD Object Descriptor

Structure Version

MQOD_CURRENT_VERSION	EQU	1
MQOD_CURRENT_LENGTH	EQU	168

Object Types

MQOT_Q	EQU	1
MQOT_NAMELIST	EQU	2
MQOT_PROCESS	EQU	3
MQOT_Q_MGR	EQU	4
MQOT_CHANNEL	EQU	5
MQOT_RESERVED_1	EQU	6

Extended Object Types

MQOT_ALL	EQU	1001
MQOT_ALIAS_Q	EQU	1002
MQOT_MODEL_Q	EQU	1003
MQOT_LOCAL_Q	EQU	1004
MQOT_REMOTE_Q	EQU	1005
MQOT_SENDER_CHANNEL	EQU	1007
MQOT_SERVER_CHANNEL	EQU	1008
MQOT_REQUESTER_CHANNEL	EQU	1009
MQOT_RECEIVER_CHANNEL	EQU	1010
MQOT_CURRENT_CHANNEL	EQU	1011
MQOT_SAVED_CHANNEL	EQU	1012

Values related to MQMD Message Descriptor

Structure Version

MQMD_CURRENT_VERSION	EQU	1

Report Options

MQRO_EXCEPTION	EQU	16777216
MQRO_EXCEPTION_WITH_DATA	EQU	50331548
MQRO_EXCEPTION_WITH_FULL_DATA	EQU	117440512
MQRO_EXPIRATION	EQU	2097152
MQRO_EXPIRATION_WITH_DATA	EQU	6291456
MQRO_EXPIRATION_WITH_FULL_DATA	EQU	14680064
MQRO_COA	EQU	256
MQRO_COA_WITH_DATA	EQU	768
MQRO_COA_WITH_FULL_DATA	EQU	1792
MQRO_COD	EQU	2048
MQRO_COD_WITH_DATA	EQU	6144
MQRO_COD_WITH_FULL_DATA	EQU	14336
MQRO_PAN	EQU	1
MQRO_NAN	EQU	2
MQRO_NEW_MSG_ID	EQU	0
MQRO_PASS_MSG_ID	EQU	128
MQRO_COPY_MSG_ID_TO_CORREL_ID	EQU	0
MQRO_PASS_CORREL_ID	EQU	64
MQRO_DEAD_LETTER_Q	EQU	0
MQRO_DISCARD_MSG	EQU	134217728
MQRO_NONE	EQU	0

Message Types

MQMT_SYSTEM_FIRST	EQU	1
MQMT_REQUEST	EQU	1
MQMT_REPLY	EQU	2
MQMT_DATAGRAM	EQU	8
MQMT_REPORT	EQU	4
MQMT_SYSTEM_LAST	EQU	65535
MQMT_APPL_FIRST	EQU	65536
MQMT_APPL_LAST	EQU	999999999

Expiry

MQEI_UNLIMITED	EQU	1

Feedback Values

MQFB_NONE	EQU	0
MQFB_SYSTEM_FIRST	EQU	1
MQFB_QUIT	EQU	256
MQFB_EXPIRATION	EQU	258
MQFB_COA	EQU	259
MQFB_COD	EQU	260
MQFB_PAN	EQU	275
MQFB_NAN	EQU	276
MQFB_CHANNEL_COMPLETED	EQU	262
MQFB_CHANNEL_FAIL_RETRY	EQU	263
MQFB_CHANNEL_FAIL	EQU	264
MQFB_APPL_CANNOT_BE_STARTED	EQU	265
MQFB_TM_ERROR	EQU	266
MQFB_APPL_TYPE_ERROR	EQU	267
MQFB_STOPPED_BY_MSG_EXIT	EQU	268
MQFB_XMIT_Q_MSG_ERROR	EQU	271
MQFB_DATA_LENGTH_ZERO	EQU	291
MQFB_DATA_LENGTH_NEGATIVE	EQU	292
MQFB_DATA_LENGHT_TOO_BIG	EQU	293
MQFB_BUFFER_OVERFLOW	EQU	294
MQFB_LENGTH_OFF_BY_ONE	EQU	295
MQFB_IIH_ERROR	EQU	296
MQFB_NOT_AUTHORIZED_FOR_IMS	EQU	298
MQFB_IMS_ERROR	EQU	300
MQFB_IMS_FIRST	EQU	301
MQFB_IMS_LAST	EQU	399
MQFB_CICS_INTERNAL_ERROR	EQU	401
MQFB_CICS_NOT_AUTHORIZED	EQU	402
MQFB_CICS_BRIDGE_FAILURE	EQU	403
MQFB_CICS_CORREL_ID_ERROR	EQU	404
MQFB_CICS_CCSID_ERROR	EQU	405
MQFB_CICS_ENCODING_ERROR	EQU	406
MQFB_CICS_CIH_ERROR	EQU	407
MQFB_CICS_UOW_ERROR	EQU	408

```
MQFB_CICS_COMMAREA_ERROR                EQU         409
MQFB_CICS_APPL_NOT_STARTED              EQU         410
MQFB_CICS_APPL_ABENDED                  EQU         411
MQFB_CICS_DLQ_ERROR                     EQU         412
MQFB_CICSD_UOW_BACKED_OUT               EQU         413
MQFB_SYSTEM_LAST                        EQU       65535
MQFB_APPL_FIRST                         EQU       65536
MQFB_APPL_LAST                          EQU   999999999
```

Encoding

```
MQENC_NATIVE                            EQU         785
```

Encoding for Binary Integers

```
MQENC_INTEGER_UNDEFINED                 EQU           0
MQENC_INTEGER_NORMAL                    EQU           1
MQENC_INTEGER_REVERSED                  EQU           2
```

Encoding for Packed-Decimal Integers

```
MQENC_DECIMAL_UNDEFINED                 EQU           0
MQENC_DECIMAL_NORMAL                    EQU          16
MQENC_DECIMAL_REVERSED                  EQU          32
```

Coded Character-Set Identifiers

```
MQCCSI_DEFAULT                          EQU           0
MQCCSI_Q_MGR                            EQU           0
MQCCSI_EMBEDDED                         EQU          -1
```

Priority

```
MQPRI_PRIORITY_AS_Q_DEF                 EQU          -1
```

Persistence Values

```
MQPER_PERSISTENT                        EQU           1
MQPER_NOT_PERSISTENT                    EQU           0
MQPER_PERSISTENCE_AS_Q_DEF              EQU           2
```

Message Flags

```
MQMF_SEGMENTATION_INHIBITED             EQU           0
MQMF_SEGMENTATION_ALLOWED               EQU           1
MQMF_MSG_IN_GROUP                       EQU           8
MQMF_LAST_MSG_IN_GROUP                  EQU          16
MQMF_SEGMENT                            EQU           2
MQMF_LAST_SEGMENT                       EQU           4
MQMF_NONE                               EQU           0
```

Values Related to MQINQ Call

Character-Attribute Selectors

```
MQCA_APPL_ID                            EQU        2001
MQCA_BACKOUT_REQ_Q_NAME                 EQU        2019
MQCA_BASE_Q_NAME                        EQU        2002
MQCA_CHANNEL_AUTO_DEF_EXIT              EQU        2026
MQCA_COMMAND_INPUT_Q_NAME               EQU        2003
MQCA_CREATION_DATE                      EQU        2004
MQCA_CREATION_TIME                      EQU        2005
MQCA_DEAD_LETTER_Q_NAME                 EQU        2006
MQCA_DEF_XMIT_Q_NAME                    EQU        2025
MQCA_ENV_DATA                           EQU        2007
MQCA_FIRST                              EQU        2001
MQCA_INITIATION_Q_NAME                  EQU        2008
MQCA_LAST                               EQU        4000
MQCA_LAST_USED                          EQU        2026
MQCA_NAMELIST_DESC                      EQU        2009
MQCA_NAMELIST_NAME                      EQU        2010
MQCA_NAMES                              EQU        2020
MQCA_PROCESS_DESC                       EQU        2011
MQCA_PROCESS_NAME                       EQU        2012
MQCA_Q_DESC                             EQU        2013
MQCA_Q_MGR_DESC                         EQU        2014
MQCA_Q_MGR_NAME                         EQU        2015
MQCA_Q_NAME                             EQU        2016
MQCA_REMOTE_Q_MGR_NAME                  EQU        2017
MQCA_REMOTE_Q_NAME                      EQU        2018
MQCA_STORAGE_CLASS                      EQU        2022
```

```
MQCA_TRIGGER_DATA                       EQU     2023
MQCA_USER_DATA                          EQU     2021
MQCA_XMIT_Q_NAME                        EQU     2024
```

Integer_Attribute Selectors

```
MQIA_APPL_TYPE                          EQU       1
MQIA_AUTHORITY_EVENT                    EQU      47
MQIA_BACKOUT_THRESHOLD                  EQU      22
MQIA_CHANNEL_AUTO_DEF                   EQU      55
MQIA_CHANNEL_AUTO_DEF_EVENT             EQU      56
MQIA_CODED_CHAR_SET_ID                  EQU       2
MQIA_COMMAND_LEVEL                      EQU      31
MQIA_CPI_LEVEL                          EQU      27
MQIA_CURRENT_Q_DEPTH                    EQU       3
MQIA_DEF_INPUT_OPEN_OPTION              EQU       4
MQIA_DEF_PERSISTENCE                    EQU       5
MQIA_DEF_PRIORITY                       EQU       6
MQIA_DEFINITION_TYPE                    EQU       7
MQIA_DIST_LISTS                         EQU      34
MQIA_FIRST                              EQU       1
MQIA_HARDEN_GET_BACKOUT                 EQU       8
MQIA_HIGH_Q_DEPTH                       EQU      36
MQIA_INDEX_TYPE                         EQU      57
MQIA_INHIBIT_EVENT                      EQU      48
MQIA_INHIBIT_GET                        EQU       9
MQIA_INHIBIT_PUT                        EQU      10
MQIA_LAST                               EQU    2000
MQIA_LAST_USED                          EQU      57
MQIA_LOCAL_EVENT                        EQU      49
MQIA_MAX_HANDLES                        EQU      11
MQIA_MAX_MSG_LENGTH                     EQU      13
MQIA_MAX_PRIORITY                       EQU      14
MQIA_MAX_Q_DEPTH                        EQU      15
MQIA_MAX_UNCOMMITTED_MSGS               EQU      33
MQIA_MSG_DELIVERY_SEQUENCE              EQU      16
MQIA_MSG_DEQ_COUNT                      EQU      38
MQIA_MSG_ENQ_COUNT                      EQU      37
MQIA_NAME_COUNT                         EQU      19
MQIA_OPEN_INPUT_COUNT                   EQU      17
MQIA_OPEN_OUTPUT_COUNT                  EQU      18
MQIA_PERFORMANCE_EVENT                  EQU      53
MQIA_PLATFORM                           EQU      32
MQIA_Q_DEPTH_HIGH_EVENT                 EQU      43
MQIA_Q_DEPTH_HIGH_LIMIT                 EQU      40
MQIA_Q_DEPTH_LOW_EVENT                  EQU      44
MQIA_Q_DEPTH_LOW_LIMIT                  EQU      41
MQIA_Q_DEPTH_MAX_EVENT                  EQU      42
MQIA_Q_SERVICE_INTERVAL                 EQU      54
MQIA_Q_SERVICE_INTERVAL_EVENT           EQU      46
MQIA_Q_TYPE                             EQU      20
MQIA_REMOTE_EVENT                       EQU      50
MQIA_RETENTION_INTERVAL                 EQU      21
MQIA_SCOPE                              EQU      45
MQIA_SHAREABILTIY                       EQU      23
MQIA_START_STOP_EVENT                   EQU      52
MQIA_SYNCPOINT                          EQU      30
MQIA_TIME_SINCE_RESET                   EQU      35
MQIA_TRIGGER_CONTROL                    EQU      24
MQIA_TRIGGER_DEPTH                      EQU      29
MQIA_TRIGGER_INTERVAL                   EQU      25
MQIA_TRIGGER_MSG_PRIORITY               EQU      26
MQIA_TRIGGER_TYPE                       EQU      28
MQIA_USAGE                              EQU      12
```

Chapter 15. CICS External Interface Support Feature (EXCI)

EXCI External Batch to CICS Communication Definition

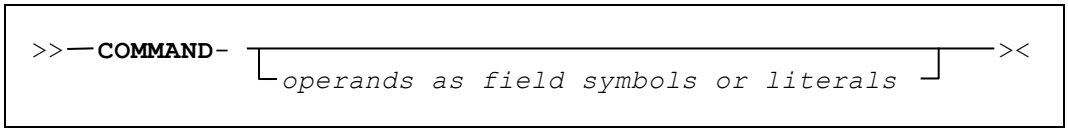
Communication between Batch and CICS is no longer available due to the sunset of QPAC for CICS (QPAC-Online) as of QPAC Release 9.10

Chapter 16. ISPF/PDF Support Feature

ISPF/TSO Command Definition

IBM ISPF under TSO MVS is fully supported by this feature. It allows the programming of complete ISPF applications instead of using for example CLIST. This feature allows an ISPF application to use known file organizations supported by CLIST and at the same time process PDS and SAM or VSAM, data bases like DB2 or SYSOUT data, and last but not least MQSeries. The complete command set of QPAC-Batch is available.

Basic Format of the ISPF Command Definitions



All official ISPF commands that are listed below are supported. Therefore the command constant, which is normally defined within the CALL ISPLINK command under CLIST, is the QPAC command itself followed by a hyphen.
The operands correspond to the normal syntax as described in the IBM manual "**ISPF Services Guide"** or "**ISPF Reference Summary"**. Not supported are are operands related to DBCS.

Following the commands are listed. The corresponding description of the operands are to be consulted in the official IBM manual. Only an overview and some hints are given here.

Command	Operanden
	Start Pop-Up Window Mode
ADDPOP-	[*field-name*] [,*row*][,*column*]
	(*not yet supported*) ***Browse Interface***
BRIF-	[ ]
	Browse a Data Set
BROWSE-	*Dsname* ,[*serial*],[*pswd-value*],[*panel-name*] *or* *Data-id* ,[*member-name*],[*format-name*]

Set Processing Modes

CONTROL-

```
'DISPLAY' [
   ,'LOCK'
   ,'LINE',line-number
   ,'SM',line-number
   ,'REFRESH'
   ,'SAVE' or 'RESTORE'
   ,'ALLVALID'
   ]
'NONDISPL' [
   ,'ENTER' or 'END'
   ]
'ERRORS' [
   ,'CANCEL' or 'RETURN'
   ]
'SPLIT'
   ,'ENABLE'
   ,'DISABLE'
'NOCMD'
'SUBTASK'
   ,'PROTECT'
   ,'CLEAR'
'TSOGUI' [
   ,'QUERY' or 'OFF' or 'ON'
   ]
   ,'REFLIST' [
   ,'UPDATE', or 'NOUPDATE'
   ]
```

Display Panels and Messages

DISPLAY-

```
   [panel-name] [,msg-id] [,cursor-field-name]
   [,cursor-position] [,stack-buffer-name]
   [,ret-buffer-name] [,ret-length-name]
   [,message-field-name]
```

Initialize Edit Recovery

EDIREC-

```
'INIT'
   ,command-name
'QUERY'
'CANCEL'
'DEFER'
```

Edit a Data Set

EDIT-

```
dsname, ,[serial]
   ,[pswd-value] ,[panel-name] ,[macro-name]
   ,[profile-name]
,data-id
   ,[member-name] ,[format-name]
   ,['YES' or 'NO']
```

Specify Edit Recovery Handling

EDREC-

```
'INIT' [,command-name]
'QUERY'
'PROCESS' [,pswd-value][,data-id]
'CANCEL'
'DEFER
```

Statistics for a file

FILESTAT-

```
var-name
   ,[var-name,var-name]
```

Upload or Download File

FILEXFER- *host_var*,*ws_war*,'HOST' *or* 'WS',
[*volume*,'BINARY' *or* 'TEXT',
'STATS' *or* 'NOSTATS',
'YES' *or* 'NO'
]

End File Tailoring

FTCLOSE- [*member-name*] [,*library*] [,'NOREPL']

Erase File Tailoring Output

FTERASE- *member-name* [,*library*]

Include a Skeleton

FTINCL- *skel-name* [,'NOFT']

Begin File Tailoring

FTOPEN- ['TEMP']

Get a Message

GETMSG- *message-id*
[,*short-message-name*]
[,*long-message-name*]
[,*alarm-name*]
[,*help-name*]
[,*type-name*]
[,*window-name*]
[,*ccsid-name*]

Allocate Application Libraries

LIBDEF- *lib-type*
[,'DATASET' *or* 'EXCLDATA' *or*
'LIBRARY' *or* 'EXCLLIBR']
[,*dataset-list or libname*]
[,'COND' *or* 'UNCOND'] *or* 'STACK'

Write Lines to the List Data Set

LIST- *dialog-variable-name*,*line-length*
[,'PAGE']
[,'SINGLE' *or* 'DOUBLE' *or* 'TRIPLE']
[,OVERSTRK']
[,'CC']

Activate a Promotion Hierarchy

LMACT- *project*,*top-group*

Close a Data Set

LMCLOSE- *data-id*

Compresses a Partitioned Data Set

LMCOMP- *data-id*

Copy Members of a Data Set

LMCOPY- *from-data-id*
,[*from-member-name*]
,*to-data-id*
,[*to-member-name*]
,['REPLACE']
,['PACK' ,['TRUNC'] ,['LOCK']

Data Set List Service

LMDDISP- *dslist-id*
,['VOLUME' *or* 'SPACE' *or* 'ATTRIB' *or* 'TOTAL']
,['YES' *or* 'NO']
,[*panel-name*]

Deactivate a Promotion Hierarchy

LMDEACT- *project*,*top-group*

Free a Data Set List ID

LMDFREE- *list-id*

Initialize a Data Set List

LMDINIT- *dslist-id-var*
,{*dsname-level*}
,{*volume-serial*}

List Data Sets

LMDLIST- *dslist-id*
,'LIST' *or* 'FREE' *or* 'SAVE'
,*dataset-var*
,['YES' *or* 'NO'
,[*group*]

Erase a Data Set

LMERASE- {*project group type*}
,{*dataset*}
,['YES' *or* 'NO']

Free Data Set from its Association with Data ID

LMFREE- *data-id*

Read a Logical Record from a Data Set

LMGET- *data-id*
,'MOVE' *or* 'LOCATE' *or* 'INVAR'
,*dataloc-var*
,*datalen-var*
,*max-length*

Create a Table with the Hierarchy Structure

LMHIER- *project,group,table-name*

Generate a Data ID for a Data Set

LMINIT- *data-id-var*
,{*project*,*group1* [,*group2*]
[,*group3*] [,*group4*] ,*type*}
,{*dsname*} ,{*ddname*}
,[*serial*] ,[*password*]
,['SHR' *or* 'EXCLU' *or* 'SHRW' *or* 'MOD']
,[*org-var*]

Add a Member to a Data Set

LMMADD- *data-id*, *member-name*
,['YES' *or* 'NO'],['NOENQ']

Delete a Member from a Data Set

LMMDEL- *data-id*, *member-name*
,['NOENQ']

Member List Service

LMMDISP- *data-id*
,['DISPLAY']
,[*pattern*]
,['YES' *or* 'NO']
,[*panel-name*]
,['ZCMD' *or* 'ZLLCMD' *or* 'ZLUDATA'
,[*top-row*]
,' ',' '
,['S' *or* 'ANY']
,[1 *or* 9]
,['ALLOWNEW'

data-id
,'GET' ,' ',['YES' *or* 'NO']

data-id
,'PUT',*member-name*,' ',' ',' ',' '
,[*lcmd-value*],[*udata-value*]

data-id
,'ADD',*member-name*,' ',' ',' ',' '
,[*lcmd-value*],[*udata-value*]

data-id
,'FREE'

Find a Library Member

LMMFIND- *data-id*,*member-name*
,['LOCK'
,[*lrecl-var*]
,[*recfm-var*]
,[*group-var*]
,['YES' *or* 'NO'

List a Library's Members

LMMLIST- *data-id*
,['LIST' *or* 'FREE' *or* 'SAVE']
,[*member-var*]
,['YES' *or* 'NO'
,[*group*]
,[*member-pattern*]

Move Members of a Data Set

LMMOVE- *from-data-id*
,[*from-member-name*]
,*to-data-id*
,[*to-member-name*]
,['REPLACE']
,['PACK']
,['TRUNC']
,['YES' *or* 'NO']

Rename a Data Set Member

LMMREN- *data-id*
,*old-member-name*,*new-member-name*
,['NOENQ']

Replace a Member of a Data Set

LMMREP- *data-id*,*member-name*
,['YES' *or* 'NO']
,['NOENQ']

Set and Store, or Delete ISPF Statistics

LMMSTATS- *data-id,member-name*
,[*version-number*],[*mode-level*],[*create-date*]
,[*last-modified-date*],[*last-modified-time*]
,[*current-size*]
,[*initial-size*],[*records-modified*],[*user-id*]
,['DELETE']
,[*4-char-year-create-date*]
,[*4-char-year-last-modified-date*]
,[ON *or* OFF *or* ASIS]

Open a Data Set

LMOPEN- *data-id*
,['INPUT' *or* 'OUTPUT'
,[*lrecl-var*],[*recfm-var*],[*org-var*]

Print a Partitioned or Sequential Data Set

LMPRINT- *data-id,member-name*
,['INDEX']
,['YES' *or* 'NO']

Promote a Data Set or Member to Another

LMPROM- {*from-project*,*fromgroup*,*from-type*,*from-member*}
,{*dsname*}
,[*serial*],[*password*]
,['MOVE',[*reason-code*]
,['YES' *or* 'NO']
,[*to-project*],[*to-group*],[*to-type*],[*to-member*]

Write a Logical Record to a Data Set

LMPUT- data-id
,'INVAR' *or* 'MOVE'
,*dataloc-var*,*data-length*
,' ',['NOBSCAN'

Give a Dialog Information about a Data Set

LMQUERY- *data-id*
,[*proj-var*],[*group1-var*],[*group2-var*]
,[*group3-var*],[*group4-var*],[*type-var*]
,[*dsn-var*],[*ddn-var*],[*serial-var*],[*enq-var*]
,[*open-var*],[*lrecl-var*],[*recfm-var*],[*dsorg-var*]
,[*alias-var*],[*password-var*]

Rename an ISPF Library

LMRENAME- *project*,*group*,*type*
,{ [*new-project*],[*new-group*],[*new-type*] }

Create a Data Set Containing Controls Info

LMREVIEW- LIBRARY *or* MEMBER
,*data-id*,*dataset*
,[*datamemb*]
,*project*,*topgroup*,*type*
,[*member*]

Write a Message to the Log Data Set

LOG- *message-id*

Obtain Panel Information

PQUERY- *panel-name*,*area-name*
[,*area-type-name*]
[,*area-width-name*]
[,*area-depth-name*]
[,*row-number-name*]
[*column-number-name*]

Query LIBDEF Definition Information

QLIBDEF- *lib-type*
[,*type-var*]
[,*id-var*]

Remove a Pop-Up Window

REMPOP- ['ALL']

Select a Panel or Function

SELECT- *length*,*keywords*

Set Next Message

SETMSG- *message-id*,
,'COND']
[,*message-field-name*]

Add a Row to a Table

TBADD- *table-name*
[,*name-list*]
[,'ORDER']
[,*number-of-rows*]

Set the Row Pointer to Bottom

TBBOTTOM- *table-name*
[,*var-name*]
[,*rowid-name*]
[,'NOREAD']
[,*crp-name*]

Close and Save a Table

TBCLOSE- *table-name*
[,'NEWCOPY' *or* 'REPLCOPY']
[,*alt-name*]
[,*percentage*]
[,*library*]

Create a New Table

TBCREATE- *table-name*
[,*key-name-list*]
[,*name-list*]
[,'WRITE' *or* 'NOWRITE']
[,'REPLACE']
[,*library*]
[,'SHARE']

Delete a Row from a Table

TBDELETE- *table-name*

Display Table Information

TBDISPL- *table-name*
[,*panel-name*]
[,*message-id*
[,*field-name*]
[,*table-row-number*]
[,*cursor-position*]
[,'YES' *or* 'NO']
[,*crp-name*]
[,*rowid-name*]
[,*message-field-name*]

Close a Table without Saving

TBEND- *table-name*

Erase a Table

TBERASE- *table-name*
[,*library*]

Determine whether a Row exists in a Table

TBEXIST- *table-name*

Retrieve a Row from a Table

TBGET- *table-name*
[,*var-name*]
[,*rowid-name*]
[,'NOREAD']
[,*crp-name*]

Modify a Row in a Table

TBMOD- *table-name*
[,*name-list*]
[,'ORDER']

Open a Table

TBOPEN- *table-name*
[,'WRITE' *or* 'NOWRITE']
[,*library*]
[,'SHARE']

Update a Row in a Table

TBPUT- *table-name*
[,*name-list*
[,'ORDER']

Obtain Table Information

TBQUERY- *table-name*
[,*key-name*]
[,*var-name*]
[,*rownum-name*]
[,*keynum-name*]
[,*namenum-name*]
[,*crp-name*]

Define a Search Argument

TBSARG- *table-name*
[,*name-list*]
[,'NEXT' *or* 'PREVIOUS']
[,*name-cond-pairs*]

Save a Table

TBSAVE- *table-name*
[,'NEWCOPY' *or* 'REPLCOPY']
[,*alt-name*]
[,*percentage*]
[,*library*]

Search a Table

TBSCAN- *table-name*
[,*name-list*]
[,*var-name*]
[,*rowid-name*]
[,'NEXT' *or* 'PREVIOUS']
[,'NOREAD']
[,*crp-name*]
[,*condition-value-list*]

Move the Row Pointer

TBSKIP- *table-name*
[,*number*]
[,*var-name*]
[,*rowid-name*]
[,*rowid*]
[,'NOREAD']
[,*crp-name*]

Sort a Table

TBSORT- *table-name*,*sort-list*

Retrieve Table Statistics

TBSTATS- *table-name*
[,*date-created-name*] [,*time-created-name*]
[,*date-updated-name*] [,*time-updated-name*]
[,*user-name*] [,*row-created-name*]
[,*rownum-name*]
[,*row-updated-name*] [,*table-update-name*]
[,*service-name*] [,*return-code-name*]
[,*status1-name*] [,*status2-name*]
[,*status3-name*]
[,*library*] [,*date-created-name-4-digit*]
[,*date-updated-name-4-digit*]

Set the Row Pointer to the Top

TBTOP- *table-name*

Clear Table Variables

TBVCLEAR- *table-name*

Translate CCSID Data

TRANS- *from-ccsid-number*,*to-ccsid-number*
,*from-variable-name*
[,*to-variable-name*] [,*data-length*]

Create a Copy of a Variable

VCOPY- *name-list*,*length-array*,*value-array*
[,'LOCATE' *or* 'MOVE']

Define Function Variables

VDEFINE- *name-list*,*variable*,*format*,*length*
[,*options-list*][,*user-data*]

Remove a Definition of Function Variables

VDELETE- *name-list*

Remove Variables from Shared or Profile Pool

VERASE- *name-list*
[,'ASIS' *or* 'SHARED' *or* 'PROFILE' *or* 'BOTH'

Retrieve Variables from a Pool or Profile

VGET- *name-list*
[,'ASIS' *or* 'SHARED' *or* 'PROFILE']

View a Data Set

VIEW- *dsname*
,[*serial*]
,[*pswd-value*],[*panel-name*],[*macro-name*]
,[*profile-name*],[*data-id*],[*member-name*]
,[*format-name*],['YES' *or* 'NO'],['YES' *or* 'NO']

Mask and Edit Processing

VMASK- *name-list*
{,'FORMAT'{,'IDTE'}
{,'STDDATE'}
{,'ITIME'}
{,'STDTIME'}
{,'JDATE'}
{,'JSTD'}
{,'USER','*mask*',*masklen*

Update Variables in the Shared or Profile Pool

VPUT- *name-list*
[,'ASIS' *or* 'SHARED' *or* 'PROFILE']

Replace a Variable

VREPLACE- *name-list*,*lengths*,*values*

Reset Function Variables

VRESET-

Example of Syntax: QPAC Program Example QPACETBH

```
PARM=LIST,NOLOG,NOLOGTIT,NOCHECK,WORK=30000
* ------------------------------------------------------------
* FILE DEFINITIONS
* ------------------------------------------------------------
IPF9=*QPETBHC0,VS,WP=WPOS5001  *. PRIMARY
IPF1=*QPETBHP1,VS,WP=WPOS5001  *. ALTERNATE INDEX P1
01=D1NAME,CL30
  =D1LNR,ZL5
  =D1PIN,CL7
  =D1ANREDE,CL8
  =D1PSA,CL3
  =D1TCODE,CL6
  =D1TELNR,CL5
  =D1LNRTEL,ZL5
  =D1OE,CL4
  =D1LNROE,ZL5
  =D1PST,CL4
  =D1ORT,CL4
  =D1RAUM,CL5
  =D1LNRRAUM,ZL5
  =D1KOS,CL6
  =D1LNRKOS,ZL5
  =D1FAXCODE,CL6
  =D1FAXNR,CL5
IPF2=*QPETBHP2,VS,WP=WPOS5001  *. ALTERNATE INDEX P2
IPF3=*QPETBHP3,VS,WP=WPOS5001  *. ALTERNATE INDEX P3
IPF4=*QPETBHP4,VS,WP=WPOS5001  *. ALTERNATE INDEX P4
* ------------------------------------------------------------
* WORKAREA
* ------------------------------------------------------------
01W=NAMELIST,CL255
   =S1NAME,CL30
   =S1TELNR,CL5
   =S1OE,CL4
   =S1RAUM,CL5
   =S1KOS,CL6
   =S1FAXNR,CL5
* ------------------------------------------------------------
   =ISPLEN,BL4
* ------------------------------------------------------------
   =QSCAN,CL4
   =QSARG,CL255

* ------------------------------------------------------------
* SET CONTROL
* ------------------------------------------------------------
CONTROL-'ERRORS  ','RETURN '
* ------------------------------------------------------------
* VDEFINES
* ------------------------------------------------------------
 VDEFINE-'NAMELIST',NAMELIST
* ------------------------------------------------------------
* DEFINE SEARCH ARGUMENTS
* ------------------------------------------------------------
 VDEFINE-'S1NAME',S1NAME
```

```
 VDEFINE-'S1TELNR',S1TELNR
 VDEFINE-'S1FAXNR',S1FAXNR
 VDEFINE-'S1OE',S1OE
 VDEFINE-'S1RAUM',S1RAUM
 VDEFINE-'S1KOS',S1KOS
 VDEFINE-'QSCAN',QSCAN
 SET QSCAN  =  'ALL'
 VDEFINE-'QSARG',QSARG
* ------------------------------------------------------------
* DEFINE TABLE VARIABLE NAMES
* ---------------------------- ------------------------------
 VDEFINE-'D1NAME',D1NAME
 VDEFINE-'D1TCODE',D1TCODE
 VDEFINE-'D1TELNR',D1TELNR
 VDEFINE-'D1OE',D1OE
 VDEFINE-'D1RAUM',D1RAUM
 VDEFINE-'D1KOS',D1KOS
 VDEFINE-'D1PST',D1PST
 VDEFINE-'D1ORT',D1ORT
 VDEFINE-'D1FAXNR',D1FAXNR
 VDEFINE-'D1PSA',D1PSA
* ------------------------------------------------------------
* CREATE TABLE FIRST
* ------------------------------------------------------------
 SET  NAMELIST   =    '('
                      'D1PSA    '      !
                      'D1NAME   '      !
                      'D1TCODE  '      !
                      'D1TELNR  '      !
                      'D1OE     '      !
                      'D1PST    '      !
                      'D1ORT    '      !
                      'D1RAUM   '      !
                      'D1KOS    '      !
                      'D1FAXNR  '      !
                      ')'



 TBCREATE-'ETB     ',,NAMELIST,'NOWRITE','REPLACE'
 TBADD-'ETB     ',NAMELIST
 TBDELETE-'ETB     '
* ------------------------------------------------------------
*
* ------------------------------------------------------------
 OPEN-I9
 DO-FOREVER
 GET-I9 AT-EOF  DOQUIT      ATEND
 TBADD-'ETB     ',NAMELIST
 DOEND
* ------------------------------------------------------------
*
* ------------------------------------------------------------
 CONTROL-'NONDISPL','ENTER'
 TBTOP-'ETB     '
 TBDISPL-'ETB     ','QPETBH01'
```

```
DO-WHILE RC < 8
 TBDISPL-'ETB     '
 IF  RC > 4 THEN GOTO EXIT_ETB  IFEND
 SET QSCAN  = 'SCAN'
* ---------------------------------------------------------------
* SET SEARCH ARGUMENT
* ---------------------------------------------------------------
 TBVCLEAR-'ETB     '
* ---------------------------------------------------------------
 SET QSCAN  = 'SCAN'
* ---------------------------------------------------------------
 SET D1NAME   =  S1NAME
 SET D1TELNR  =  S1TELNR
 SET D1OE     =  S1OE
 SET D1RAUM   =  S1RAUM
 SET D1KOS    =  S1KOS
* ---------------------------------------------------------------
 SET QSARG    =             '('              !
                            'D1NAME,GE,'     !
                            'D1TELNR,GE,'    !
                            'D1OE,GE,'       !
                            'D1RAUM,GE,'     !
                            'D1KOS,GE)'
* ---------------------------------------------------------------
 TBSARG-'ETB     ',' ',' ',' ',QSARG
 CONTROL-'NONDISPL','ENTER'
 TBDISPL-'ETB     ','QPETBH01'
 IF  RC > 4 THEN GOTO EXIT_ETB  IFEND
 DOEND
* ---------------------------------------------------------------
EXIT_ETB:
 TBCLOSE-'ETB     '
 VRESET-
 CLOSE-I9
END
```

Panel Definition: Example QPACETBH01

```
)ATTR DEFAULT(%+!)
 ç TYPE(TEXT)   COLOR(WHITE)  INTENS(LOW)
 $ TYPE(TEXT)   COLOR(TURQ  ) INTENS(LOW)
 ! TYPE(TEXT)   COLOR(red   ) INTENS(LOW)
 ` TYPE(TEXT)   COLOR(BLUE )  INTENS(HIGH)
 # TYPE(OUTPUT) COLOR(YELLOW) INTENS(LOW) JUST(ASIS)
 _ TYPE(INPUT ) COLOR(RED)   INTENS(LOW) JUST(ASIS) HILITE(USC
 * TYPE(OUTPUT) COLOR(GREEN)  INTENS(LOW) JUST(ASIS)
)BODY
%---------------------- Internes Telefonbuch -----------------
%COMMAND ===>_ZCMD                          %SCROLL ===>_SCIN
%
%Generischer Such-Begriff
+Name:_Z                        +TelNr:_Z    +OE:_Z   +Raum:_Z
%
%PSA Vorwa/TelNr Name          OE   PST  ORT  RAUM  KOS    FAX
%-------------------------------------------------------------
)MODEL ROWS(&QSCAN)
*Z  *Z     *Z    *Z           *Z   *Z   *Z   *Z    *Z     *Z
)INIT

 .HELP = QHETBH01


 .ZVARS = '(                  +
            S1NAME            +
            S1TELNR           +
            S1OE              +
            S1RAUM            +
            S1KOS             +
            D1PSA             +
            D1TCODE           +
            D1TELNR           +
            D1NAME            +
            D1OE              +
            D1PST             +
            D1ORT             +
            D1RAUM            +
            D1KOS             +
            D1FAXNR           +
           )'

 VGET       (S1NAME
             S1TELNR
             S1OE
             S1RAUM
             S1KOS
             S1FAXNR) PROFILE

 IF (&S1NAME   = &Z) &S1NAME   = '*'
 IF (&S1TELNR  = &Z) &S1TELNR  = '*'
 IF (&S1OE     = &Z) &S1OE     = '*'
 IF (&S1RAUM   = &Z) &S1RAUM   = '*'
 IF (&S1KOS    = &Z) &S1KOS    = '*'
 IF (&S1FAXNR  = &Z) &S1FAXNR  = '*'
```

```
)REINIT
     REFRESH (ZCMD
              S1NAME
              S1TELNR
              S1OE
              S1RAUM
              S1KOS
                     )

)PROC

 IF (&S1NAME   = &Z) &S1NAME   = '*'
 IF (&S1TELNR  = &Z) &S1TELNR  = '*'
 IF (&S1OE     = &Z) &S1OE     = '*'
 IF (&S1RAUM   = &Z) &S1RAUM   = '*'
Command ===>                                    Scroll ===> PAGE
 IF (&S1KOS    = &Z) &S1KOS    = '*'
 IF (&S1FAXNR  = &Z) &S1FAXNR  = '*'

 VPUT       (S1NAME
             S1TELNR
             S1OE
             S1RAUM
             S1KOS
             S1FAXNR) PROFILE

)END
```

CLIST Definition Example

```
 PROC   0
 CONTROL MAIN  NOFLUSH NOLIST NOCONLIST NOSYMLIST MSG
 /*     CALL QPAC ISPF                                  */
 ISPEXEC LIBDEF ISPLLIB DATASET ID('QPAC.LOADLIB')  STACK
 ALLOC FI(QPACPGM) DA('USER.QPGM') SHR REUS
 ALLOC FI(QPACLIST) SYSOUT(A)
 ALLOC FI(QPETBHC0) DA('VSAM.HOSTETB.OSYS.C0') SHR REUS
 ALLOC FI(QPETBHP1) DA('VSAM.HOSTETB.OSYS.P1') SHR REUS
 ALLOC FI(QPETBHP2) DA('VSAM.HOSTETB.OSYS.P2') SHR REUS
 ALLOC FI(QPETBHP3) DA('VSAM.HOSTETB.OSYS.P3') SHR REUS
 ALLOC FI(QPETBHP4) DA('VSAM.HOSTETB.OSYS.P4') SHR REUS
 ISPEXEC  SELECT PGM(QPAC)  PARM(QPGM=QPACETBH)  +
          NEWAPPL(ETBH) NEWPOOL PASSLIB  MODE(FSCR)
 FREE  FI(QPETBHC0)
 FREE  FI(QPETBHP1)
 FREE  FI(QPETBHP2)
 FREE  FI(QPETBHP3)
 FREE  FI(QPETBHP4)
 FREE  FI(QPACLIST)
 FREE  FI(QPACPGM)
 ISPEXEC LIBDEF ISPLLIB
 END
```

Appendix A. Basic Instruction Formats (Summary)

Overview

In this appendix the most important instruction formats of the **former QPAC-Batch Basic Part** are summarized.
These instruction formats do not support symbolic addressing nor do they support any form of automatic data conversion and should therefore, whenever possible, be replaced by the new high-level-format of the QPAC `SET` instruction or at least by the implicit position symbols.

```
IPOS1,OPOS1,80
```

Imperative Instructions and Operations

General Formats

```
FROM-ADDRESS,TO-ADDRESS,OPERATION,LENGTHS

1. FROM-ADDR,TO-ADDR,OPERATION,FROM-LENGTH,TO-LENGTH

2. FROM-ADDR,TO-ADDR,OPERATION,FROM-LENGTH

3. FROM-ADDR,TO-ADDR,FROM-LENGTH

4. FROM-ADDR,TO-ADDR,EDIT-MASK
```

FROM-ADDR can be:	*iadr*	= input area address
	wadr	= work area address
	lit	= literal / constant
TO-ADDR can be:	*oadr*	= output area address
	wadr	= work area address

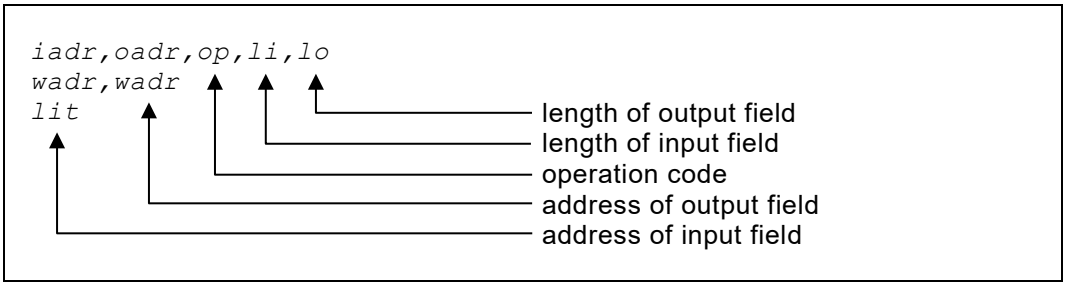
OPERATION is an operation code

LENGTH is always given in bytes

EDIT-MASK is an edit literal

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Format 1



This format is used for all instructions that can have two lengths, e.g. processing of packed fields.

Format 2

```
iadr,oadr,op,l
wadr,wadr
lit
```

This format is used for all instructions needing only one length, e.g. certain move operations.

Format 3

```
iadr,oadr,l
wadr,wadr
lit
```

Special format for simple move operation.

Format 4

```
iadr,oadr,editmask
wadr,wadr
```

Special format for editing operations.

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Literals / Constants

`C'....'` `'....'`	**Character constants** The value is the character string between the two apostrophes. The format attribute `C` can be omitted. An apostrophe in the character string can be defined by doubling it. e.g. `C'JOHN''S'`
`X'....'`	**Hexadecimal constant** The hexadecimal characters between the apostrophes are taken as a constant. e.g. `X'4040F1'`
`P'....'`	**Packed constant** The decimal digits between the apostrophes are converted to packed format and used as an arithmetic value. A minus sign can be defined, and if it is missing, the value is considered to be positive. e.g. `P'-10'` `P'10-'`
`F'....'`	**Fullword constant** The decimal digits between the apostrophes are converted to binary format and used as an arithmetic value. A minus sign can be defined, and if it is missing, the value is considered to be positive. A fullword definition always results in a 4 byte binary constant. e.g. `F'-100'` `F'100-`

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Simple Move Operation

```
iadr,oadr,l
wadr,wadr
lit
```

- *iadr* is logically moved to *oadr*.
  The length refers to bytes.

```
1,1,80
180,25,30
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
5000,7100,100
6230,6510,8
```

- Instead of *iadr* **any type of constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
C'QPAC',1,4
C'QPAC',1

P'125',6006,2
P'125',6006

X'40',7000,1
X'40',7000

F'1',5000,4
F'1',5000
```

- If no length is specified on simple move operations **a length of 1**is assumed.

```
6010,6020 [,1]
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Boolean Operations

Boolean AND

```
iadr,oadr,AND,l
wadr,wadr
C' '
X' '
```

- The AND function is applied to *iadr* and *oadr*.
- The result is stored in *oadr*.
- The length refers to bytes.
- If no length is defined, **a length of 1** is assumed.

```
                          1 + 1 = 1
1,7000,AND,100            1 + 0 = 0        both 1 => 1
X'40',5000,AND            0 + 1 = 0        all other cases => 0
                          0 + 0 = 0
```

Boolean OR

```
iadr,oadr,OR,l
wadr,wadr
C' '
X' '
```

- The OR function is applied to *iadr* and *oadr*.
- The result is stored in *oadr*.
- The length refers to bytes.
- If no length is defined, **a length of 1** is assumed.

```
                          1 + 1 = 1
1,7000,OR,100             1 + 0 = 1        both 0 => 0
X'F0',5700,OR             0 + 1 = 0        all other cases => 1
                          0 + 0 = 0
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Boolean XOR

```
iadr,oadr,XOR,l
wadr,wadr
C' '
X' '
```

- The XOR function is applied to *iadr* and *oadr*.
- The result is stored in *oadr*.
- The length refers to bytes.
- If no length is defined, **a length of 1** is assumed.

```
                          1 + 1 = 0
1,7000,XOR,100            1 + 0 = 1      both equal => 0
X'F0',5100,XOR            0 + 1 = 1      all other cases => 0
                          0 + 0 = 0
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Algebraic Operations

Addition

```
iadr,oadr,A,l,l
wadr,wadr
P' '
```

- *iadr* is **added** to *oadr*.
- Both fields are packed fields.
- The lengths refer to bytes.

```
1,5,A,8,8
10,2,A,8,4
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,6050,A,8,8
100,6200,A,5,8
```

- Instead of *iadr* a **packed constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
P'125',6020,A,,8
P'125',6020,A,2,8
P'125-',6030,A,,8
```

- If no length is specified on operations with packed fields **a length of 8** is assumed.

```
6010,6020,A    [,8,8]
100,6010,A,5   [,8]
P'125',6020,A  [,2,8]
```

- This length default value is valid for all operations that process packed fields: A, S, M, D, ZA, CB, CD, P, U.

- Exception: packed literal **without** operation code:

```
P'100',60
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Subtraction

```
iadr,oadr,S,l,l
wadr,wadr
P' '
```

- *iadr* is **subtracted** from *oadr*.
- Both fields are packed fields.
- The lengths refer to bytes.

```
1,5,S,8,8
10,2,S,8,4
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,6050,S,8,8
100,6200,S,5,8
```

- Instead of *iadr* a **packed constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
P'125',6020,S,,8
P'125',6020,S,2,8
P'125-',6030,S,,8
```

- If no length is specified on operations with packed fields **a length of 8** is assumed.

```
6010,6020,S    [,8,8]
100,6010,S,5   [,8]
P'125',6020,S  [,2,8]
```

- This length default value is valid for all operations that process packed fields: A, S, M, D, ZA, CB, CD, P, U.

- Exception: packed literal **without** operation code:

```
P'100',60
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Multiplication

```
iadr,oadr,M,l,l
wadr,wadr
P' '
```

- *oadr* is **multiplied** by *iadr*.
- The result (product) is stored in *oadr*.
- Both fields are packed fields.
- The lengths refer to bytes.
- Overflow data may be truncated in the result field **without warning**.

```
5,10,M,8,8
15,22,M,4,8
```

Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,6050,M,8,8
100,6200,M,5,8
```

- Instead of *iadr* a **packed constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
P'125',6020,M,,8
P'125',6020,M,2,8
P'125-',6030,M,,8
```

- If no length is specified on operations with packed fields **a length of 8** is assumed.

```
6010,6020,M    [,8,8]
100,6010,M,5   [,8]
P'125',6020,M [,2,8]
```

- This length default value is valid for all operations that process packed fields: A, S, M, D, ZA, CB, CD, P, U.

- Exception: packed literal **without** operation code:

```
P'100',60
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Division

```
iadr,oadr,D,l,l
wadr,wadr
P' '
```

- *oadr* is **divided** by *iadr*.
- The result (quotient) is stored in *oadr*, *iadr* is the divisor.
- A remainder is not available.
- Both fields are packed fields.
- The lengths refer to bytes.
- **Division by zero results in a quotient of 0**. The length of the divisor should not exceed 8 bytes, to prevent truncation of leading digits.

```
5,10,D,8,8
15,22,D,4,8
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,6050,D,8,8
100,6200,D,5,8
```

- Instead of *iadr* a **packed constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
P'125',6020,D,,8
P'125',6020,D,2,8
P'125-',6030,D,,8
```

- If no length is specified on operations with packed fields **a length of 8** is assumed.

```
6010,6020,D    [,8,8]
100,6010,D,5   [,8]
P'125',6020,D  [,2,8]
```

- This length default value is valid for all operations that process packed fields: A, S, M, D, ZA, CB, CD, P, U.

- Exception: packed literal **without** operation code:

```
P'100',60
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Conversion Operations

Packed to Binary Conversion

```
iadr,oadr,CB,l,l
wadr,wadr
P' '
```

- Convert **packed field** *iadr* **to binary field** *oadr*.
- The *iadr* field length can be between 1 and 16 bytes.
- The *oadr* field has a length of between 1 and 4 bytes and need not be aligned to a word boundary.

```
5,10,CB,8,4
10,22,CB,16,4
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,6050,CB,8,4
100,5200,CB,5,4
```

- Instead of *iadr* a **packed constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
P'125',7250,CB,,4
P'125',7250,CB,2,4
```

- If no length is specified, **a length of 8** is assumed for packed fields, and **a length of 4** is assumed for binary fields.

```
6010,5010,CB    [,8,4]
100,5110,CB,5   [,4]
P'125',5020,CB  [,2,4]
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Binary to Packed Conversion

```
iadr,oadr,CD,l,l
wadr,wadr
F' '
```

- Convert **binary field** *iadr* **to packed decimal field** *oadr*.
- The *iadr* field has a length of between 1 and 4 bytes and need not be aligned to a word boundary.
- The *oadr* field length can be between 1 and 16 bytes.

```
5,10,CD,4,8
10,22,CD,4,5
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
5000,6000,CB,4,8
100,6200,CB,4,4
```

- Instead of *iadr* a **fullword constant** can be defined.

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
F'1',6000,CD,,8
F'1',6020,CD,4,8
```

- If no length is specified, **a length of 4** is assumed for binary fields, and **a length of 8** is assumed for packed fields.

```
5010,6010,CD    [,4,8]
100,6110,CD,4   [,8]
F'1',6020,CD    [,4,8]
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Pack Operation

```
iadr,oadr,P,l,l
wadr,wadr
```

- **Pack** the zoned-decimal field in *iadr* into *oadr*
- The sign in *oadr* is set to F after operation unless the zoned-decimal value is negative (sign D).
- Packing a blank field is therefore possible.

```
5,10,P,4,8
10,22,P,4,5
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
5000,6000,P,16,8
100,6200,P,8,4
```

- Instead of *iadr* a **zoned-decimal constant** can be defined (character constant with numeric contents).

- If the length of the operation is determined by a literal, the length need not to be specified in the instruction. However, if the length is specified, it must be in accordance with the defined literal.

```
'123',6000,P,,8
C'123',6000,P,,8
'123',6020,P,3,8
```

- If no length is specified, **a length of 16** is assumed for zoned-decimal fields and **a length of 8** is assumed for packed fields.

```
5010,6010,P    [,16,8]
100,6110,P,4   [,8]
'123',6020,P   [,3,8]
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Unpack Operation

```
iadr,oadr,U,l,l
wadr,wadr
```

- **Unpack** the packed field *iadr* into zoned-decimal field *oadr*.

```
5,10,U,8,16
10,22,U,2,3
```

- Instead of *iadr* and *oadr* a *wadr* can be defined, *wadr* being a working storage address from 5000 - 20999.

```
6000,5000,U,8,16
6215,100,U,4,8
```

- If no length is specified, **a length of 16** is assumed for zoned-decimal fields and **a length of 8** is assumed for packed fields.

```
6010,5010,U    [,8,16]
100,5110,U,4   [,16]
```

Zero Add Operation

```
iadr,oadr,ZA,l,l
wadr,wadr
P' '
```

- *iadr* is a packed field and is **moved arithmetically** to *oadr*.
- The contents of *oadr* prior to this operation are lost.
- Lengths refer to bytes.
- Instead of *iadr* and *oadr* a *wadr* can be defined.
- Instead of *iadr* a **packed constant** can be defined.

```
1,5,ZA,8,8
6000,5000,ZA
P'0',5100,ZA,1,8
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Hexadecimal Conversion

```
iadr,oadr,CX,l
wadr,wadr
```

- *iadr* is converted to a **hexadecimal printable format** in *oadr* in order to, for example, print it.
- *oadr* length is twice the *iadr* length.
- The length refers to bytes.

```
'QPAC',5000                    results in:
TO-O1
5000,1,CX,4                    1...5....10...5....20..
PUT-O1                         D8D7C1C3
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Editing Operations

User Specified Edit Masks

```
iadr,oadr,E'literal'
wadr,wadr
```

- The contents of the packed *iadr* field are **edited** into *oadr* using the literal as edit mask.
- The edit mask determines the length of the output and therefore must take into account the length of the input *iadr* field.
- The first character in the mask is the **fill character**.
- Digit positions are defined by a 9, but positions on which zero suppression is required are defined by a Z instead of a 9.
- Punctuation symbols or any other required symbols can be included in the mask, as can the minus sign (-) in the right most mask position.

```
15,10,E' 999999'
6000,1,E'$ZZZZZ9-'
3,8,E' ZZ.ZZ9,99-'
CDATE,90,E' 99.99.99'
CTIME,100,E' 99:99:99'
```

User Defined Hexadecimal Edit Masks

```
iadr,oadr,EX'literal'
wadr,wadr
```

- Same editing rules as under E.
- The mask is defined with hexadecimal characters according to assembler conventions.

```
15,10,EX'40F9F9F9F9F9F9'
6000,1,E' 999999'
```

Predefined Edit Masks

```
iadr,oadr,EDA,l1,l2
          EDAZ
          EDAS
wadr,wadr
                    Mask type A = ▲▲▲▲▲▲–
```

- The packed *iadr* field is edited into *oadr* according to mask type A.
- This extended edit operation enables editing without punctuation.
- A negative packed field results in a '–' sign stored in the right most position of *oadr*.
- **Zero suppression** takes place if either EDAZ or EDAS is specified:
  EDAS results in zero suppression up to, **but not including** the last digit.
  EDAZ results in zero suppression up to, **and including** the last digit.
- *l1* is the length of *iadr* in bytes, *l2* is the number of bytes, **without the sign,** that are needed for the edited value starting at the right most digit.
- If no length is specified for *l1*, a **length of 8** is assumed.
- If *l2* is missing, the length is calculated according to *l1*.

```
10,65,EDAZ,5,4
```

```
pos.10        =  X'019376219D'  l1=5
fully edited  =     19376219-
pos.65        =           219-  l2=4
```

```
iadr,oadr,EDB,l1,l2
wadr,wadr
                    Mask type B = ▲▲.▲▲.▲▲.▲▲
```

- The packed *iadr* field is edited into *oadr* according to mask type B.
- This extended edit operation enables editing in groups of 2 decimal digits, separated by a full stop.
- A negative *iadr* value is not marked as such.
- Zero suppression is not possible.
- *l1* is the length of *iadr* in bytes, *l2* is the number of bytes required for the edited value, starting at the right-most digit, including internal punctuation characters.
- If no length is specified for *l1*, a **length of 8** is assumed.
- If *l2* is missing, the length is calculated according to *l1*.

```
10,65,EDB,5,8

                        pos.10        =  X'019376219D' l1=5
                        fully edited  =   19.37.62.19
                        pos.65        =      37.62.19  l2=8
```

The editing rules for the following edit mask are the same as those defined for EDB**:**

```
iadr,oadr,EDC,l1,l2
wadr,wadr
                    Mask type C = ▲▲:▲▲:▲▲:▲▲:▲▲
```

The editing rules for the following edit masks are the same as those defined for EDA**:**

```
iadr,oadr,EDD,l1,l2
          EDDZ
          EDDS

                    Mask type D = ▲▲▲.▲▲▲.▲▲▲,▲▲▲
```

```
iadr,oadr,EDE,l1,l2
          EDEZ
          EDES

                    Mask type E = ▲▲▲,▲▲▲,▲▲▲.▲▲–
```

```
iadr,oadr,EDF,l1,l2
          EDFZ
          EDFS
```

Mask type F = ▲▲▲'▲▲▲'▲▲▲.▲▲–

```
iadr,oadr,EDG,l1,l2
          EDGZ
          EDGS
```

Mask type G = ▲▲▲ ▲▲▲ ▲▲▲–

```
iadr,oadr,EDH,l1,l2
          EDHZ
          EDHS
```

Mask type H = ▲▲▲,▲▲▲,▲▲▲–

```
iadr,oadr,EDI,l1,l2
          EDIZ
          EDIS
```

Mask type I = ▲▲▲.▲▲▲.▲▲▲–

```
iadr,oadr,EDK,l1,l2
          EDKZ
          EDKS
```

Mask type K = ▲▲▲▲▲▲▲▲▲▲,▲▲–

Special Value Instructions

General format

At initialization time, QPAC generates **special registers**, whose contents may be of interest to the user.

These registers cannot be altered by the user, they can only be accessed, i.e. used as input fields in an instruction. Their contents can be made available by special instructions that are syntactically simple moves.

Instead of *iadr*, the name of the special register is used, the length is defined implicitly by the name of the register.

System Date

```
DATE,oadr
     wadr                          8 bytes value
```

After executing this instruction, *oadr* or *wadr* contain the actual system date at the time the QPAC started, in the following format:

```
C'DD.MM.YY'
```

System Time

```
TIME,oadr
     wadr                          8 bytes value
```

After executing this instruction, *oadr* or *wadr* contain the actual system time at the time the QPAC started, in the following format:

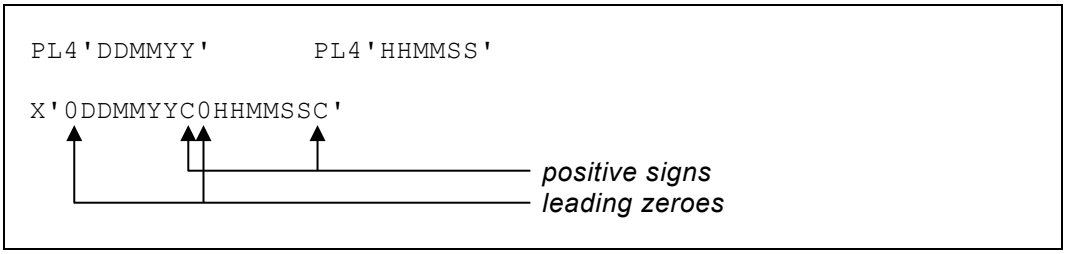
```
C'HH:MM:SS'
```

For further, more detailed information please consult Chapter 7: The High-Level Format Instruction SET.

Current Date/Time

```
CDTIME,oadr
       wadr                          2 x 4 bytes packed (8 bytes)
```

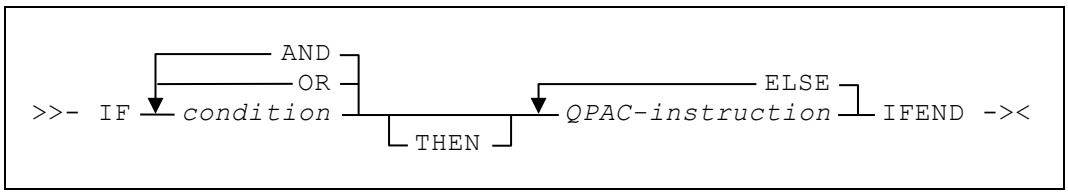
After executing this instruction, the first 4 bytes field contains the current date, the second 4 bytes field the time, in the following packed format:



Attention: `CDTIME` is only valid with the basic instruction format!

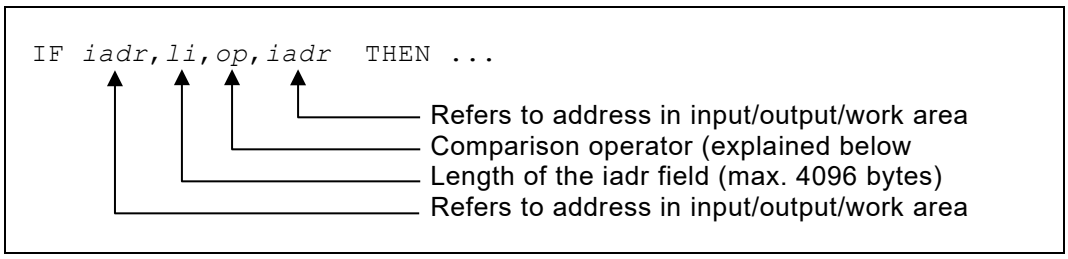
The IF THEN ELSE Instruction (Basic Format)

General Format of the Condition Instruction



For further, more detailed information please consult Chapter 8: Logic Control Commands.

Format 1 - Logical Comparison (CLC)

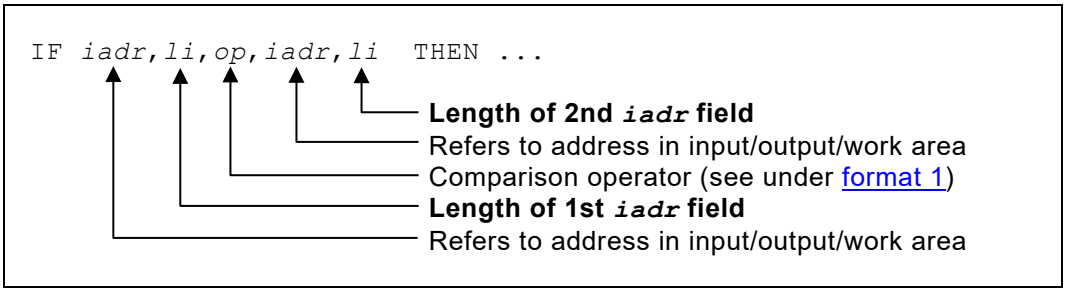


- Valid comparison operators:

EQ	= equal
NE	= not equal
GT	= greater than
LT	= less than
GE	= greater or equal
LE	= less or equal

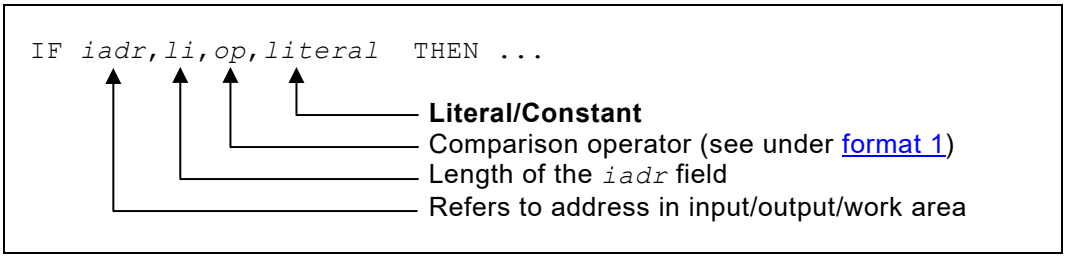
- Instead of an *iadr*, a *wadr* or an *oadr* can always be defined.

Format 2 - Arithmetic Comparison of Packed Fields (CP)



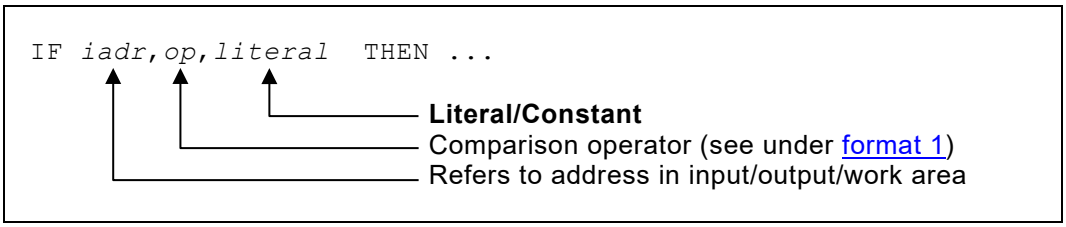
- If a **second length is defined**, QPAC assumes a **comparison of packed fields**.

Format 3 - Comparison with Constants (Length Specified)



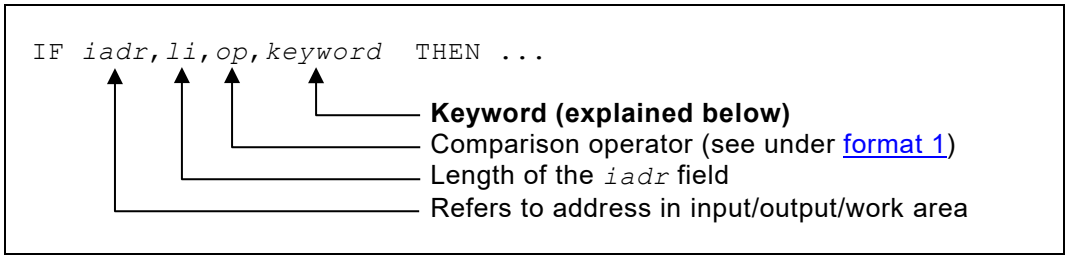
- The second comparison operand can be defined as a **literal**.
- The length specified must correspond to that of the literal.
- In case of a packed literal, the length is specified as required.

Format 4 - Comparison with Constants (Length Not Specified)



- If the literal used is character or hexadecimal, the first length and its positional comma can be omitted. If the length is defined, it must correspond to that of the literal (see format 3).
- When using packed literals, the length must be defined, otherwise 8 is assumed for the *iadr* field.
  When using 'full word' literals, a length of 4 bytes is assumed.
- the comparison is made logically or arithmetically, according to the rules specified under format 3.

Format 5 - Logical Comparison with Keyword



- Valid keywords are:

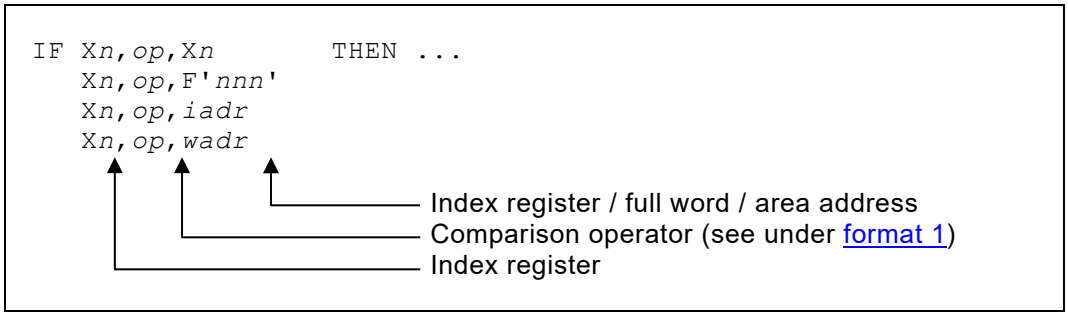
NUMERIC	= zoned decimal format
SPACE	= blank
ZERO	= zoned decimal format

- The compared field must be in character or zoned decimal format, the default length value is 1 byte, maximum length 4095 bytes.

PACKED	= packed format

- Field contents are checked to see whether they are correctly packed, the default length value is 8.

Format 6 - Binary Arithmetic Comparison



- The contents of an **index register** can be compared with another index register, with a full word literal, with a working storage area, or with an input field.

- A length should not be defined; a **value of 4** is assumed.

For further, more detailed information please consult Chapter 8: Logic Control Commands.

```
IF 10,EQ,C'1' THEN
|  IF 20,EQ,C'2' THEN
|  |  IF 30,EQ,C'3' THEN
|  |  |  1,1,80
|  |  IFEND
|  |  P'1',6020,A
|  IFEND
|  P'1',6010,A
IFEND
```

```
IF 10,EQ,C'1' THEN
|
|  IF 20,EQ,C'2' THEN
|  |
|  |  IF 30,EQ,C'3' THEN
|  |  |
|  |  |  IF 40,EQ,C'4' THEN
|  |  |  |
|  |  |  |  IF 50,EQ,C'5' THEN
|  |  |  |  |
|  |  |  |  |  1,1,1
|  |  |  |  |  2,2,2
|  |  |  |  |  4,4,4
|  |  |  |  ELSE
|  |  |  |  |
|  |  |  |  |  8,8,8
|  |  |  |  |
|  |  |  |  IFEND
|  |  |  |
|  |  |  ELSE
|  |  |  |
|  |  |  |  IF 60,EQ,C'6' THEN
|  |  |  |  |
|  |  |  |  |  1,1,1
|  |  |  |  |  2,2,2
|  |  |  |  |  4,4,4
|  |  |  |  ELSE
|  |  |  |  |
|  |  |  |  |  8,8,8
|  |  |  |  |
|  |  |  |  IFEND
|  |  |  |
|  |  |  IFEND
|  |  |
|  |  ELSE
|  |  |
|  |  |  1,1,1
|  |  |
|  |  IFEND
|  |
|  IFEND
|
IFEND
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z